Rapport de Stage Ouvrier

Stagiaire : Nathan SEIGNOLE

10/06/2024 - 09/08/2024

The goal of this technical internship was to design an image sensing system using two LVDS-capable sensors. It would then send an image or video stream to a computer. I have chosen to implement this using the onsemi PYTHON1300 sensor and the MYIR Z-Turn v2 FPGA board, as this board supports multiple LVDS interfaces and contains a powerful system-on-chip.
The goal being to make a system that operates the sensors, we design a board (PCB) that integrates both the FPGA board and the two sensors. However, as the sensors do not have sockets and are supposed to be soldered directly upon a PCB, I have instead decided to design two boards, a main one and a breakout board for the sensors, so that if any errors are detected on the main board after after soldering the components, the sensors need not be discarded. The resulting PCBs will be printed and assembled by an external company.
In parallel to that, I have developed a custom IP that controls the sensors following commands sent by the computer to the system. Namely, the IP can turn on and off the sensors and write to their configuration registers by a custom SPI interface.
Then, I have developed a custom IP to retrieve pixel values from the PYTHON1300 sensors. This involves deserializing the data from the LVDS channels, which poses a number of problems as they are supplied faster than the FPGA clock can handle. The solution was to use FPGA peripherals called the ISERDES, that can serialize or deserialize incoming data and send it to the FPGA fabric at a much slower clock speed, instead of deserializing inside of our IP and thus causing a number of timing problems.
Once the main design has been elaborated, I moved on to starting the code that would control the HDL design : as the board used integrates both a FPGA and processor, one can write an C application code to control advanced functions of the system. The goal here is to have a code that would respond to commands sent to its UART by a computer and send the image stream on the Ethernet or HDMI ports. Unfortunately, this part has not been finished due to a lack of time.

Le but de ce stage technique était de réaliser un système d'acquisition d'image en utilisant deux capteurs supportant le LVDS. Il enverrait ensuite un flux vidéo vers un ordinateur. J'ai choisi d'implémenter ce système avec le capteur PYTHON1300 de onsemi et la carte FPGA MYIR Z-Turn v2, qui supporte plusieurs interfaces LVDS et contient un *system-on-chip* puissant.
On commence par dessiner un circuit imprimé (PCB) qui intègre la carte FPGA et les deux capteurs. Comme les capteurs n'ont pas de support et sont supposés être soudés directement en surface d'un PCB, j'ai décidé de réaliser deux circuits : un PCB principal et deux *breakout boards* pour les capteurs, pour éviter de devoir jeter les capteurs une fois soudés en cas d'erreur sur le plus grand PCB principal. Les cartes seront imprimés par une entreprise extérieure à l'institut.
En parallèle de cela, j'ai développé une IP personnalisée qui contrôle les capteurs selon les commandes envoyées depuis l'ordinateur vers le système. Précisément, l'IP permet de les allumer et les éteindre, ainsi qu'écrire dans leurs registres de configuration en utilisant une interface SPI personnalisée.
Une autre IP a été réalisée qui récupère les pixels envoyés depuis les capteurs PYTHON1300. Cela induit de désérialiser les données venant des canaux LVDS, ce qui pose certains problèmes d'implémentation car ces données arrivent généralement plus vite que ce que l'horloge du FPGA peut fournir. La solution à ces problèmes est d'utiliser des périphériques présents autours du FPGA appelés « ISERDES », qui peuvent sérialiser ou désérialiser des données entrantes et les envoyer au sein de la logique du FPGA bien plus lentement, au lieu de désérialiser les canaux au sein de l'IP et causer de nombreux problèmes de timing.
Après que le design principal soit élaboré, je suis passé à l'étape du code : comme la carte contient un *system-on-chip* qui inclut un processeur et un FPGA, on peut écrire une application en C qui contrôle les fonctions avancées du système. L'objectif ici est d'avoir un code qui répond aux commandes envoyées vers son UART par un ordinateur et distribue les flux d'image vers les ports Ethernet ou HDMI. Malheureusement, à cause d'un manque de temps, cette partie n'a pas été achevée.

Le stagiaire souhaite ici remercier M. Nizar OUARTI, qui a donné l’opportunité de réaliser ce stage , beaucoup d’aide, ainsi qu’un projet fort intéressant. Il souhaite également remercier son ami, camarade, et collègue Youssef CHAOUI AZIZ, pour son optimisme et son soutien durant toute cette période. Enfin, l’équipe de l’ISIR pour leur accueil, et particulièrement les stagiaires du Hall SIMA, qui ont su créer une atmosphère plaisante et accueillante.

Table des matières

Abréviations :

- FPGA : *Field Programmable Gate Array*
- LVDS : *Low-Voltage Differential Signaling*
- PCB : *Printed Circuit Board*
- SPDT : *Simple Pole, Double Throw*
- CLCC : *Ceramic Leadless Chip Carrier*
- SoC : *System on Chip*
- GPIO : *General Purpose In-Out*
- HDL : *Hardware Description Language*
- IP : *Intellectual Property*
- DMA : *Direct Memory Access*
- FIFO : *First-In First-Out*

Précision supplémentaire : « Carte FPGA » désigne l'intégralité de la carte MYIR Z-Turn V2, tandis que « FPGA » ne désigne que la partie programmable.

1 Contexte

Ce stage technique a été réalisé au sein de l'**ISIR**, l'Institut des Systèmes Intelligents et de Robotique, présent au sein de la faculté des sciences de Sorbonne Université. Il s'agit d'un institut de recherche qui touche à des domaines variés tels que les mathématiques, la robotique, ou l'intelligence artificielle.

Sous la tutelle de M. Nizar OUARTI, son objectif est de réaliser un système de captation d'image à partir de deux capteurs, et de transmettre un flux vidéo haut débit vers un ordinateur, en vue d'un traitement d'image ultérieur. Même si toutes sortes de matériel sont adaptés à ce projet, on a choisi de s'orienter vers une implémentation sur **carte FPGA**, en raison de l'adaptabilité de cette technologie et mon expérience professionnelle dessus.

2 Travail accompli

2.A Choix du matériel

2.A.1 Capteur

La toute première partie du projet a été de déterminer quel capteur d'image utiliser. La seule spécification donnée soit qu'il communique en signal LVDS, un type de signal différentiel qui permet de préserver l'intégrité de la donnée transportée.

