


| | | |
|---|--|--|
|  | Stage d'exécution <input type="checkbox"/> Stage élève – ingénieur <input type="checkbox"/> Stage année en entreprise : <input checked="" type="checkbox"/> 12 mois <input type="checkbox"/> 1 ^{er} semestre <input type="checkbox"/> 2 ^{ème} semestre Stage projet de fin d'études <input type="checkbox"/> | Date du stage du 08/09/20 au 31/08/21 Année 2020/2021 |
| | | |

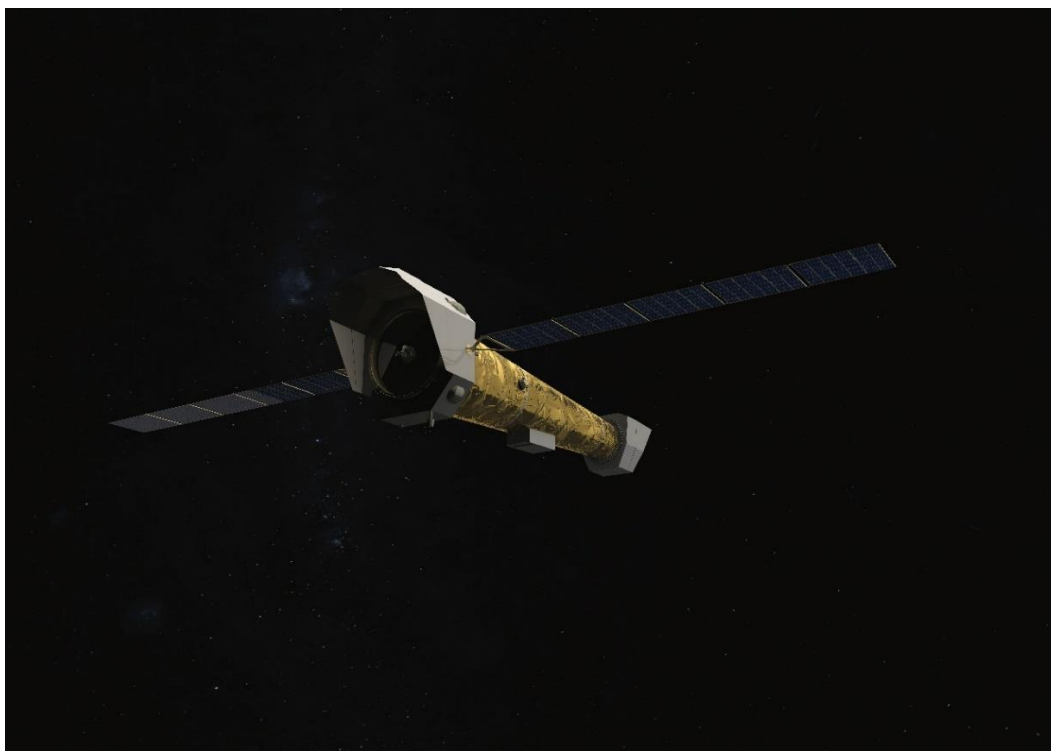
Nom et Prénom du stagiaire : ROLLAND Noémie

Spécialité : ☐ CGP ☒ ETI

Confidentialité du rapport : ☐ OUI ☒ NON

Développement d'un Firmware sur FPGA d'adressage de capteurs pour un projet spatial

Development of a sensor addressing firmware on FPGA for a space project



Nom de l'organisme d'accueil : Institut de Recherche en Astrophysique et en Planétologie

Ville : Toulouse

Pays : France

Nom et Prénom du maître de stage : RAVERA Laurent

Table des matières

| | |
|---|----|
| Acronymes..... | 3 |
| Remerciements | 4 |
| 1. L'IRAP | 5 |
| 2. Le projet Athena X-IFU | 5 |
| 2.1. L'observatoire Athena | 5 |
| 2.2. Le spectromètre X-IFU | 6 |
| 3. Le DRE et le TDM..... | 7 |
| 4. La fonction Row Addressing and Synchronization et le flot de développement | 9 |
| 4.1. Objectif et spécification du RAS | 9 |
| 4.2. Flot de développement | 9 |
| 5. Le Firmware de la fonction RAS | 10 |
| 5.1. La technologie utilisée | 10 |
| 5.2. Développement des fonctions VHDL..... | 10 |
| 5.2.a. Module de réception/stockage et de lecture | 11 |
| 5.2.b. Module de pilotage | 13 |
| 5.2.c. Top module row_addressing..... | 15 |
| 6. Interfaçage avec le PC..... | 15 |
| 6.1. Front Panel Opal Kelly | 15 |
| 6.2. EGSE développé en C++ | 16 |
| 7. Environnement de tests et procédure de vérification | 17 |
| 7.1. Validation par simulation | 18 |
| 7.2. Validation sur cible | 18 |
| 8. Résultats..... | 19 |
| 9. Evolutions de la fonction | 22 |
| 10. Perspectives personnelles | 23 |
| 11. Conclusion..... | 23 |
| 12. Bibliographie..... | 24 |

Acronymes

ADC : Analogue to Digital Converter

Amp SQUID : Amplifier SQUID

Athena : Advanced Telescope for High ENergy Astrophysics

DAC : Digital to Analogue Converter

DRE : Digital Readout Electronics

EGSE : Electrical Ground Support Equipment

EP : Event Processing

FIFO : First In First Out

FPGA : Field Programmable Gate Array

FSM : Finite State Machine

HK : HouseKeepings

IRAP : Institut de Recherche en Astrophysique et en Planétologie

MUX SQUID : MultipleXer SQUID

RAS : Row Address and Synchronization (aussi appelé Row Addressing ou Row Switching)

REV : REVerb (chevauchement)

SPI : Serial Peripheral Interface

SQUID : Superconducting Quantum Interference Device

TDM : Time Domain Multiplexing

TES : Transition Edge Sensor

TC : TeleCommands

TM : TeleMetry

X-IFU : X-Ray Integral Field Unit

Remerciements

Je tenais à remercier dans un premier temps mon tuteur de stage Laurent Ravera qui m'aura accueillie et accompagnée tout au long de ce stage. J'ai pu, grâce à son accompagnement et à la mission proposée, apprendre beaucoup et aiguillier encore un peu plus mon choix de carrière.

J'aimerais également remercier les membres de l'IRAP avec lesquels j'ai pu échanger tant à propos du travail qu'à propos de sujets plus légers. Tous ces échanges ont contribué à mon intégration et au bon déroulé de mon année au laboratoire. J'ai une pensée particulière pour les membres de l'équipe DRE du projet Athena X-IFU, équipe dans laquelle j'ai été intégrée, qui ont su me mettre à l'aise très rapidement, me soutenir tout au long de l'année et avec qui j'ai pu passer de très bons moments conviviaux.

Enfin je voudrais remercier Mr Didier Barret ainsi que Mme Estelle Barret grâce auxquels j'ai pu trouver ce stage de césure malgré la période difficile dans laquelle nous nous plaçons.

1. L'IRAP

L'IRAP (Institut de Recherche en Astrophysique et en Planétologie) est né le 1^{er} janvier 2011 suite à la fusion de tout ou partie de 4 laboratoires de recherche. Environ 300 personnes travaillent à l'IRAP : 110 chercheurs et enseignants-chercheurs, 80 permanents d'appui et de soutien à la recherche, 50 doctorants et 50 post-docs ou CDD, à cela s'ajoute nombre d'étudiants en stage.

Les objectifs scientifiques de l'IRAP portent sur l'étude et la compréhension de l'Univers et de son contenu. Pour cela l'ensemble des projets menés sont répartis dans 6 groupes thématiques :

- DIP : Dynamique des Intérieurs Planétaires,
- PS2E : Physique du Soleil, des Etoiles et des Exoplanètes,
- GAHEC : Galaxies, Astrophysique des Hautes Energies et Cosmologie,
- PEPS : Planètes, Environnements et Plasmas Spatiaux,
- MICMAC : Milieu Interstellaire, Cycle de la Matière, Astro-Chimie,
- SISU : Signal Image en Sciences de l'Univers.

C'est dans l'équipe thématique de GAHEC que se place le projet Athena X-IFU sur lequel porte ce stage.

L'IRAP compte de nombreuses collaborations aussi bien à l'échelle régionale qu'à l'échelle internationale avec des organismes comme le CNES, l'ESA, l'ESO, la NASA, la JAXA...

2. Le projet Athena X-IFU

2.1. L'observatoire Athena

Athena (Advanced Telescope for High ENergy Astrophysics) est une mission observatoire à rayon X sélectionnée par l'ESA dans le cadre de son programme Cosmic Vision afin d'observer l'univers chaud et énergétique. Athena est de classe L (large) au sein de ce programme et devrait décoller en 2034.

L'objectif de cet observatoire est de recueillir des données sur le rayonnement X et ainsi comprendre comment les baryons chauds s'assemblent pour former des groupes et clusters de galaxies ainsi que d'en apprendre plus sur la physique d'accrétion en éléments compacts. Ces observations permettront d'étudier la croissance des trous noirs super-massifs ainsi que certains processus associés. L'observatoire Athena sera placé sur une orbite héliocentrique autour du point de Lagrange L2 qui se trouve à 1,5 millions de km de la Terre dans la direction antisolaire en dehors des ceintures de Van Allen.

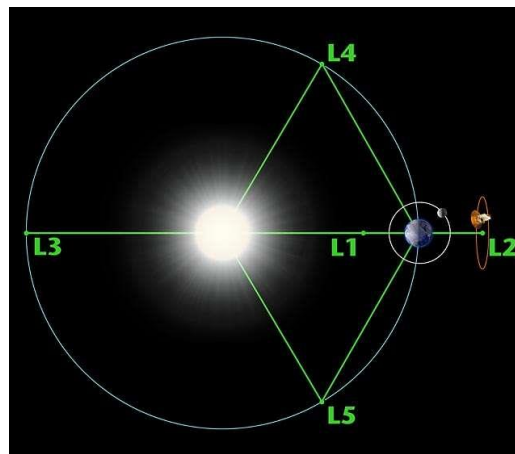


Figure 5 : Point de Lagrange L2

Athena sera un télescope à rayon X de longueur de focale de 12m, les photons seront dirigés par un miroir vers les deux instruments qui seront à son bord, le WFI (Wide Field Imager) un spectro-imageur large champ et X-IFU (X-Ray Integral Field Unit) qui feront leurs observations en alternance.

