Dispositifs électroniques avancés pour la CV-QKD

Damien Fruleux, Eleni Diamanti, Philippe Grangier

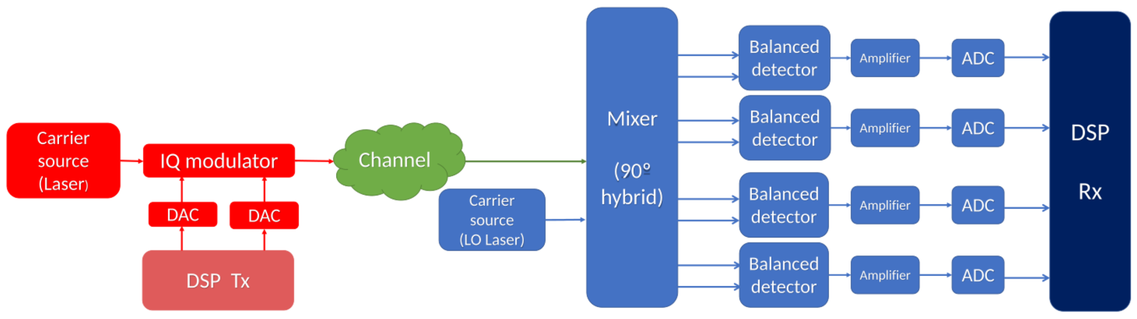
Dispositifs électroniques avancés pour la CV-QKD

Damien Fruleux(1), Eleni Diamanti(1), Philippe Grangier(2)

(1) LIP6, CNRS, Sorbonne Université - (2) IOGS, CNRS, Université Paris-Saclay

Introduction

La **QKD (Quantum Key Distribution)** permet à deux interlocuteurs (Alice et Bob) d'échanger des clés privées : la sécurité de l'échange étant assurée par les lois de la physique quantiques. La **DV-QKD (Discrete Variable)** code les informations sur les propriétés des photons individuels, tandis que la **CV-QKD (Continuous Variable)** code les informations sur les quadratures du champ électromagnétique, ce qui permet d'utiliser des composants de télécommunications optiques classiques cohérentes [1] [2].

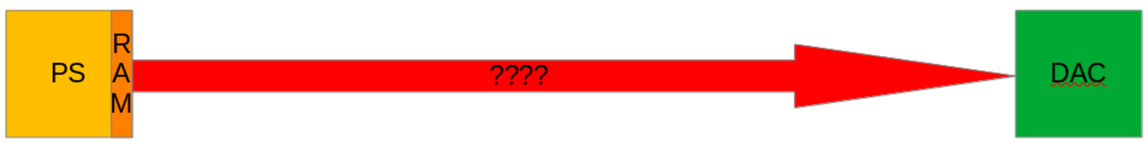


Réalisation d'un **générateur de signaux arbitraires (AWG)** entièrement personnalisable - et par extension d'un **numériseur (digitizer)** - avec la plateforme de développement **Zynq UltraScale+ RFSoC ZCU111** :

- 8 14-bit DACs @ 6.554 GSa/S
- 8 12-bit ADCs @ 4.096 GSa/S
- ARM Cortex-A53 (GNU/Linux) - ARM Cortex-R5 (temps-réel)
- PS : 4 Go RAM SODIMM, extensible jusqu'à 32 Go
- PL : 4 Go SDRAM, non amovible
- 4 SFP28 @ 25 Gbps - 100 Gbps

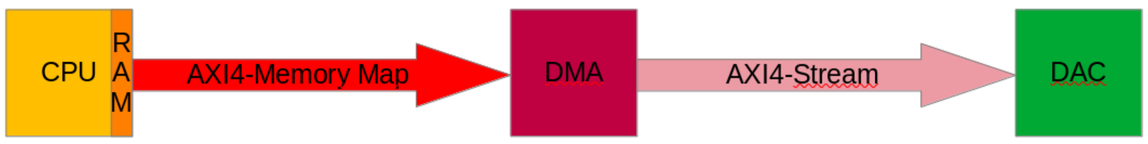
A - Détails de l'architecture

Générer des données numériques, les stocker temporairement dans la RAM (PS ou PL), afin de les convertir en signaux analogiques avec les DACs.



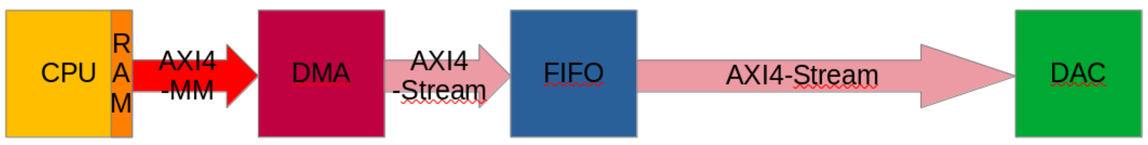
DMA (Direct Memory Acces) : périphérique du CPU qui permet un accès direct entre la RAM (PS ou PL) et un périphérique externe (HDD, SSD...), sans intervention du processeur, si ce n'est pour démarrer et arrêter le transfert.
Chez Xilinx, il s'agit d'un dispositif qui permet des transferts entre une interface **AXI4 (Memory Map) : ici la PSRAM** et une interface **AXI4-Stream : ici les DACs**.

La partie en amont de la DMA exploite le protocole AXI4 (Memory Map), qui fonctionne par **paquets de données**, tandis que la partie en aval exploite le protocole AXI4-Stream, qui fonctionne par **flux de données**.