Après recherches sur le site de l'A3 (*Association for Advancing Automation*), j'ai pu obtenir une liste de caméras qui pourraient correspondre au projet – on recherche un produit de type « caméra-bloc » ou « capteur d'image ». La majorité des produits listés sont malheureusement introuvables chez les revendeurs, et doivent être commandés par devis sur le site du fabricant. Cela veut bien souvent dire que le prix est très élevé, ou que les commander en petites quantités est difficile.

En cherchant donc des produits similaires chez les revendeurs (ici, Mouser, Farnell, et Digikey), j'ai trouvé le **PYTHON1300**, fabriqué par **onsemi**. Il supporte le LVDS sur plusieurs canaux parallèles et permet facilement de fixer une lentille S-Mount standard. Également, son prix est raisonnable, et sa résolution est haute par rapport à la taille du capteur (1.3 Mégapixels). Il n'existe pas de support mécanique adapté du format du PYTHON1300 : il faudra donc le souder directement sur carte. La référence utilisée ici est **NOIP1SN1300A-QTI**, c'est-à-dire monochrome et à 4 canaux LVDS : une horloge simple de 72 MHz doit lui être fournie, qui sera générée par une des PLL de la carte FPGA.

En plus du capteur, on a choisi la lentille M12B1618IRM12 et son support MIPI LHLD12 de chez **Basler**, un autre fabricant de capteurs d'image. Les deux correspondent au format M12, légèrement plus grand que la taille du PYTHON1300, et devraient donc s'insérer sans problème autour du capteur.

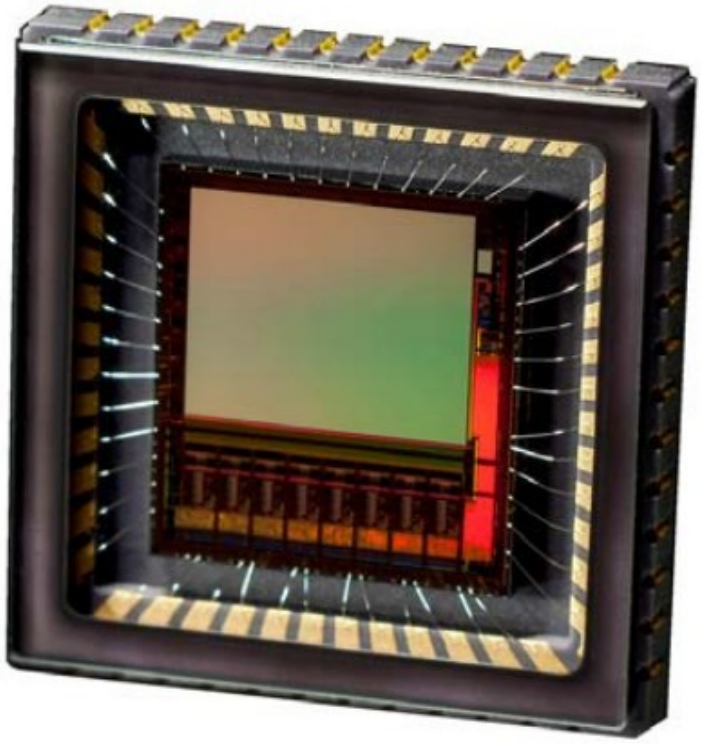

Fig. 1 : vue du PYTHON1300.

2.A.2 Carte FPGA

Pour pouvoir assurer une communication LVDS stable, il est nécessaire que la carte choisie comporte des canaux différentiels dédiés. Un capteur PYTHON1300 utilise 6 paires LVDS pour communiquer : la carte doit donc comporter **au moins 12 paires LVDS**.

Parmi les cartes FGPA connues pour bien fonctionner avec le LVDS, plusieurs noms se sont démarqués sur les sites Internet : côté Altera, ce sont les cartes de développement basées sur les

FPGAs **MAX 10** et **Arria II GX**. Côté Xilinx, la FPGA **Kintex-7** et le system-on-chip (SoC) **Zynq**, qui contient un processeur Cortex-A9 et une FPGA Artix-7 dans la même puce. Ayant personnellement plus d'expérience sur les logiciels du fabricant Xilinx que ceux d'Intel, et ayant également déjà développé pour SoC, je me suis penché sur les cartes Zynq.

La cartes basées sur le Zynq les plus populaires sont fabriquées par **Digilent** : nommément, la **Zybo** et la **Zedboard**. Ces cartes sont cependant très chères, et presque « trop complexes » pour notre cas : c'est pour cela qu'en cherchant du côté d'autres fabricants, j'ai trouvé une carte bien moins chère et plus petite : la **Z-Turn v2**, de la société **MYIR Tech**.

La MYIR Z-Turn v2 est basée sur le Zynq-7020, une version qui dispose de plus d'élements logiques et de mémoire que le Zynq de base. Elle dispose de 24 paires LVDS dédiées, de ports USB, Ethernet et HDMI, et est programmable par carte SD. Sur **Vivado** et **Vitis**, les logiciels propriétaires de AMD/Xilinx, qui servent respectivement à programmer la partie FPGA et la partie processeur, la carte n'est pas officiellement supportée : en revanche, une grande partie du *pinout* de la Z-Turn V2 est partagée avec sa version précédente, la Z-Turn, dont les fichiers de support sont disponibles au public.