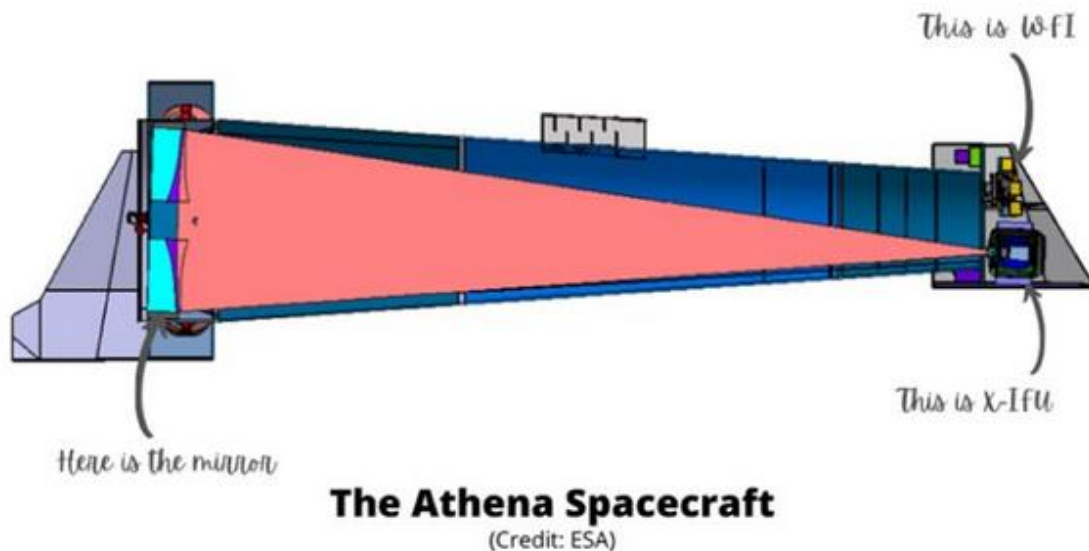


Figure 6 : L'observatoire Athena

2.2. Le spectromètre X-IFU

L'instrument X-IFU est un spectromètre cryogénique qui observera des rayons-X avec une résolution spectrale très haute de 2,5 eV pour un rayonnement d'énergie de 7keV et une haute qualité d'imagerie. Ceci permettra de capturer des images d'objets astrophysiques dans lesquelles chaque pixel fournira un grand nombre d'information. Ces observations devront se faire à une température de 55 mK puisque c'est à cette température que les capteurs fonctionnent de façon optimale.

Le développement de cet instrument est géré par un consortium international (France, Etats-Unis, Japon, Pays-Bas, Espagne...). Les activités techniques du consortium sont pilotées par le CNES et les activités scientifiques sont pilotées par l'IRAP. Chaque pays et structure participe à une partie du projet. L'IRAP, laboratoire au sein duquel se déroule ce stage, est impliqué dans l'électronique numérique de lecture (DRE).

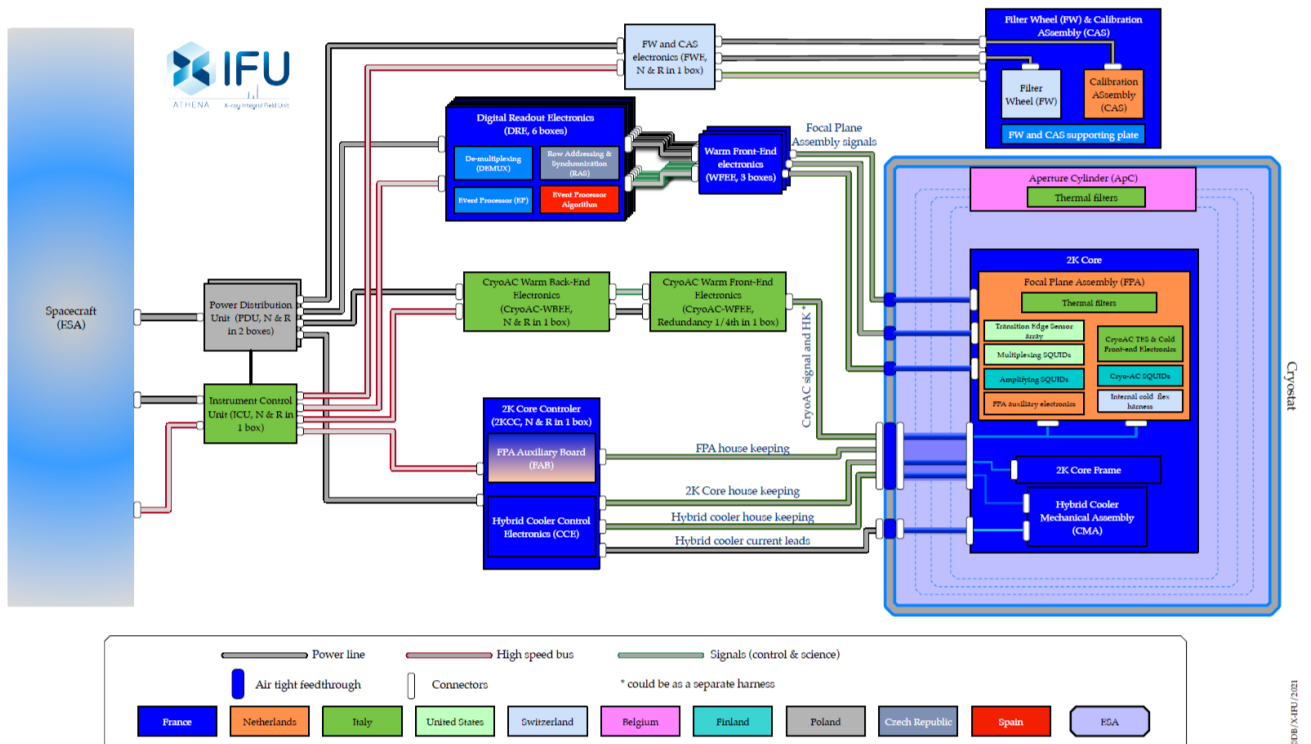


Figure 7 : Diagramme fonctionnel de l'instrument X-IFU

3. Le DRE et le TDM

Le DRE (Digital Readout Electronics) contrôle les 3168 cellules de détection de la matrice de détecteurs TES (Transition Edge Sensors) qui se trouvent dans la partie froide de l'instrument. Le DRE comporte 4 modules DEMUX qui ont pour rôle de récupérer les signaux en sortie de l'électronique des détecteurs, un module EP (Event Processing) qui permet l'interfaçage entre l'extérieur et l'intérieur du DRE et un module RAS qui pilote la lecture des détecteurs, il sera décrit par la suite (cf figure 8).

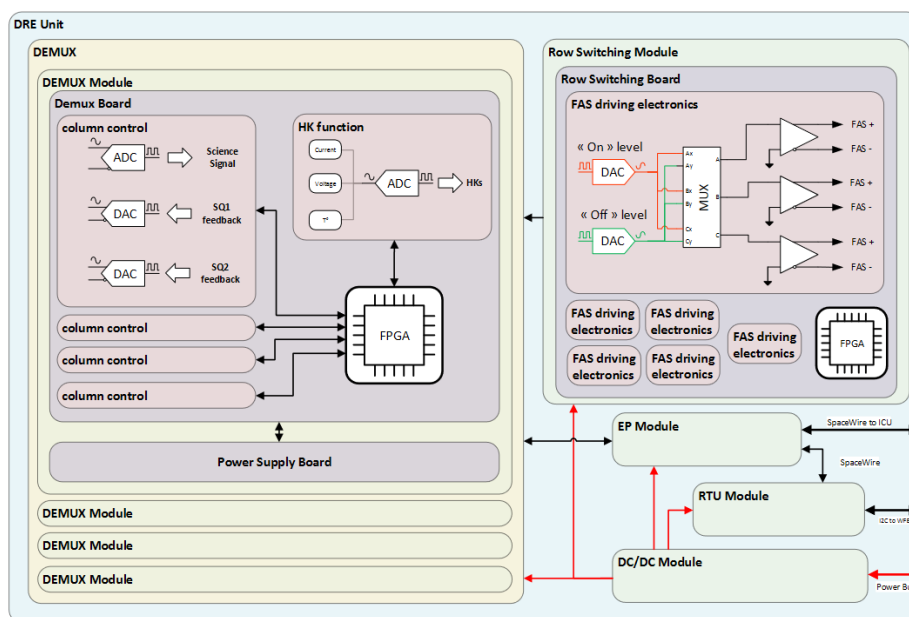


Figure 8 : Schéma fonctionnel du DRE

Le DRE effectue la lecture de 96 chaînes de mesure constituées chacune de 33 cellules TES. Les microcalorimètres TES sont des capteurs de particules cryogéniques. Lorsqu'un photon est absorbé, le rayonnement est transformé en chaleur ce qui fait varier la résistance thermo-dépendante du TES. Pour pouvoir mesurer de façon significative le rayonnement du photon, les détecteurs sont plongés dans un bain cryogénique à une température de 55 mK (cf figure 9).

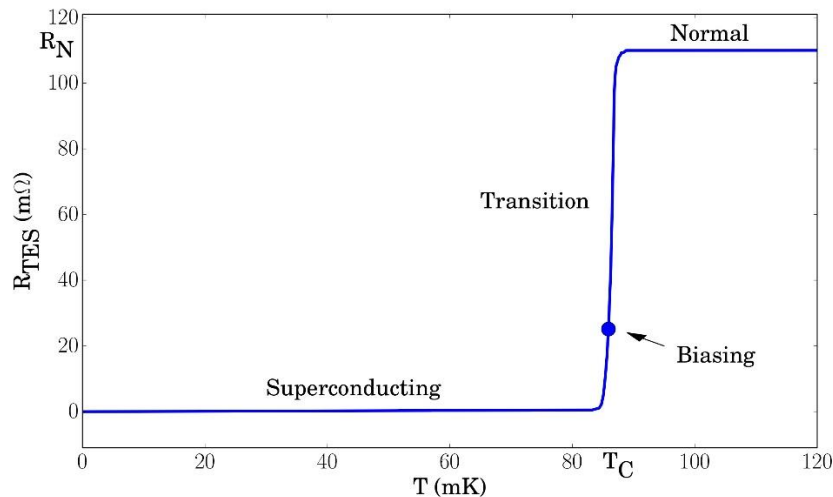


Figure 9 : Principe et caractéristique d'un TES

Afin d'obtenir un fonctionnement optimal du TES, la température ainsi que la tension de polarisation doivent le maintenir dans sa zone de transition. De cette façon, la moindre variation de température entraîne une variation significative de sa résistance.