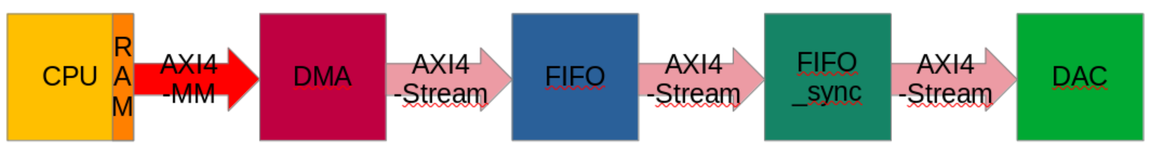
Pour assurer la continuité des données et synchroniser les 2 flux, on utilise une **FIFO (First Input, First Output)** cadencée par 2 horloges distinctes.

Cette mémoire tampon, permet d'**adapter le flux rapide, mais intermittent**, provenant de la DMA **en flux plus lent, mais continu, pour les DACs**.



L'IP développée par Xilinx pour exploiter les DACs utilise le protocole AXI4-Stream comme base, mais **ne prend pas en compte le signal de validation (TVALID)** et échantillonne en permanence les dernières données disponibles.

Réalisation d'une IP : *FIFO_sync* qui impose des données nulles si le signal TVALID n'est plus présent.



Génération d'un signal jusqu'à un débit de **5.20 Go/s (333MHz * 128 bits) = 2.6 GSa/s (128MHz * 256 bits)**. Profondeur mémoire de **32 Go = 16 GSa**.

Interface HPx entre PS et PL limite la fréquence d'échantillonnage. Possibilité d'utiliser la PL-RAM. On peut alors obtenir un débit de **18.75 Go/s (300 MHz * 512 bits) = 9.375 GSa/s (128MHz * 256 bits)**. Profondeur mémoire de **4 Go = 2 GSa**.

B - Utilisation avec Pynq et limites

Utiliser la bibliothèque *Pynq* pour effectuer le transfert :

1. charger le bitstream dans la mémoire (fonction *overlay*)
2. réserver la quantité de mémoire nécessaire (fonction *allocate*, elle-même basée sur la bibliothèque *cma* qui permet une allocation dynamique)
3. remplir la mémoire réservée avec les échantillons à convertir
4. lancer le transfert DMA (fonctions *sendchannel.transfer* et *sendchannel.wait*)

Fonctionne parfaitement avec de petites quantités de données : **1 transfert DMA, soit 64 Mo maximum**, mais pas possible d'effectuer des **transferts continus de l'ensemble de la mémoire**.

Il y a plusieurs raisons :

A - Fonctionnement à très haut niveau des fonctions de la bibliothèque *Pynq* : le temps entre la fin de l'exécution de la 1ère fonction *dma.sendchannel.transfer(input_buffer_a)* et le temps entre le début de la 2ème fonction *dma.sendchannel.transfer(input_buffer_b)* est trop long pour permettre un transfert continu. Besoin d'une **FIFO suffisamment profonde** pour agir comme une mémoire tampon assez longtemps.

B - Pas possible d'allouer plus de 128 Mo de données en utilisant la fonction *allocate* fournie par la bibliothèque *Pynq* : cette mémoire est allouée dans l'espace du Kernel à l'aide du pilote *xlnk*. La mémoire maximale allouable est définie au moment de la compilation du Kernel à l'aide des paramètres de mémoire de la *cma*. Par défaut elle est spécifiée à 128 Mo. Besoin de **recompiler le Kernel** en modifiant certaines options.

C - **PL-RAM non utilisable** avec la DMA dans la version 2.6 de *Pynq*.

C - Utilisation d'un pseudo-driver de périphérique

Utiliser des fonctions bas niveau en C pour **initier des transferts DMA en utilisant directement les registres de contrôle** : plus nécessaire d'allouer de la mémoire et possibilité d'effectuer des transferts continus de toute la mémoire disponible :

1. charger le bitstream dans la mémoire (fonction *overlay*)
2. remplir l'espace mémoire désiré avec les échantillons à convertir
3. lancer le transfert DMA en utilisant directement les registres de configuration accessibles dans l'espace d'adressage.

Fonctions disponibles dans la bibliothèque *POSIX* standard : *open("/dev/mem", ...)* pour **utiliser la mémoire comme un descripteur de fichier** et *int16_t* mem = mmap(...)* pour pouvoir **lire et ecrire avec les fonctions classiques**.

Zone mémoire **sans allocation préalable**, donc on peut **écraser des données** que le Kernel avait sauvegardé à cette adresse mémoire ou bien **perdre les données** que l'on avait placé à cette adresse mémoire car le Kernel a enregistré autre chose : il faut utiliser une **zone mémoire réservée** !

Solution : **utiliser une barrette de 32Go** pour la PS-RAM et laisser les 4 Go de mémoire "allouée" au Kernel (pour le Kernel et l'OS).

Résultat : on dispose de **28 Go de mémoire "non allouée"** par le Kernel, utilisable comme la PL-RAM. C'est un simple espace d'adressage de 28Go.

Conclusion

Elaboration d'un générateur de signaux arbitraires paramétrable et adapté à notre utilisation (meme principe avec les ADCs pour réaliser un numériseur).

Plateforme expérimentale suffisamment flexible et performante pour s'adapter aux contraintes de notre projet de recherche.

Plus de détails sur mon GitHub : https://github.com/DamienFruleux/.

References

[1] Luis Trigo Vidarte. *Design and implementation of high-performance devices for continuous-variable quantum key distribution*. Université Paris Saclay, 2019.

[2] P. Jouguet et al. "Experimental demonstration of long-distance continuous-variable quantum key distribution". In: *Nature Photon* 7 (2013), p. 378.