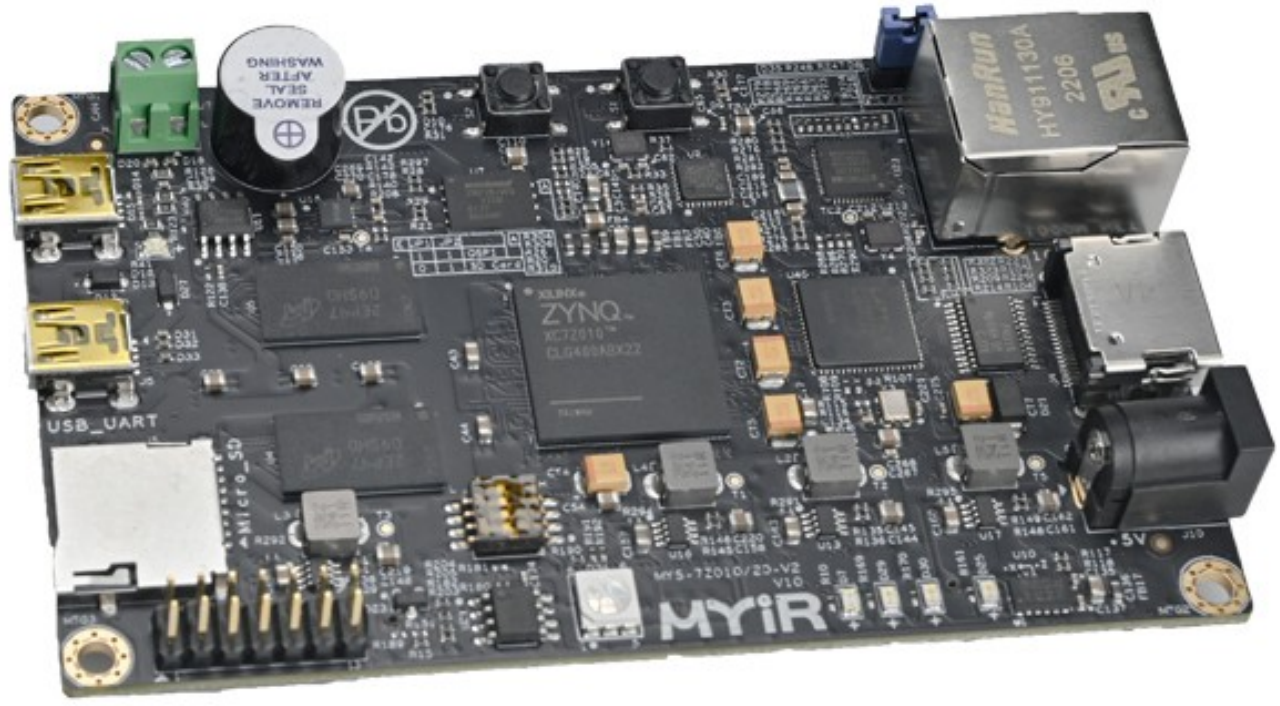

Fig. 2 : vue de la Z-Turn V2.

2.B Partie Circuit

Dans ce projet, la « Partie Circuit » revient à réaliser le circuit imprimé du système, ou PCB. Ce PCB devra accueillir les deux capteurs, la carte FPGA, ainsi que tous les composants supplémentaires nécessaires au bon fonctionnement du circuit.

2.B.1 Composants supplémentaires

Deux interrupteurs triples SPDT, des NX3L4053. Le PYTHON1300 nécessite une procédure d'allumage et d'extinction bien précises, qui incluent d'allumer ou d'éteindre ses trois sources de courant. Par sécurité, au lieu d'utiliser des pattes GPIO pour cette fonction, et donc tirer un courant trop élevé à travers ces pattes, on a choisi d'alimenter le capteur avec des sources de courant dédiées, nommément les pattes « 3V3 » de la Z-Turn V2, et de faire passer chacun de ces courants à travers un interrupteur.

Un régulateur de tension 1.8V, de type MIC55. La Z-Turn V2 ne fournissant pas de source de tension 1.8V, et cette tension étant nécessaire pour l'alimentation des PYTHON1300, il est nécessaire d'utiliser un régulateur de tension externe.

Quatre couples de connecteurs Tiger Eye de chez Samtec (référence SFML pour les femelles et TFML pour les mâles). Ils servent à connecter le PCB principal avec les plus petites cartes contenant les capteurs. Ils ont été choisi en raison de leur blindage et leur stabilité mécanique. L'arrangement des pattes du PYTHON1300 étant plutôt asymétrique, on a choisi d'utiliser un couple de connecteurs 30-pin pour tous les signaux différentiels, et un couple 20-pin pour les autres signaux.

2.B.2 Design du PCB

Pour dessiner le schéma du circuit et l'arrangement du PCB, j'ai utilisé le logiciel **KiCad**. Le PCB sera imprimé à l'extérieur du laboratoire, par la compagnie JLCPCB, puis assemblé par un autre intermédiaire.

Les objectifs principaux de ce PCB sont, par ordre de priorité :

1. Transmettre les signaux LVDS des capteurs vers la carte de la manière la plus intègre possible
2. Synchroniser les *triggers* des deux capteurs pour permettre une prise de vue simultanée

3. Permettre une configuration rapide des capteurs

4. Placer les capteurs à 10 cm l'un de l'autre

L'objectif N°1 nécessite déjà de faire quelques recherches pour trouver des recommandations de design pour accommoder au mieux des paires LVDS. C'est un sujet très vaste, et beaucoup de documentation a été consultée pour renseigner de design : elle possède sa propre section dans la bibliographie.

La décision a été prise très tôt de réaliser un PCB sous forme de "shield", qui viendrait directement se fixer aux connecteurs GPIO de la carte Z-Turn V2, pour éviter de tirer des câbles qui introduiraient du bruit dans le circuit.

Comme le circuit à réaliser est à relativement hautes fréquences (360 MHz au maximum) et nécessitant un contrôle fin des délais de propagations, et donc de l'impédance, on a choisi une configuration à 4 couches avec contrôle d'impédance, noté JLC04161H-7628 sur le site de JLCPCB.

En suivant les recommandations d'Altera et de Texas Instruments, l'ordre des couches choisi est, de haut en bas : LVDS, VCC, GND, puis « autres signaux » - cela permet de protéger les signaux différentiels de l'influence des signaux simples. En revanche, comme dans notre cas il n'y a pas qu'un seul VCC mais six différents, et qu'ils peuvent être activés ou désactivés, on a choisi l'arrangement LVDS, GND, VCC, et autres signaux. La couche VCC (In2.Cu) un plan d'alimentation +3.3V générique pour les composant de la couche du bas (B.Cu) - et la couche GND (In1.Cu) contient les pistes d'alimentation pour les capteurs. Enfin, la couche du haut (F.Cu) contient les signaux hautes-fréquences : les paires LVDS et les signaux SPI.

Une autre étape essentielle avant de commencer le design est d'intégrer les contraintes de l'imprimeur de circuit, JLCPCB. Elles peuvent êtres ensuite rajoutées directement dans les netclasses, les tailles prédéfinies, ou les contraintes du projet KiCad. Les critères particulièrement importants sont l'écart minimal entre pistes (on cherche à rapprocher les pistes d'une même paire le plus possible), la l'épaisseur minimale de piste, et l'espacement entre divers objets du circuit, notamment les trous et les composants.