Cette contrainte de température impose une technique de lecture présentant le moins de perte de chaleur possible, ainsi une lecture individuelle de chaque pixel est impossible à cause de la quantité de fils que cela représenterait (minimum 2 fils pour chacun des 3168 détecteurs). La technique choisie est donc celle du multiplexage en temps (TDM), elle consiste à lire les 33 cellules d'une colonne de lecture séquentiellement en les adressant grâce à leur ligne et à leur cluster. Pour cela, on multiplexe les détecteurs en pilotant des interrupteurs placés en début de ligne et en début de cluster, ce qui réduit considérablement le nombre de fils utilisés. Les détecteurs d'une colonne sont organisés en 4 clusters composés chacun de 9 lignes (cf figure 10), cela permet de lire les 33 pixels séquentiellement en utilisant uniquement 13 interrupteurs (24 fils). C'est dans ce contexte que se place le développement du prototype de la fonction Row Addressing and Synchronization qui permettra de démontrer la faisabilité et de valider l'architecture imaginée.

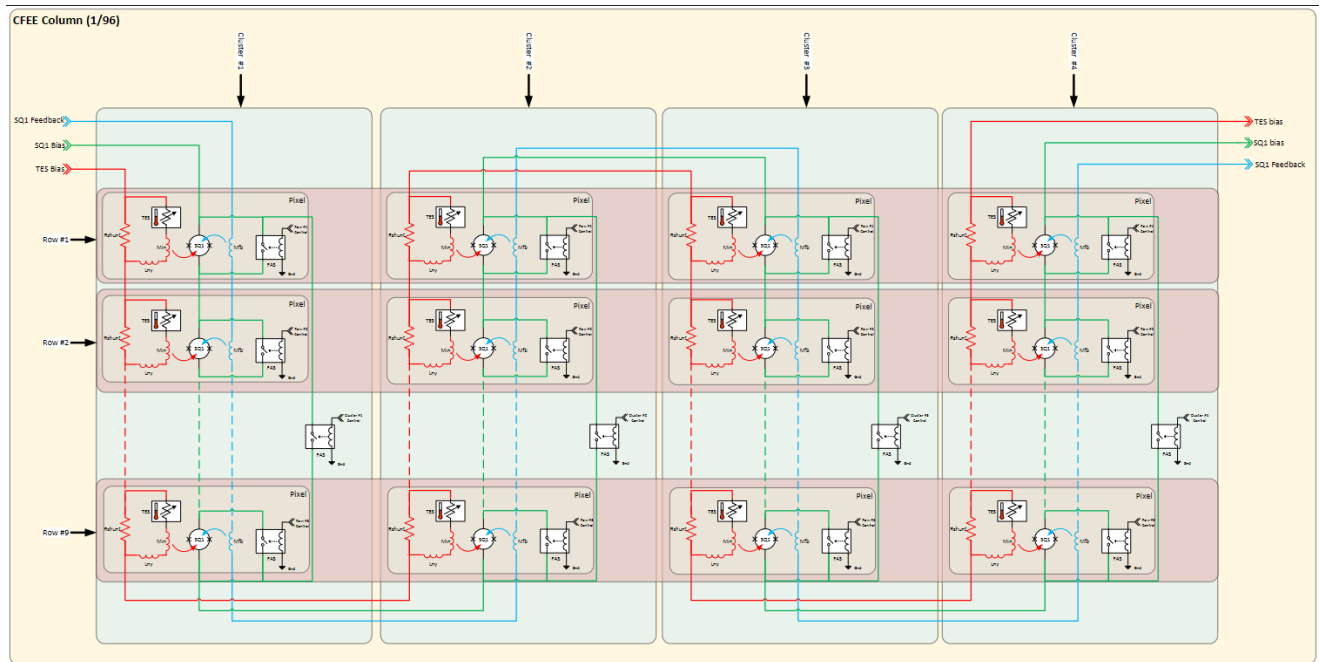


Figure 10 : Lecture TDM par ligne et cluster

Dans ce schéma, on peut observer une colonne de lecture composée de 33 pixels. L'architecture imaginée correspondant à une répartition en 9 lignes et 4 clusters. Par exemple, en activant la ligne 0 et le cluster 0, on adresse le pixel 0.

4. La fonction Row Addressing and Synchronization et le flot de développement

4.1. Objectif et spécification du RAS

L'objectif du développement du modèle de démonstration de la fonction RAS est de valider l'architecture imaginée et d'adopter la technique de lecture TDM pour les détecteurs. Cette fonction se place dans le module DRE, elle reçoit des commandes et des signaux depuis l'EP et génère des signaux pour piloter les interrupteurs ainsi qu'un signal de synchronisation pour les modules DEMUX.

La fonction doit pouvoir piloter 13 interrupteurs : 9 interrupteurs pour les lignes et 4 pour les clusters d'une colonne de lecture. Chaque interrupteur est commandé par une séquence envoyée par l'utilisateur, cette séquence s'exprime sur 40 bits. En plus des commandes envoyées, des paramètres sont également saisis par l'utilisateur, ces paramètres concernent la fréquence de fermeture des interrupteurs, le temps de chevauchement entre deux interrupteurs, la longueur de la séquence prise en compte... C'est à partir de l'ensemble de ces commandes que la fonction traite les séquences et les manipule pour générer les signaux de pilotage des interrupteurs. La fonction RAS a aussi pour objectif de générer un signal de synchronisation en fonction des séquences reçues, ce signal est ensuite envoyé aux modules DEMUX.

4.2. Flot de développement

Le développement de la fonction RAS a été fait en équipe : un ingénieur de l'IRAP a pris en charge la conception de la carte électronique, un autre ingénieur a développé des moyens de test logiciels et on m'a

confié la responsabilité du développement du Firmware du FPGA. Mon travail s'est ainsi articulé autour de trois activités principales : le développement de modules VHDL pour le FPGA ainsi que de son environnement de test en Python, la validation des modules par simulation et les tests directement sur FPGA.

A cela s'ajoute une certaine rigueur sur la gestion en configuration puisque chaque version est tracée à l'aide de Git et doit répondre à des exigences. J'ai pu participer à l'amélioration de ce système de gestion en trouvant la méthode la plus adaptée pour un projet sur FPGA. Ainsi chaque modification et action est suivie afin de pouvoir retrouver à tout moment une version antérieure et reproduire les tests à l'identique.

5. Le Firmware de la fonction RAS

5.1. La technologie utilisée

La fonction RAS est pilotée par un FPGA. Pour ce modèle de démonstration il a été choisi d'utiliser le module d'intégration FPGA USB3.0 compact XEM7310-A75T Opal Kelly doté d'un FPGA Xilinx Artix-7. L'ensemble du développement VHDL est fait sur Vivado 2020.1. Le choix de cette technologie s'accompagne d'un kit de développement logiciel : le Front Panel Opal Kelly (cf figure 11).

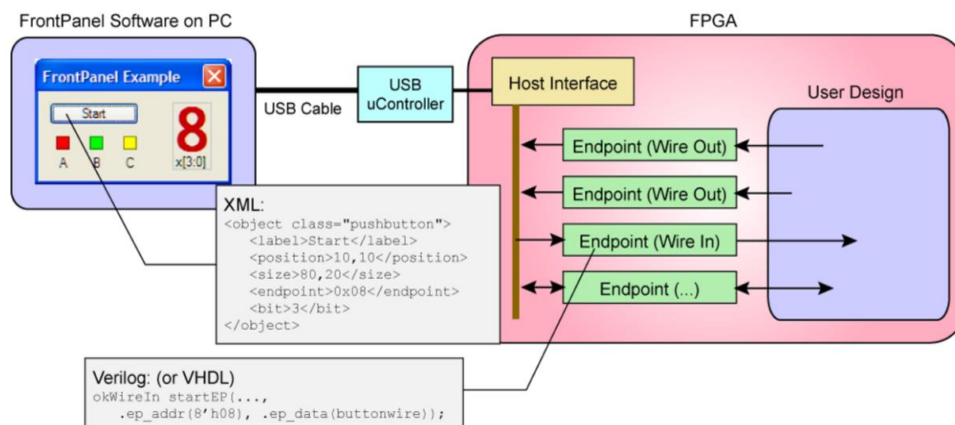


Figure 11 : Principe du Front Panel Opal Kelly

Ce dernier permet de faciliter la configuration, la communication et l'interfaçage entre le PC et le FPGA. Il présente plusieurs outils :

- Les composants Front Panel HDL implantés dans le FPGA,
- Le Firmware dans lequel on implémente les composants HDL du Front Panel,
- L'API (Application Programming Interface) qui permet la programmation d'une interface graphique avec des composants virtuels pour communiquer directement avec les composants HDL.

5.2. Développement des fonctions VHDL

Le Firmware RAS a donc pour objectif de gérer la réception de commandes envoyées en entrée du FPGA et de les stocker afin de les utiliser et les manipuler pour générer les signaux qui pilotent l'activation des interrupteurs. Pour cela le Firmware se découpe en 2 fonctions principales que l'on distingue grâce à la valeur du paramètre mode :

- La gestion des commandes, l'enregistrement et la re-lecture des paramètres et séquences lorsque mode = '0',

- Le pilotage des interrupteurs en fonction des séquences et des paramètres stockés dans le FPGA et la synchronisation avec la fonction DEMUX lorsque mode = '1'.

On trouve également dans le Firmware des interfaces pour communiquer avec le monde extérieur et le PC ainsi qu'un bloc de pilotage de DACs. Ces DACs ont pour rôle d'envoyer le bon niveau de tension aux interrupteurs en fonction des signaux de sortie du FPGA.