Une fois ces données rassemblées, il faut maintenant trouver des dimensions de pistes qui donneraient une bonne impédance. Comme tous les paramètres du substrat sont fixés, il suffit de jouer avec la largeur et la longueur jusqu'à trouver une bonne valeur d'impédance : ici, on cherche 50Ω en signal simple et 100Ω inter-piste en différentiel. Cela permet de créer nos deux « types de pistes » ou *netclasses* personnalisées : une pour les signaux différentiels et l'autre pour les signaux simples.

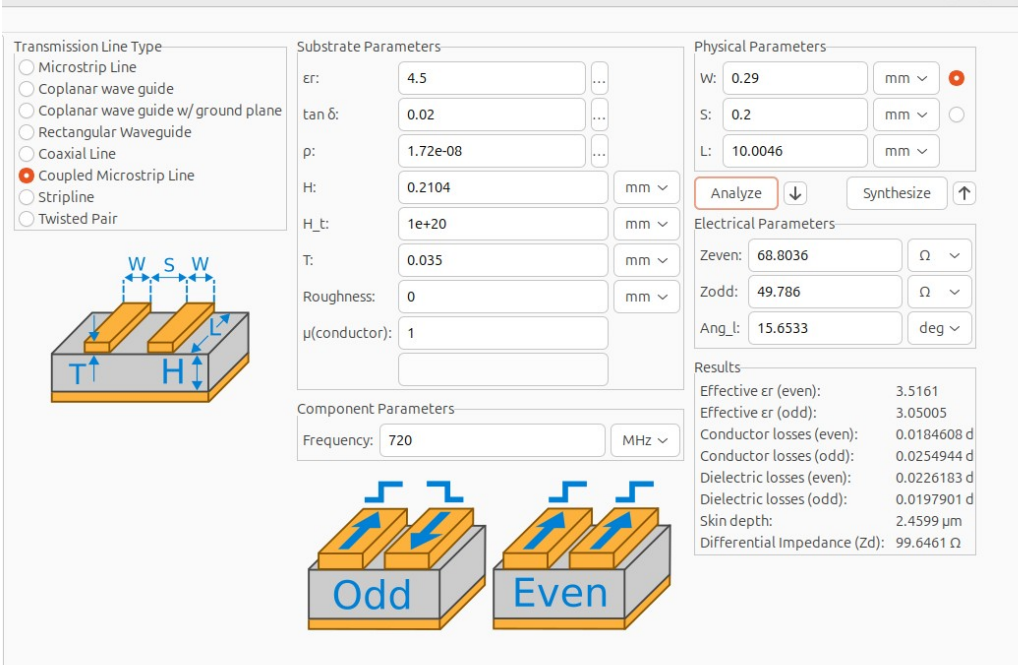

Fig. 3 : paramètres des pistes du PCB, calculés avec KiCad.

Avant de commencer le routage des pistes, il faut évidemment disposer composants sur le PCB, du moins celles dont la position est fixée par nos contraintes. Ici, les connecteurs J1 et J2 sont alignés et à exactement 53 mm l'un de l'autre. Les deux capteurs sont espacés de 100 mm, même si cette distance peut être changée légèrement si le routage devient trop compliqué.

Horizontalement, ils sont centrés autour de la pin 50 de J1, pour que les traces LVDS soient de longueur très proches chez les deux capteurs, ce qui permet de les synchroniser.

On commence donc par tracer les pistes LVDS, les plus courtes possibles. Bien sûr, comme les pattes des capteurs ne sont pas en face des pattes du connecteur au vu de leurs largeur différentes, les pistes d'un même bus ne seront pas de même longueur. Cela peut en revanche être corrigé après coup : l'objectif premier est plutôt de limiter le *skew* (décalage temporel) au sein d'une même paire en routant les pistes très simplement.

La décision a également été prise d’inverser la « polarité » des signaux LVDS venant du capteur. En effet, un problème courant des PCBs intégrant des émetteurs et récepteurs différentiels est de devoir croiser les pistes si l’ordre positif-négatif n’est pas le même sur les deux appareils. Cela nécessiterait d’ajouter un bon nombre de vias au PCB et d’induire beaucoup de bruit. Heureusement, la spécification LVDS permet d’inverser les signaux positif et négatif : il faudra juste inverser l’état logique du signal au sein du FPGA.

On a choisi de router les pistes de l'interface SPI sur la couche du haut pour plusieurs raisons : déjà, la couche à signaux simple du bas est très encombrée et garantir un tracé efficace pour cette

interface synchrone n'est pas une mince affaire. De plus, les interférences SPI / LVDS ne sont pas un problème majeur, car ces deux interfaces ne seront pas utilisées au même moment. Au moment d'une configuration par SPI, les données envoyées par le capteur par LVDS ne sont pas importantes, ou le capteur n'est pas configuré et n'envoie simplement rien sur ce canal. Des précautions nécessaires ont toutefois été prises pour éloigner ces deux interfaces, notamment au niveau du retour vers la masse.

Sur la face du bas, on dispose le régulateur de tension et les interrupteurs de façon symétrique, pour réduire la longueur des pistes et limiter les croisements. En cas de chevauchement inévitable, on privilégie de faire passer les pistes d'alimentation par le plan de masse - c'est une pratique courante qui permet une bonne dissipation thermique, et en général les signaux qui évoluent très lentement peuvent passer sans problème par le plan de masse interne.

Une fois tout cela routé, il faut relier les plans de masse et d'alimentation entièrement avec de nombreux vias, et découpler les grands puits de courant avec des capacités.

Une partie importante de notre design est de limiter le *skew* entre les signaux et leur horloge, voire entre certains signaux eux-mêmes. Comme l'impédance des pistes est constante sur tout le circuit, la vitesse des signaux est la même, et donc que pour faire arriver deux signaux au même moment, il faut rendre leurs longueurs de piste égales. L'outil « de tuning » de KiCad peut faire cela : il suffit de noter la longueur de la piste la plus longue (on ne peut pas facilement raccourcir des pistes), puis de rallonger les autres pistes avec l'outil, qui crée des "accordéons". Cette pratique permet de synchroniser toutes les paires LVDS et les signaux SPI avec leurs horloges, et de rendre les deux triggers simultanés pour permettre une prise de vue synchronisée chez les deux capteurs.