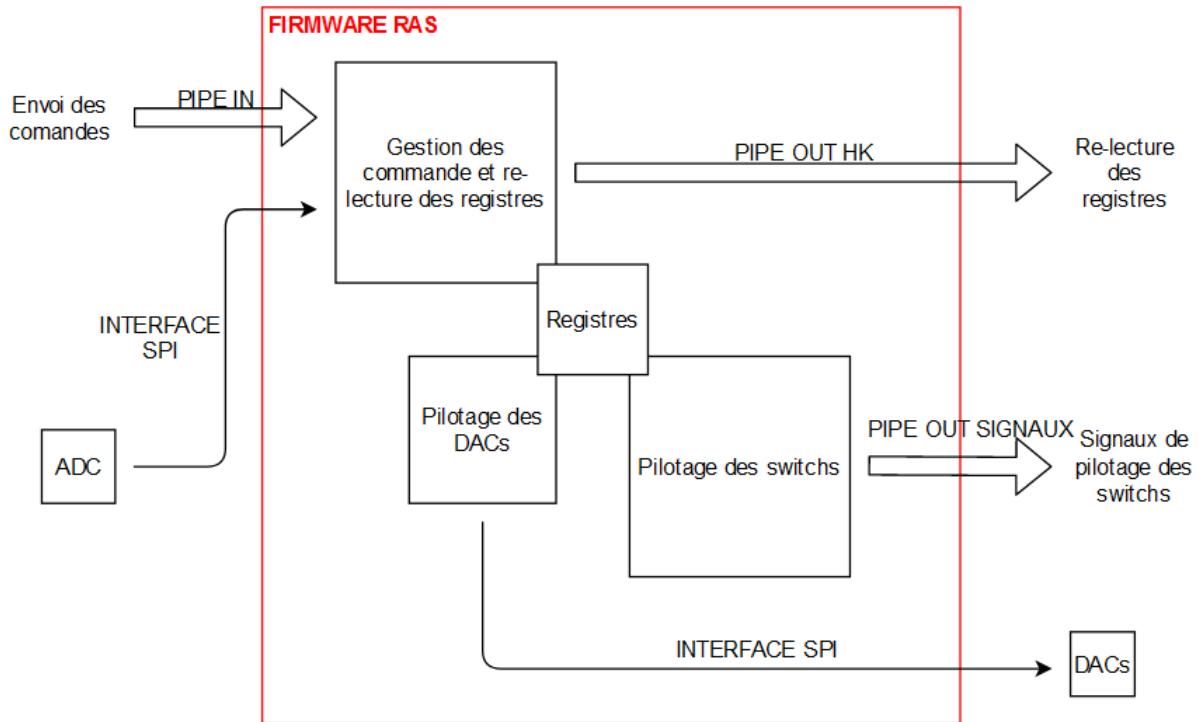


Figure 12 : Schéma du Firmware RAS et de ses interfaces

L'horloge interne du FPGA à 200 MHz est divisée pour obtenir les fréquences désirées (100 MHz et 5 MHz), ces fréquences sont ensuite utilisées pour la lecture des registres et le pilotage de rows. Les fonctions de division de fréquence ne sont pas présentées dans ce rendu par souci de synthèse. Afin de respecter les règles de codage CNES seule une fréquence (100 MHz) est utilisée comme horloge, le signal de fréquence 5MHz est un clock enable.

5.2.a. Module de réception/stockage et de lecture

Ce module permet de réaliser deux fonctions différentes : il permet de recevoir des commandes de l'extérieur et de les stocker dans des registres mais il permet aussi de pouvoir lire le contenu des registres (HouseKeeping HK). Les registres contiennent chacun une commande sur 32 bits où chaque bit correspond à un paramètre ou à un morceau de la séquence, un dictionnaire de commandes a été établi pour lister l'ensemble de ces commandes, à noter que les séquences des interrupteurs exprimés sur 40 bits seront saisies dans deux registres différents (RowX LSB et MSB).

[illegible]

Figure 13 : Dictionnaire de commandes

Chaque registre est accessible grâce à une adresse sur 8 bits. En plus de ces bits d'adresse, on dispose d'un bit pour indiquer l'action que l'on souhaite réaliser. Si on souhaite lire le contenu d'un registre il suffit de mettre ce bit d'action à '0'. La lecture des registres peut se faire à n'importe quel moment tandis que l'écriture dans les registres n'est possible que dans le cas où mode='0' (sauf pour le changement de valeur de mode). Si on souhaite écrire dans un registre, il suffit de donner la valeur '1' au bit d'action, ensuite on peut envoyer la valeur de sa commande. La réception des commandes se fait en deux temps : réception de l'adresse en premier puis écriture des données dans le registre stockage pointé par l'adresse reçue. Ces registres sont répertoriés dans un package VHDL.

Les commandes sont envoyées depuis un PC par l'intermédiaire du Front Panel de Opal Kelly, on peut ainsi envoyer directement des mots dans le FPGA depuis un fichier grâce à un Pipe In. Le Pipe In est un « tunnel » qui permet de transporter des mots de 32 bits, son fonctionnement sera précisé dans la partie 6. Pour faire interface entre le Pipe In et le module de réception et de stockage (ce sera également le cas pour interfacer le module de pilotage et les Pipe Out), on utilise une FIFO qui permet de gérer le changement de domaine d'horloge (Front Panel : 100,8 MHz, module de réception : 100 MHz) et stocker les données qui viennent du Pipe, on vient donc lire dans cette FIFO et non dans le Pipe. Ces FIFO nous permettent également de récupérer des indicateurs (FIFO vide, FIFO pleine, données valides...) qui facilitent la réception des données. Les FIFO utilisées sont des Independants Clock Block RAM FIFO implémentées depuis l'IP Generator de Xilinx sur Vivado. Voici la machine d'état (FSM) simplifiée décrivant la réception, le stockage de commande et la lecture de registre :

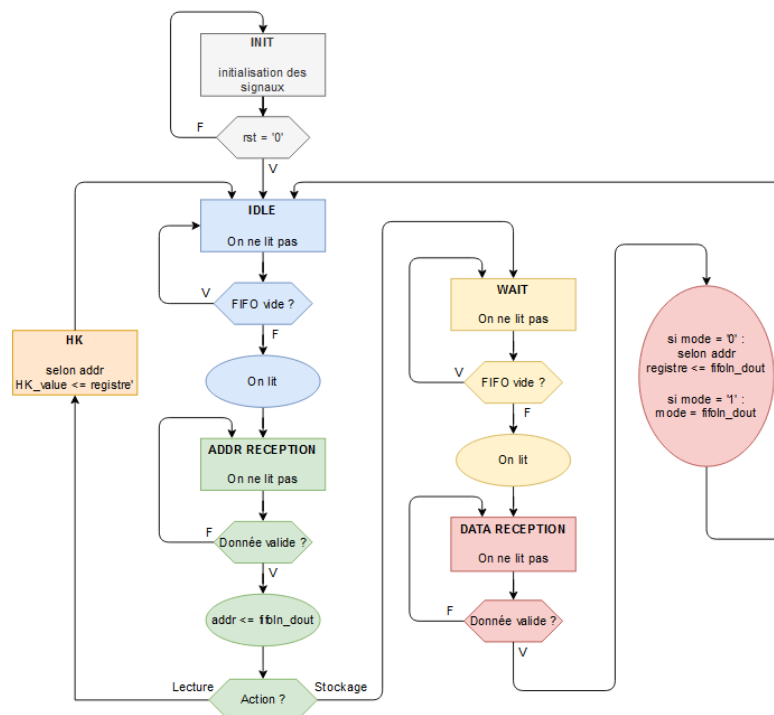


Figure 1 : Machine d'état simplifiée décrivant la réception, le stockage et la lecture de commandes

5.2.b. Module de pilotage

L'objectif de ce module est de récupérer les séquences stockées dans les registres et de les manipuler en appliquant les différents paramètres stockés dans les autres registres. En sortie on obtient ainsi les signaux de pilotage des interrupteurs. Dans un souci de précision on veut pouvoir ajuster les timings des signaux d'adressage. On souhaite donc pouvoir ajouter un chevauchement positif ou négatif entre deux interrupteurs fermés avec une résolution de réglage de 10 ns. Chaque interrupteur est ainsi fermé pendant $T_{row} +/ - REV$. Avec T_{row} la période de fermeture d'un interrupteur et REV le temps de

chevauchement, $-70\text{ ns} \leq REV \leq +70\text{ ns}$, ces 2 paramètres sont saisis par commande. Voici l'architecture du module de pilotage d'un seul interrupteur, ce module est implémenté pour chaque interrupteur piloté :

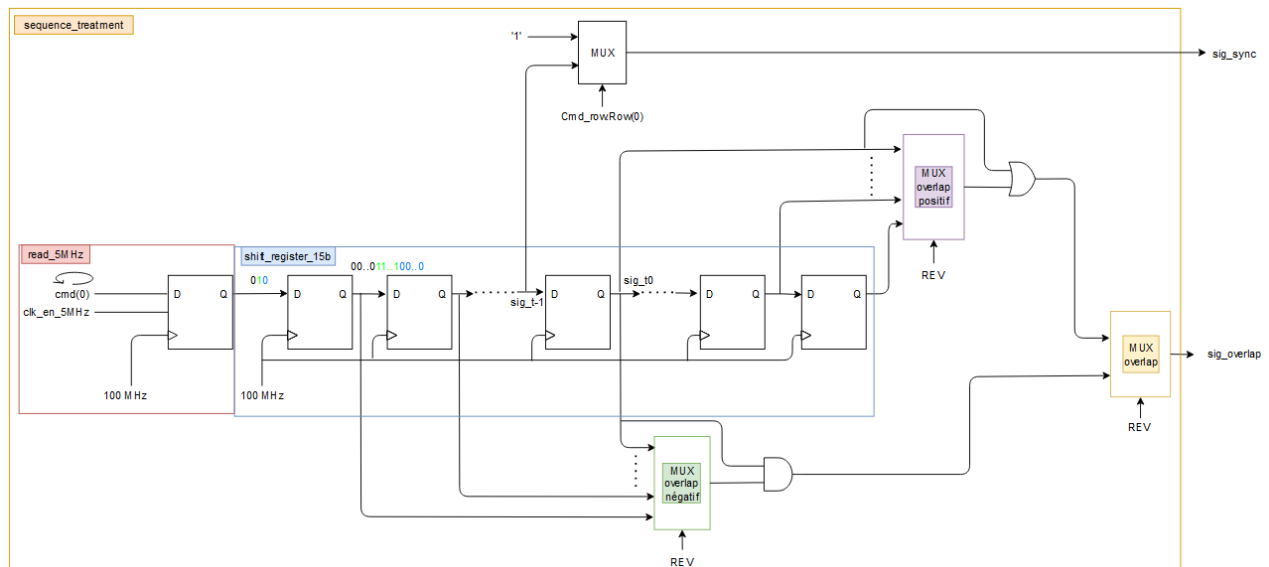


Figure 2 : Module de pilotage des switches

Dans un premier temps on lit chaque bit de la séquence stockée toutes les 200 ns (5 MHz), pour cela on utilise une bascule pilotée par une horloge à 100 MHz et un signal `clk_en` (clock enable) à 5MHz qui permettra d'activer la sortie de la bascule uniquement toutes les 200 ns pendant 10 ns (le signal de `clk_en` possède une fréquence f_{row} variable que l'on peut régler par commande, pour simplifier les choses dans ce document on gardera une fréquence de 5MHz).