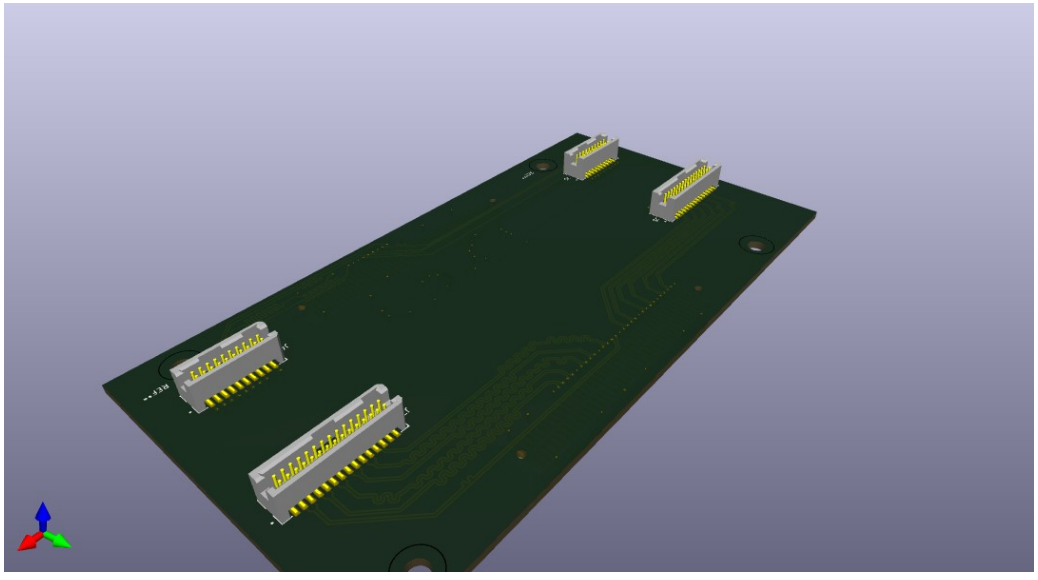

Fig. 4 : vue de la face du haut du PCB principal.

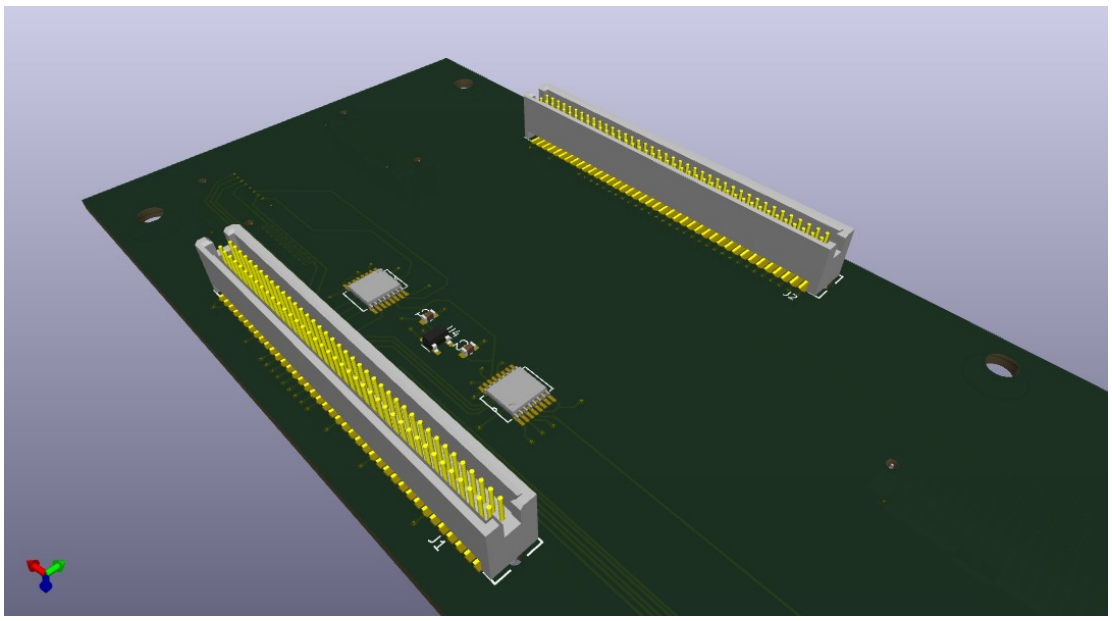

Fig. 5 : vue de la face du bas du PCB principal.

Après le design d'une première version, il a été décidé de séparer le PCB en deux : une carte principale et deux *breakout boards*, des PCBs secondaires et identiques, qui contiendraient chacune un capteur et qui viendraient se fixer au PCB principal. Les deux seraient reliés par les connecteurs mentionnés précédemment. Ainsi, toute erreur sur la carte principale ne nécessiterait pas de jeter les capteurs, qui sont soudés en CLCC et donc très difficiles à enlever d'un PCB défectueux.

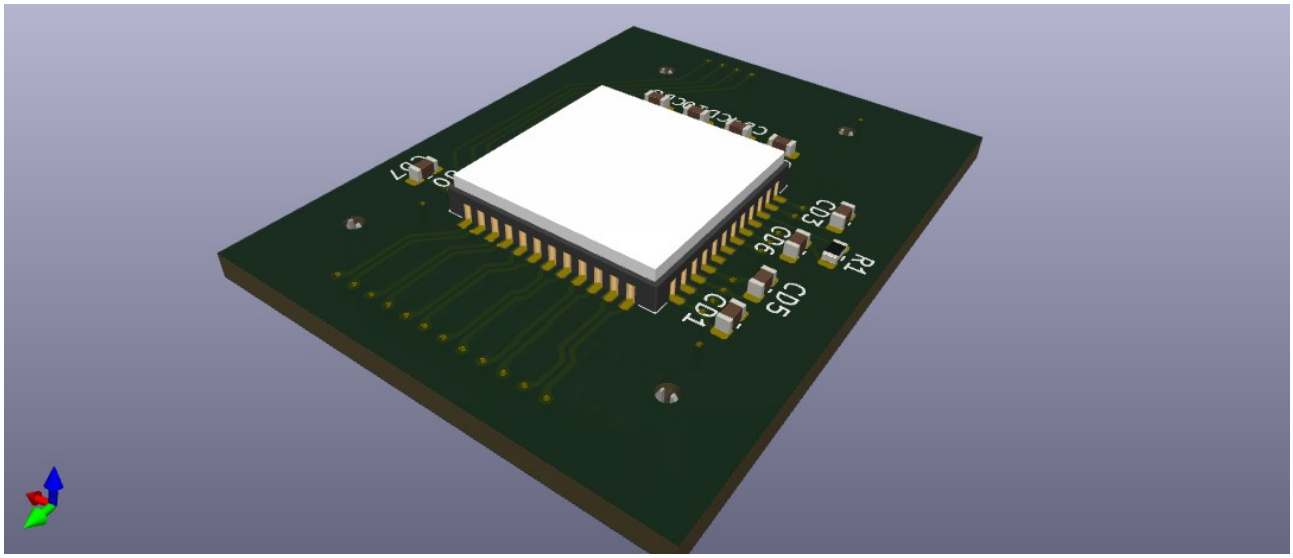

Fig. 6 : vue de la face du haut de la Breakout Board

2.C Partie HDL

Par « Partie HDL » j'entends le design du système qui va être programmé dans le SoC Zynq. Cela inclus la programmation, en VHDL, des IPs qui vont être utilisées dans ce système.

Cette partie est principalement réalisée au sein du logiciel Vivado, qui sert de simulateur VHDL et de créateur de système.

2.C.1 Design du système

Sur un SoC comme le Zynq-7020, il est possible d'utiliser le "Block Design" de Vivado, un utilitaire qui permet d'assembler différentes IP matérielles dans un seul schéma, en traçant des bus et des interfaces pour les connecter. Il permet aussi d'instancier l'IP **Zynq Processing System**, qui tourne à la fois sur le processeur du Soc et la partie programmable et qui présente une forte interconnexion avec la partie FPGA du SoC. Cela permet de programmer la carte avec des instructions complexes en C ou en C++, voire avec un OS embarqué - le Zynq utilise PetaLinux, une version de Linux embarquée crée par Xilinx.

L'objectif principal de notre design sur FPGA est d'assurer une bonne acquisition de l'image depuis les capteurs. Il y a plusieurs considérations à prendre en compte : déjà, il faudra réaliser une IP matérielle personnalisée à notre cas d'usage, qui pourra faire l'interface entre les capteurs et le processeur. Le choix de l'interface entre les deux est également important : Xilinx propose le bus AXI et ses sous types pour une interface rapide entre toute IP et le processeur . Le capteur PYTHON1300 a un débit idéal maximal de 720 Mb/s : nous choisirons donc l'interface AXI-Stream, adaptée au *streaming* continu de données à haut débit.

L'utilisation du bus AXI-Stream nécessite l'utilisation d'une IP de Xilinx importante : le DMA (Direct Memory Access). Cela veut dire que toutes les données de l'IP **noip_lvds_stream** se situent dans "le domaine mémoire", et que le bus AXI-Stream les rapatrie vers "le domaine logique".

En plus de cela, il faudra réaliser une IP de contrôle des capteurs, qui sera responsable de leur allumage (en actionnant les interrupteurs) et de leur configuration (à travers une liaison SPI). Cela sera implémenté dans l'ip **noip_ctrl**.

Pour assurer la communication à travers les protocoles demandés, on utilise plusieurs solutions : le *Processing System* contient un driver Ethernet intégré, qu'il suffit juste de connecter aux pins Ethernet de la carte. Pour la partie HDMI, en revanche, le périphérique utilisé dans la Z-Turn V2 est non-standard, et nécessite donc d'écrire une IP personnalisée. Je n'ai malheureusement pas pu aller très loin dans la conception de cette IP pour cause de manque de temps.

Le design principal (le processeur et nos IPs) est entouré d'IPs et de périphériques qui permettent son bon fonctionnement : accesseur mémoire, manipulateurs logiques, désérialiseurs, buffers, etc.

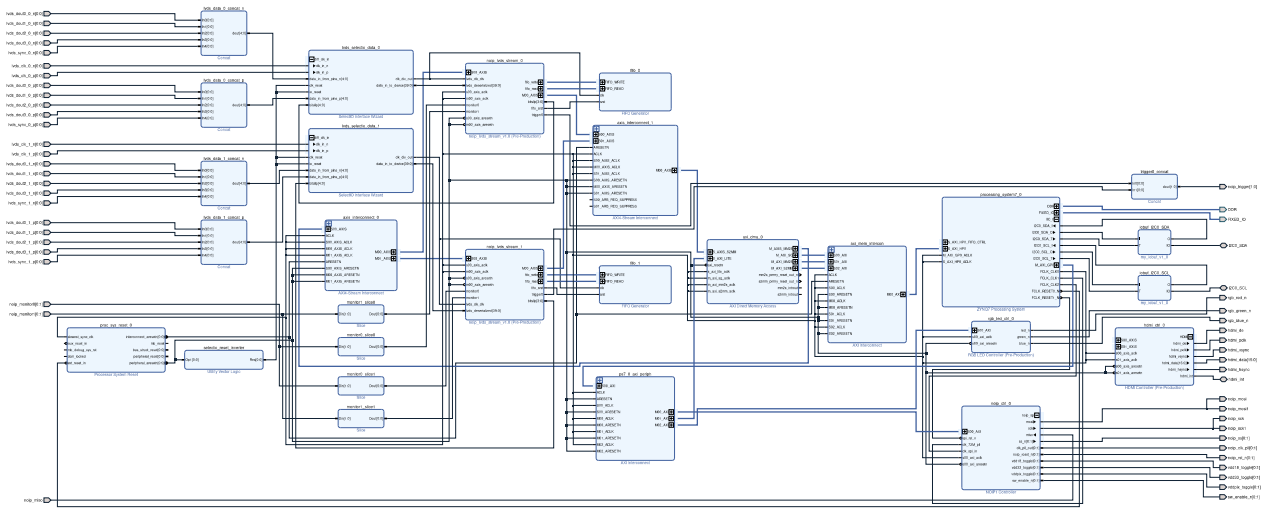

Fig. 7 : vue Block Design *du système : entrées à gauche, sorties à droite.*

2.C.2 noip_lvds_stream

L'objectif de cette entité (aussi appelée « streamer ») est de recevoir correctement les pixels depuis les capteurs et de les transmettre vers le processeur par le **bus AXI-Stream** dédié.

La version du PYTHON1300 utilisée ici peut fonctionner en deux profondeurs d'image : 10-bit ou 8bit. Comme on ignore la couleur dans notre cas d'utilisation, on a choisi d'utiliser une profondeur d'image plus petite, pour accélérer le transfert d'une image. La fréquence de l'horloge LVDS reste à 360 MHz.