On vient ensuite lire la sortie de la première bascule à une fréquence de 100 MHz (toutes les 10 ns) ce qui nous permet d'obtenir une version de la séquence dans laquelle chaque bit est séparé dans le temps de 10 ns (contre 200 ns précédemment) : $\frac{200\text{ ns}}{10\text{ ns}} = 20$. Chaque bit de la séquence apparaît donc maintenant 20 fois. Le but est ensuite de créer du retard sur le signal (dans notre cas on retarde le signal de 14 coups d'horloge). Le signal en sortie de la 7^{ème} bascule (au milieu) correspond à notre signal de référence (c'est-à-dire ni en avance, ni en retard).

Suivant la valeur de REV (paramètre saisi par commande) on va vouloir ajouter un chevauchement positif ou négatif au signal, pour cela il suffit de récupérer le signal retardé (dans le positif ou le négatif) correspondant à la valeur de REV et de le combiner au signal modèle. Pour ce choix on utilise des multiplexeurs. On dispose de 3 multiplexeurs :

- 1 multiplexeur pour les chevauchements négatifs qui va venir piocher dans les signaux en avance (entrée de la 1^{ère} bascule à la sortie 7^{ème} bascule) par rapport au signal de référence,
- 1 multiplexeur pour les chevauchements positifs qui va venir piocher dans les signaux en retard (sorties de la 8^{ème} bascule à la 14^{ème} bascule) par rapport au signal de référence,
- 1 multiplexeur qui va choisir le chevauchement positif ou le chevauchement négatif en fonction du signe de REV (si $REV(3)=0$ on choisit le chevauchement positif, si $REV(3)=1$ on choisit le chevauchement négatif) .

Avant d'arriver au dernier multiplexeur qui détermine le signe du chevauchement, un ET logique est appliqué entre le signal modèle et le signal en avance choisi (on obtient ainsi un signal plus court), et un OU logique est appliqué entre le signal modèle et le signal en retard choisi (on obtient ainsi un signal plus long).

Ce sont les résultats de ces opérations qui se trouvent en entrée du dernier multiplexeur, on peut observer un exemple dans la figure 14.

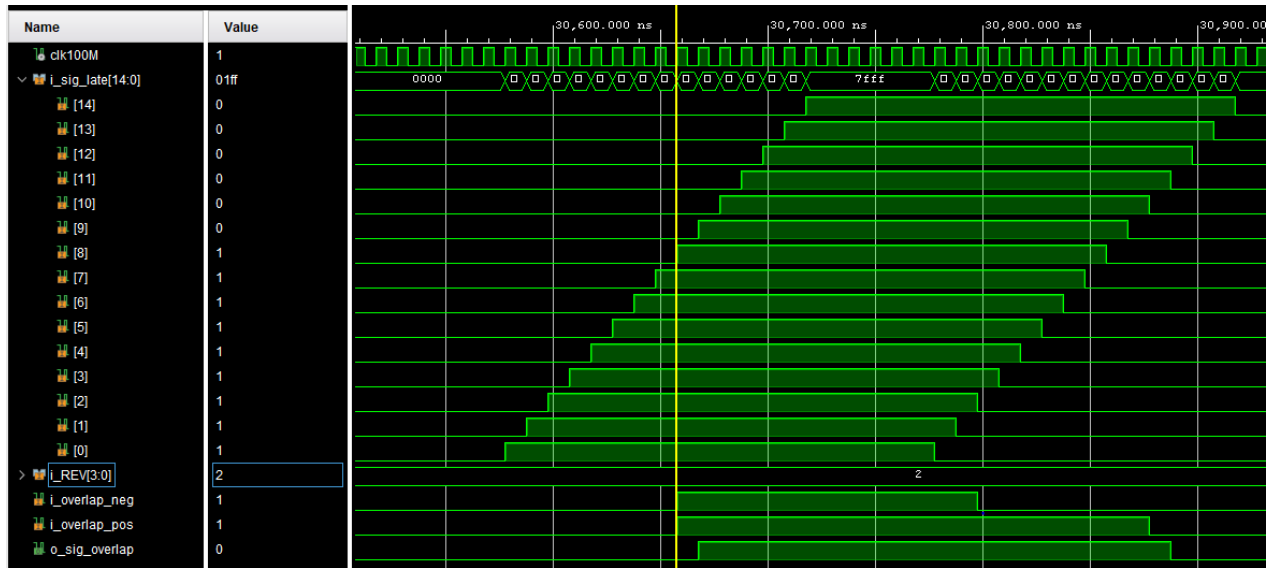


Figure 14 : Chronogramme des signaux retardés

Dans ce chronogramme on observe les signaux retardés, le signal de référence ici est sig_late(7). On voit que REV = + 2 donc le signal avec chevauchement positif est le résultat d'un OU logique entre le signal de référence sig_late(7) et le signal retardé de 2 coups d'horloge sig_late(9). On remarque que le signal overlap_pos est un coup d'horloge en retard dû à l'horloge du process du multiplexeur.

Une autre fonction remplie par le module de pilotage des interrupteurs est d'envoyer un signal de synchronisation au module DEMUX (afin de synchroniser l'activation d'un pixel et la lecture de ce pixel). Ce signal sera activé (= '1') lors d'un début de séquence pendant une période T_{row} renseignée par l'utilisateur par commande externe. Pour construire ce signal, il faut d'abord trouver quel(s) interrupteur(s) va être fermé en premier. Pour cela on utilise les registres dans lesquels sont stockées les séquences des interrupteurs. Si le premier bit de la séquence est égal à '1' alors le signal de l'interrupteur en question sera utilisé dans la construction du signal de synchronisation, si le premier bit est égal à '0' alors un signal constant égal à '1' sera utilisé. Un ET logique est ensuite appliqué entre tous les signaux sélectionnés et on obtient le signal de synchronisation global.

5.2.c. Top module row_addressing

Le module row_addressing a pour objectif de relier la fonction de réception et stockage des commandes et le module de pilotage des interrupteurs, il permet également d'implémenter les composants Opal Kelly pour utiliser le Front Panel et ainsi recevoir et faire sortir des données de la carte. De la même façon que pour la réception, le module présente un process qui gère l'envoi des signaux de sortie dans le composant PipeOut du Front Panel toutes les 10 ns à la sortie de la carte. Un module qui pilote des DACs externes est également présent dans ce top module, il est actuellement encore en cours de développement.

6. Interfaçage avec le PC

6.1. Front Panel Opal Kelly

Le pilotage des rows se fait à partir de commandes envoyées par l'utilisateur, il faut donc prévoir d'établir la communication entre celui-ci et le FPGA. Pour cela, on utilise le Front Panel Opal Kelly depuis un PC. Pour cela on utilise des Pipe In et Out, ce sont des tunnels qui nous permettent d'envoyer des mots de 32 bits depuis un fichier vers le FPGA ou inversement. Ils sont implémentés dans le Firmware puisque ce sont des composants HDL implantés sur la carte Opal Kelly. Pour manipuler ces composants on établit une interface graphique facilement utilisable avec laquelle on pourra échanger des données avec le FPGA. Pour connecter cette interface et les composants HDL, on utilise des endpoints, ce sont des terminaux avec des plages d'adresse déterminées en fonction des composants utilisés, les composants HDL sont ainsi connectés aux mêmes endpoints que les composants virtuels qui permettent leur manipulation.

L'application Front Panel est donc utilisée dans un premier temps, elle permet de configurer la carte Opal Kelly et communiquer. Pour communiquer, il faut développer un fichier XML dans lequel on place des composants virtuels connectés aux mêmes endpoints que les composants HDL implantés sur la carte. Des composants okFilePipe sont utilisés pour les transferts de données depuis ou vers un fichier binaire pour se connecter aux Pipe In et Pipe Out utilisés, mais aussi un okTriggerButton, qui, quand on clique dessus, déclenche un Trigger In sur la carte et enfin un okTriggerLog qui nous permet d'afficher les Trigger Out de la carte.

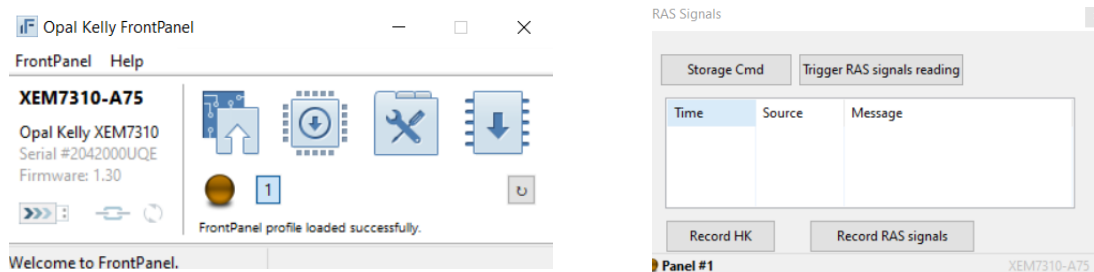


Figure 3 : API et interface XML du Front Panel

Afin d'envoyer les données vers le FPGA dans le bon format mais également de récupérer les signaux en sortie et de pouvoir les lire, on utilise des scripts Python. J'ai développé ces scripts pour permettre à l'utilisateur de n'avoir à saisir que les commandes qu'il souhaite envoyer au FPGA dans un fichier texte. Un premier script va donc ajouter les adresses correspondantes à chaque registre adressé avant la bonne commande. Un second script se chargera de convertir les données présentes dans le fichier texte au format attendu en entrée du Pipe In. Enfin pour pouvoir lire le fichier binaire que l'on récupère en sortie du Pipe Out avec la valeur des signaux de sortie toutes les 10 ns, on utilise un autre script qui lit le fichier binaire et qui convertit les données dans un fichier texte que l'on peut exploiter.

Une version du Front Panel est également disponible pour la simulation. Pour cela, il faudra ajouter les process simulant les composants HDL dans le fichier Test Bench VHDL. Il faudra également implémenter la version de simulation de ces composants (on aura donc la version des composants HDL pour la synthèse et l'implémentation et la version pour la simulation). Cette simulation est tout de même limitée puisque qu'elle ne permet de visualiser que la simulation comportementale et non les simulations Post-Place et Post-Route.

6.2. EGSE développé en C++

Afin de faciliter l'interfaçage entre le PC et le FPGA et l'envoi de commande, j'ai pu collaborer avec un membre de l'équipe pour créer un EGSE développé en C++. Il permet la saisie de commande de façon bien plus graphique et manipule les composants Front Panel de la même façon qu'expliqué précédemment.

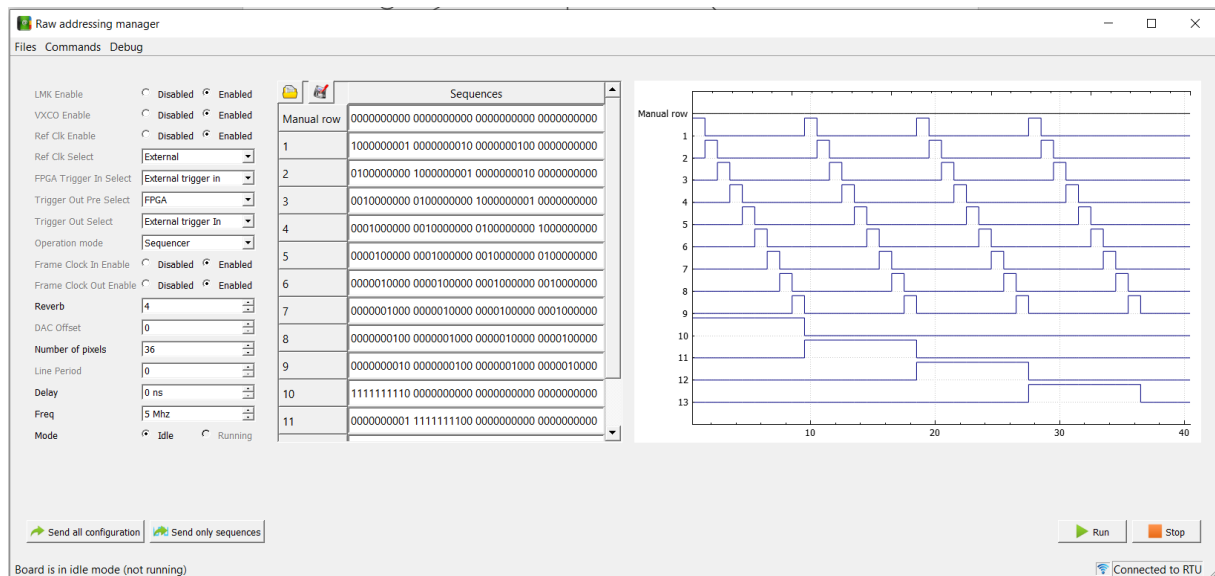


Figure 4 : Logiciel EGSE X-IFU Studio

7. Environnement de tests et procédure de vérification

Une autre partie non négligeable dans le développement d'un Firmware est de prévoir, en parallèle, des procédures pour le tester et le valider. Il y a deux cas de figure, les tests en simulation et les tests sur cible FPGA. Le but de la procédure de test est de pouvoir la répéter plusieurs fois à l'identique au fil du développement du Firmware (tests de non régression), pour cela il faut donc minimiser l'intervention de l'utilisateur et automatiser le plus possible la procédure. Seul le fichier texte avec les paramètres et les séquences de commande sera donc saisi manuellement. Le schéma de l'environnement de test est à voir en figure 15.

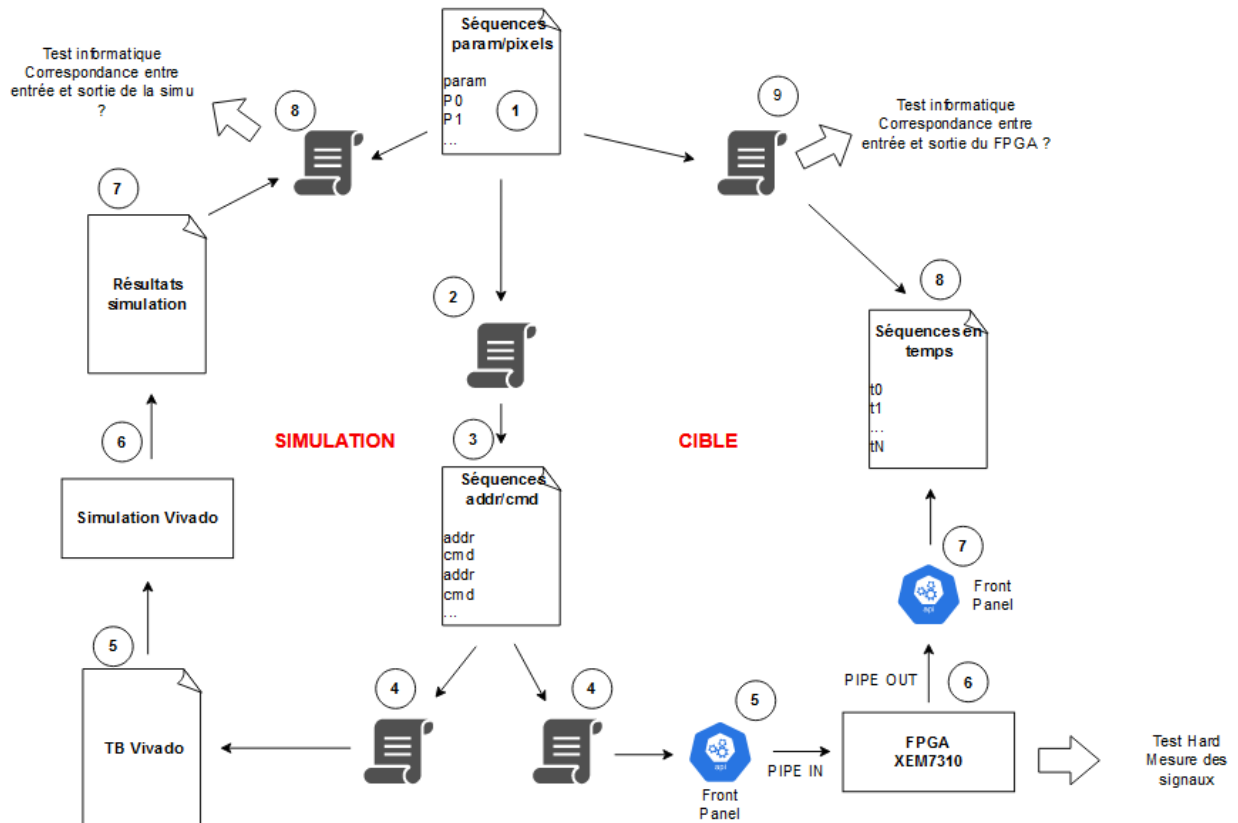


Figure 15 : Environnement de test du Firmware RAS

7.1. Validation par simulation

La simulation se fait grâce à l'outil de développement de Firmware Vivado. Pour faire une simulation, on utilise un fichier VHDL Test Bench dans lequel on a ajouté les process de simulation des composants du Front Panel Opal Kelly. Cet ajout permet de simuler les composants implantés sur la carte tels que le Pipe In ou le Pipe Out et obtenir des résultats de simulation proches de la réalité. La génération du fichier VHDL Test Bench se fait automatiquement à partir d'un script python qui récupère les commandes saisies manuellement. Une fonction est également ajoutée au fichier VHDL Test Bench pour récupérer les résultats de simulation dans un fichier texte. Ce fichier de résultats est ensuite manipulé afin de le comparer avec les commandes fournies avant la simulation.

7.2. Validation sur cible

Une fois la synthèse et l'implémentation du Firmware réalisées à l'aide de l'outil Vivado, on peut générer le fichier de configuration que l'on vient charger sur la carte depuis l'API du Front Panel. Une fois la cible configurée, on peut envoyer les commandes et récupérer les signaux de sortie à l'aide du Front Panel comme le décrit la partie précédente. Le but dans la phase de test est de vérifier si les signaux observés à l'oscillateur et dans le fichier de sortie du Pipe Out correspondent à ceux attendus.

Pour la solution Hardware, il suffit de récupérer les signaux sur les pins alloués et utiliser un analyseur logique pour les visualiser sur l'oscillateur, la vérification consiste ensuite en l'observation et la mesure des séquences observées. Il faut ainsi valider chacune des exigences relatives de la version testée.

Pour la solution gérée par les scripts Python, il suffit de récupérer le fichier contenant les données en sortie de la carte, un premier script permet d'obtenir un fichier texte lisible et un second procède à la manipulation des données et à la comparaison entre les résultats et les commandes saisies au préalable.