Le PYTHON1300 est capable de fournir des images à très grande vitesse. Même si, dans notre cas, on contrôlera la prise de vue avec nos *triggers*, il est nécessaire de prendre en compte le temps que prend le streamer à recevoir une image et à l'envoyer, pour éviter de perdre des images en cours de route ou d'en stocker inutilement. Le calcul est le suivant : en résolution maximale, le PYTHON1300 prend une image de 1280 colonnes de 1024 pixels. L'image est divisée en « kernels » de 8 pixels de large, dont les pixels sont envoyés en simultané sur les quatre canaux LVDS : il faut donc 16 coups d'horloge LVDS par kernel. Une ligne étant composée de 160 kernels, il faut 2560 coups d'horloge ou 7.11 ms pour transmettre une ligne, sans compter le temps passé sans envoyer de données, tel que le temps d'alignement ou le *Frame Overhead Time*, etc. Au niveau de la liaison AXI-Stream, on peut transmettre 32 bits par coup d'horloge AXI, à 100 MHz. Cela veut dire qu'une ligne ne prend que 4 µs à être transmise, ce qui est bien plus rapide que le temps de réception.

Le résultat ci-dessus part de l'hypothèse que le récepteur AXI-Stream (le DMA, dans ce cas) est toujours disponible ! Ce n'est peut être pas le cas. Par sécurité, on place un **FIFO** qui servira de *buffer* entre le récepteur de pixels et le maître AXI-Stream, au cas où les pixels arrivent plus vite qu'ils ne peuvent être envoyés, par exemple si le DMA est occupé. Malheureusement, trop peu de tests ont été effectués pour explorer la multitude de cas qui peut résulter de l'utilisation de cette IP sur carte.

En plus du problème de la vitesse de transfert de l'image vers le processeur, la taille de l'image complète est également à considérer. Avec une profondeur d'image de 8 bits et la résolution maximale de 1280x1024, chaque image prend 131072 octets. Les deux images prises simultanément prendraient donc un peu moins de 3 Mo : or la RAM en bloc de la partie FPGA du système ne peut stocker que 5 Mo, et est déjà utilisée en grande partie par plusieurs IPs du design (notamment le DMA). Le processeur, de son côté, dispose de deux périphériques de RAM de 512 Mo chacun. On privilégiera donc d'envoyer les images vers le processeur le plus vite possible pour éviter de les stocker dans la partie FPGA.

Ces considérations étant résolues, le streamer est essentiellement une grande machine à état qui se cale sur les informations du canal **sync** pour déterminer la séquence d'envoi de pixels du PYTHON1300, et qui les envoie sur le bus AXI-Stream.

L'un des objectifs du streamer est de désérialiser les envois de son capteur, que ce soit de la synchronisation ou des données. Ces cinq canaux sont indépendants et envoient un bit par coup d'horloge LVDS (l'horloge la plus rapide, à 288 MHz voire 360 MHz pour le mode 10 bits). La partie déserialisation doit également pouvoir s'aligner automatiquement avec les sorties du capteur, au cas où la connexion est établie « en cours de route ». Pour implémenter cette fonctionnalité, on se sert le *training pattern* du capteur, un mot particulier (ici 0xE9) envoyé par le capteur sur ses canaux LVDS dès qu'il est au repos, et qui permet à un récepteur de s'y aligner. Il suffit donc de comparer le mot envoyé par le capteur avec le *training pattern*, et d'extraire leur décalage (ici appelé *bitslip*). Deux solutions sont possibles pour la désérialisation.

Coder un désérialiseur sur-mesure : Cela nécessite de faire rentrer l'horloge LVDS dans l'IP et d'échantillonner sur cette horloge les canaux *data* et *sync*. Le problème que cette méthode pose est qu'une grande partie de la logique de l'IP (machine à états principale, désérialiseur, fonction de détermination de bitslip) est calée sur l'horloge rapide, ce qui induit des violations de timing. Mes tentatives de générer une sous-horloge, lvds_word_ready, qui serait 8 fois moins rapide que `lvds_clk`, se sont révélées infructueuses à cause de la correction de bitslip qui fait "glisser" le front montant de cette horloge. A l'implémentation, ce signal n'est pas reconnu comme une horloge, et donc n'est pas routé à travers des pistes dédiées « pour horloge », ce qui crée des latences gigantesques. Ces erreurs de timing sont survenues lors du test de l'IP dans le système tout entier,

donc pendant une phase où je croyais avoir fini le développement du streamer. Cela m'a valu de recommencer son développement du début.

Utiliser les ISERDES : La partie programmable de la puce Zynq est un FPGA de la famille Artix-7 qui contient, à ses entrées/sorties, des périphériques de type ILOGIC (dans notre cas ISERDES) qui peuvent servir de sérialiseurs-désérialiseurs génériques. Cela permet d'isoler la logique rapide aux pins d'entrée, loin de la logique principale, et donc de ne pas induire d'importantes erreurs de timing en la faisant traverses tout le FPGA. Il suffit ensuite d'utiliser l'horloge divisée par 8 en sortie des SERDES pour la logique du streamer.

J'ai également remarqué qu'utiliser les SERDES permet également à Vivado de placer les streamers et leurs FIFOs très près des ILOGIC, donc des entrées/sorties, ce qui réduit les potentielles erreurs de latence qui pourraient survenir quand les streamers sont placés plus près de la partie processeur.