8. Résultats

Les différentes versions du Firmware livrées ont été testées à l'aide de la procédure précédemment expliquée. Les séquences envoyées sont décrites en figure 16, les mêmes séquences sont envoyées pour la simulation et les tests sur carte, elles sont lues de droite à gauche.

```

Row 0  0000000000001000000001000000001000000001
Row 1  0000000000001000000001000000001000000010
Row 2  00000000000010000000010000000010000000100
Row 3  000000000000100000000100000000100000001000
Row 4  000000000100000000010000000010000000010000
Row 5  00000001000000000100000000100000000100000
Row 6  00000010000000001000000001000000001000000
Row 7  00000100000000010000000010000000010000000
Row 8  00001000000000100000000100000000100000000
Row 9  000000000000000000000000000000000000000111111111
Row 10 0000000000000000000000000000000000000001111111111000000000
Row 11 0000000000000000000000000000000000000000000000000000000000
Row 12 0000111111111100000000000000000000000000000000000000000000

```

Figure 16 : Séquences des rows envoyées

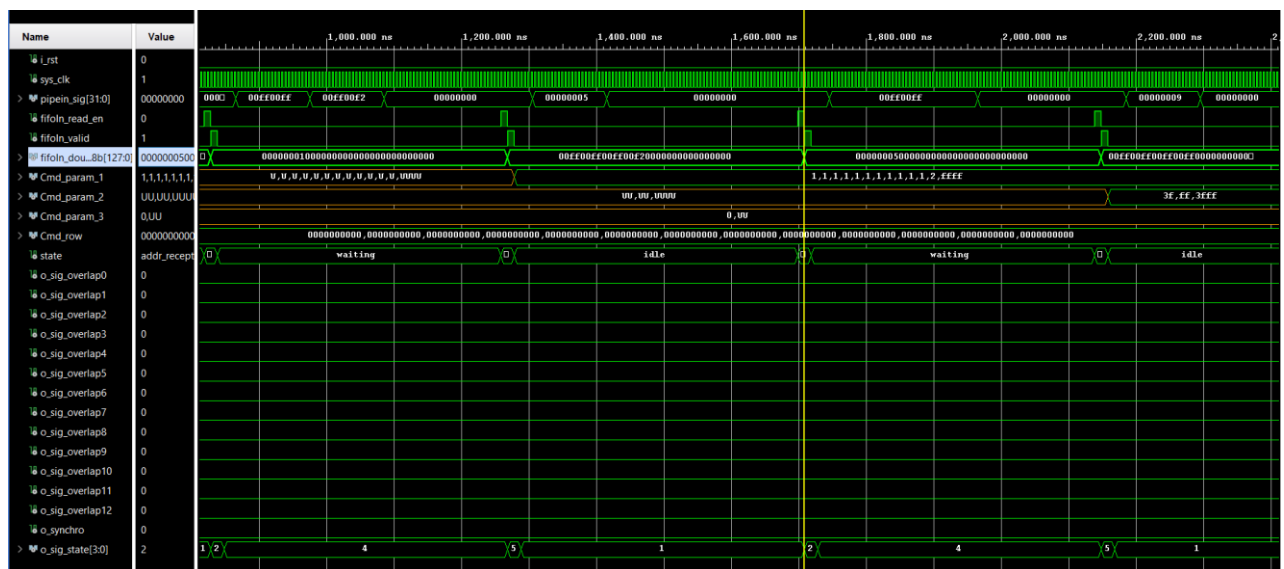


Figure 17 : Simulation partie réception

Dans un premier temps, les résultats obtenus sont ceux de simulation. Sur la figure 17, on observe la phase de réception des commandes, on remarque que le Pipe In reçoit des données et grâce aux signaux `fifoln_rd_en` (read enable de la FIFO en entrée) et `fifoln_valid` (valid de la FIFO en entrée) on récupère les données en sortie de la FIFO. On vient récupérer ensuite dans ce signal de sortie les données utiles et on stocke les adresses et les commandes, ainsi on remplit les registres. Dans la figure, on voit que l'on stocke dans un premier temps dans le premier registre de paramètre puis dans le second.

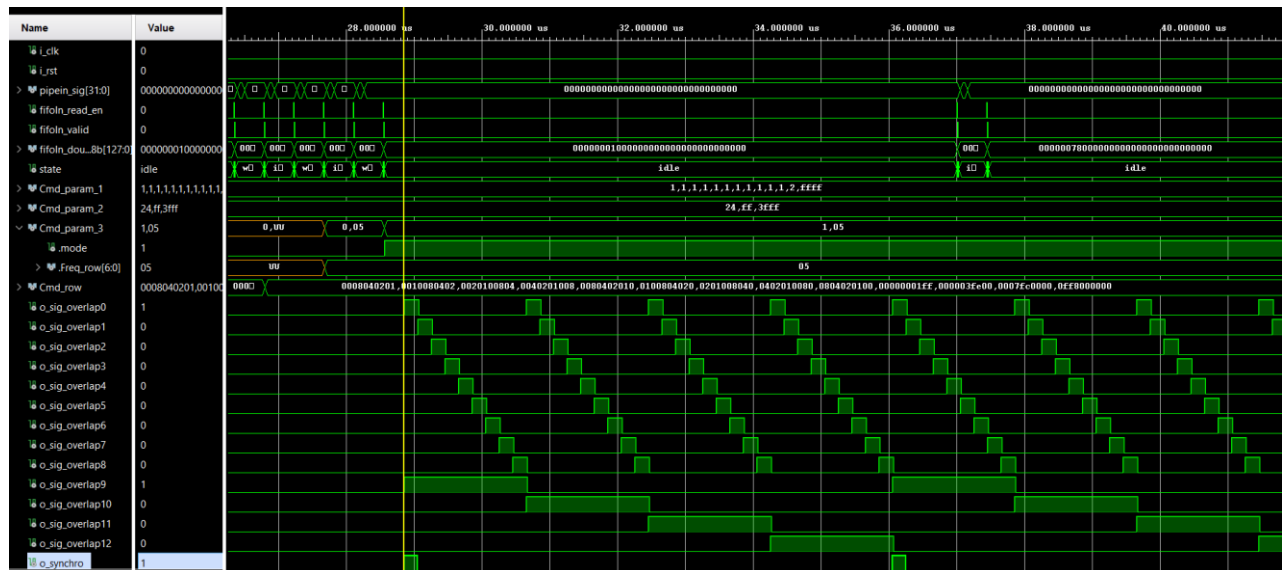


Figure 18 : Simulation partie pilotage

Sur la figure 18, on observe, toujours en simulation, la phase de pilotage du module RAS. Une fois l'ensemble des registres remplis et le paramètre MODE passé à '1', le module rentre en phase de pilotage en suivant les séquences stockées dans le registre de séquence de l'interrupteur. Le module applique également les différents paramètres que l'utilisateur a saisis comme le temps de chevauchement entre 2 interrupteurs ou la fréquence de fermeture des interrupteurs. Dans la figure cette fréquence est égale à 5MHz et on voit les 9 lignes s'activer pendant une activation d'un cluster. C'est ainsi que l'on balaye 36 pixels en influant sur l'activation de seulement 13 interrupteurs.

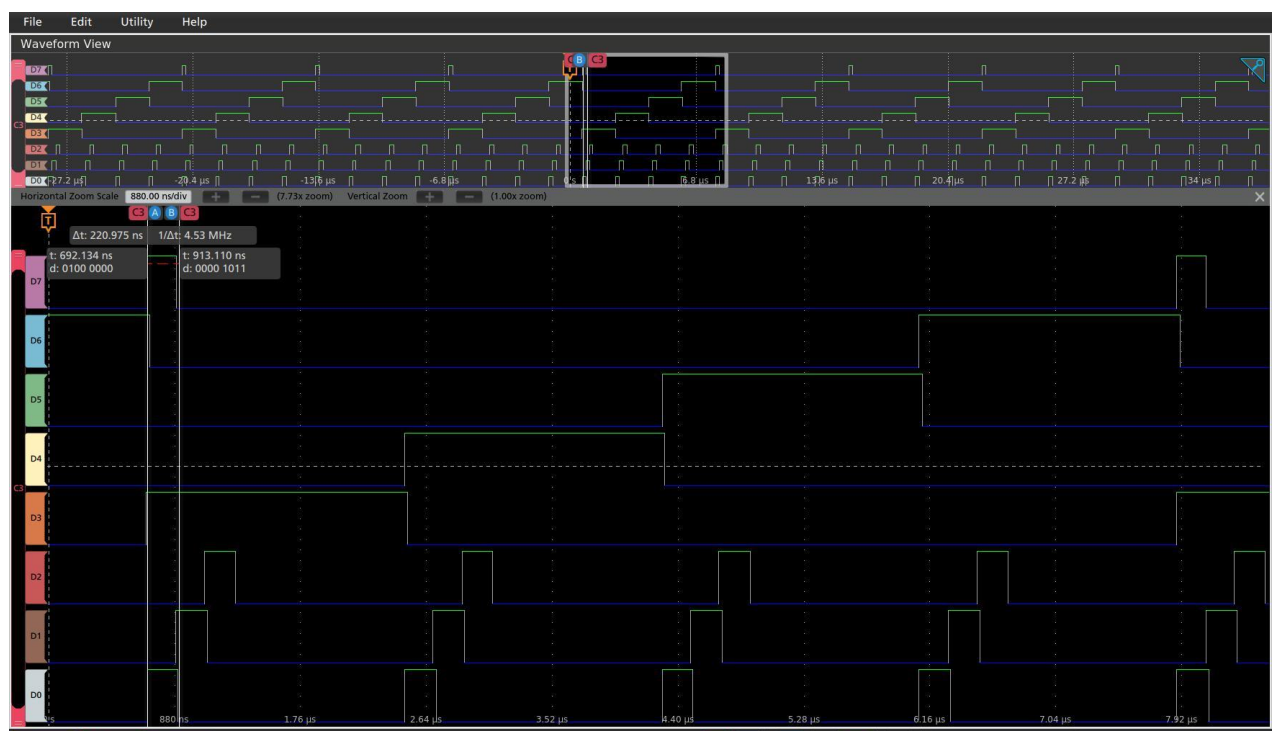


Figure 19 : Mesure du temps d'activation d'un row avec REV = 2 et freq_row = 5 MHz