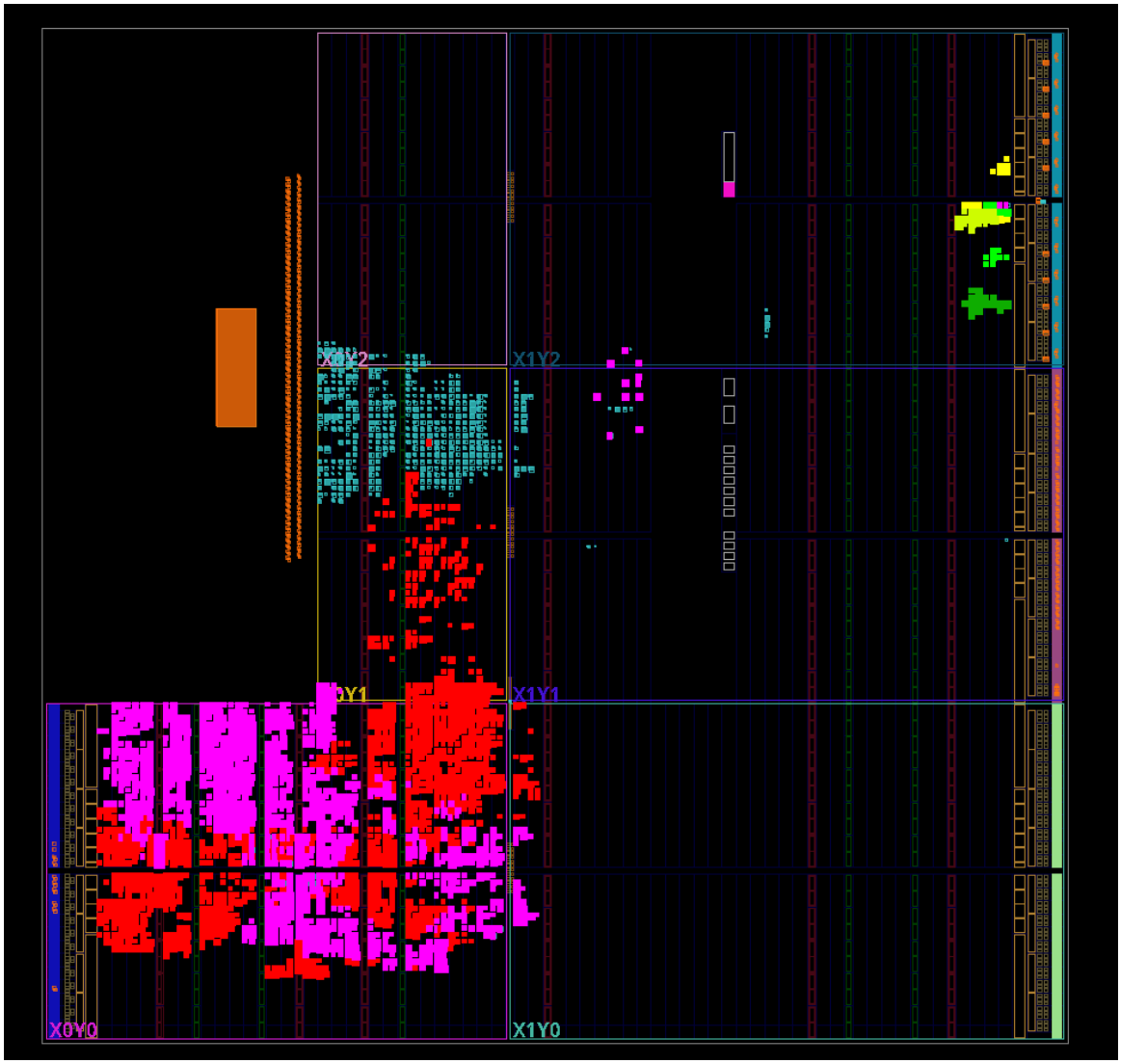

Fig. 8 : vue du design implémenté sur FPGA.

En jaune et vert, les streamers et leurs FIFO ; en rouge, le DMA ; en rose, les interconnections AXI.

2.C.3 noip_ctrl

Le contrôleur de capteurs **noip_ctrl**, permet de contrôler les deux capteurs PYTHON1300 : les allumer, les éteindre, et les configurer en utilisant l'interface SPI. Il dispose d'un port pour bus AXI-Lite et un simple protocole de communication avec le processeur. Le processeur peut envoyer les commandes « allumer », « éteindre », « écrire SPI » ou « lire SPI », puis le numéro du capteur suivi d'autres arguments si nécessaire. Le contrôleur répond simplement avec un code de statut « idle », « OK », « SPI lu » ou « occupé », suivi des données lues si la commande le demandait.

Par SPI, on peut écrire dans les registres de configuration du PYTHON1300, ce qui permet d'adapter le capteur à notre cas d'usage : la *Region of Interest* (ROI) sera la résolution maximale, les quatre canaux LVDS seront utilisés, ainsi que le mode 8-bit et la PLL interne. Le mode d'acquisition d'image est également configuré sur « *triggered* » pour déclencher une prise de vue ponctuelle depuis la carte FPGA.

Dans notre cas d'usage, il a fallu coder une entité « driver SPI », bien qu'il existe bon nombre d'IPs publiques qui remplissent cette fonction : le PYTHON1300 a une implémentation bien spécifique du standard SPI. Non seulement les données sont sur 16 bits tandis que l'adresse est sur 9 bits, mais les données lues sur MISO doivent être échantillonnées sur un front descendant de l'horloge SCK, au lieu du front montant habituel.

Autrement, cette IP réalise l'allumage des capteurs en actionnant à la suite les interrupteurs requis, selon le timing planifié par la *datasheet* du PYTHON1300.

2.D Partie Logicielle

Dans notre application, la partie logicielle consiste à coder des **drivers** pour nos IPs personnalisées et créer un code principal pour le processeur. Cette partie est codée en C.

Le système sera contrôlé par ligne de commande ou avec un code Python depuis un ordinateur, à travers une liaison USB vers l'UART du Zynq. La boucle principale du processeur consisterait donc à recevoir une commande depuis l'UART, et en fonction de l'instruction, envoyer un mot AXI aux contrôleurs, ou diriger le flux d'image venant des streamers vers les sorties Ethernet ou HDMI. A l'allumage du système, un programme d'allumage et de configuration des capteurs est également executé

À ce jour, uniquement les drivers ont été entamés, pour manque de temps et en raison de la difficulté de tester toute la partie logicielle sans avoir accès à la carte. Le fait que la carte ne peut

pas être programmée par USB, mais seulement avec une carte SD bootable rend tout cela encore plus difficile.

3 Conclusion

Même si le projet n'a pas été réalisé dans sa totalité, je suis satisfait du projet et du travail que j'ai effectué. Ce stage m'a apporté beaucoup d'expérience dans le design de système, de connaissances sur la technologie FPGA, et surtout sur le design de PCB. Réaliser un circuit concret et avec des contraintes exigeantes a été très intéressant : et la réalisation d'un système tout entier sur FPGA, qui était une première pour moi, a été très stimulant.

Bien sûr, une partie conséquente de développement et de tests reste à faire : mais je suis convaincu que le travail que j'ai réalisé et documenté constitue une base solide.