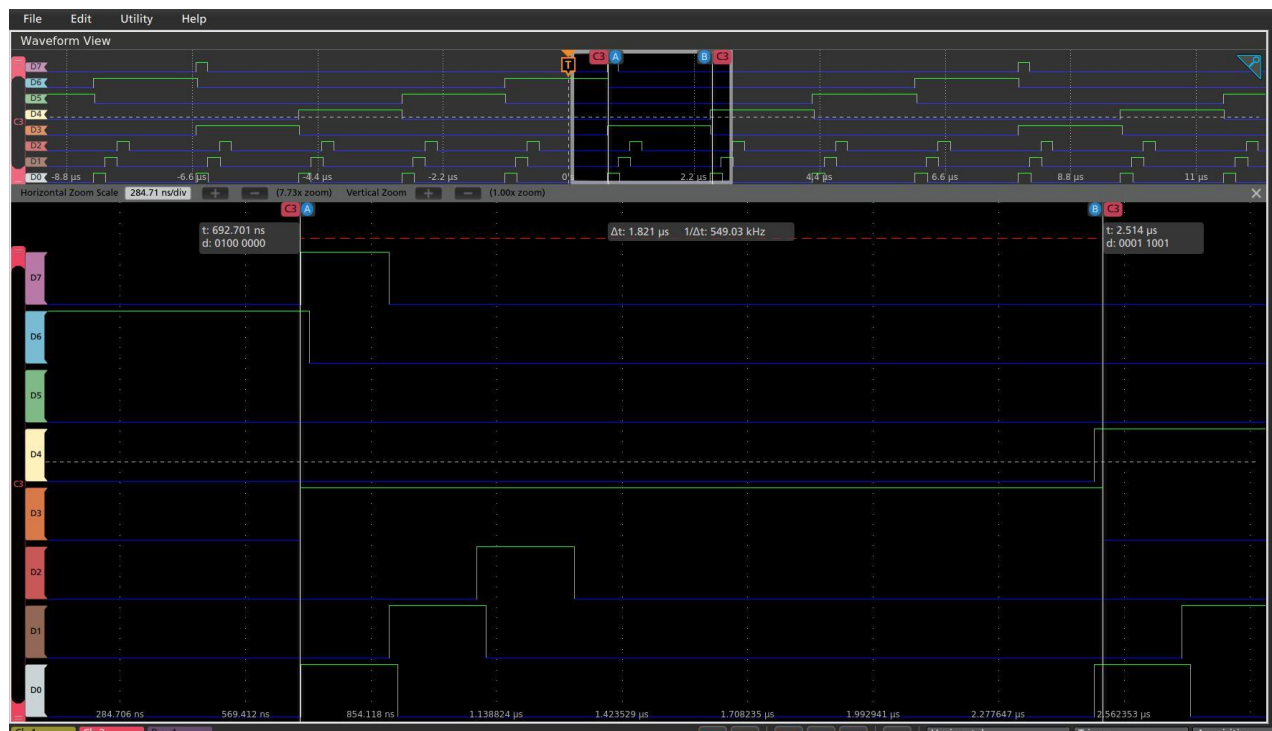


Figure 20 : Mesure du temps d'activation d'un cluster avec REV = 2 et freq_row = 5 MHz

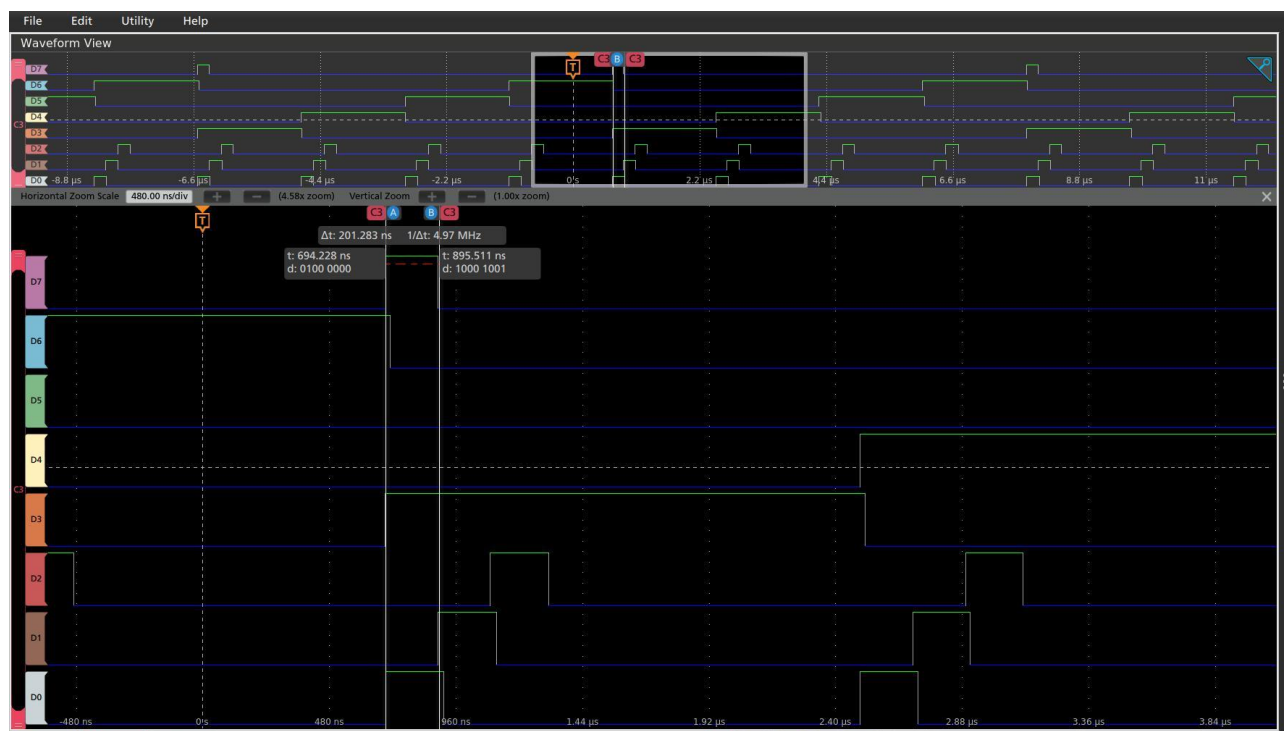


Figure 21 : Mesure du temps d'activation du signal synchro avec freq_row = 5 MHz

Une fois les simulations faites, on a pu tester le module en configurant la cible FPGA, les résultats informatiques n'étant pas très parlant, on montrera seulement les résultats récupérés à l'oscilloscope. On ne récupère que les signaux de sortie de pilotage ainsi que le signal de synchronisation, pour cela on vient brancher un analyseur logique sur les pins alloués aux signaux. Les mesures ne sont pas exactement précises car on place les curseurs à la main, mais donnent cependant une idée de la justesse du module. Sur la figure 19, on peut observer la mesure du temps de fermeture d'un interrupteur avec les paramètres REV = + 2 et

$\text{freq_row} = 5 \text{ MHz}$, on mesure environ 220 ns ce qui correspond bien à $T_{\text{row}} = 200 \text{ ns}$ auxquels on ajoute les 20 ns de chevauchement. Sur la figure 20, on mesure le temps de fermeture d'un interrupteur de cluster, celui-ci est fermé durant l'activation des 9 lignes successives c'est-à-dire pendant 1.8 μs , à cela on ajoute 20 ns de chevauchement. Enfin dans la figure 21, on mesure le temps à '1' du signal de synchronisation, celui est bien égal à 200 ns comme voulu en saisissant $\text{freq_row} = 5 \text{ MHz}$.

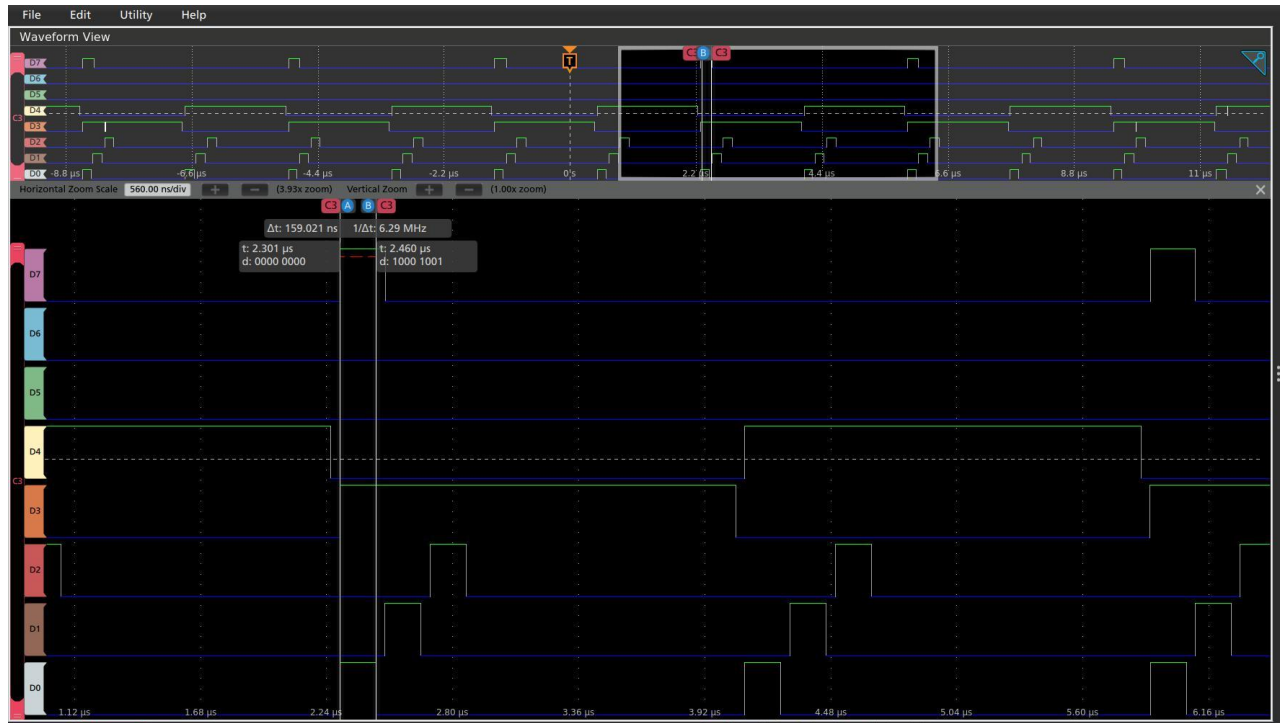


Figure 22 : Mesure du temps d'activation d'un row avec $\text{REV} = -4$ et $\text{freq_row} = 5 \text{ MHz}$

On peut aussi vouloir vérifier que la longueur de la séquence est réglable et que le chevauchement peut être négatif, on voit cela dans la figure 22. Les paramètres saisis sont la longueur de séquence = 18 et $\text{REV} = -4$. On remarque en effet que la séquence s'arrête après l'activation de 2 clusters et non 4 comme précédemment ($2 \text{ clusters} * 9 \text{ rows} = 18$). On remarque aussi que le temps de fermeture d'un interrupteur est maintenant égal à environ 160 ns ce qui est bien égal à 200 ns moins 40 ns de chevauchement négatif.

Les résultats correspondent donc aux attentes et aux exigences.

9. Evolutions de la fonction

Les prochaines évolutions portent sur les exigences que devra respecter la fonction RAS. Dans un premier temps, le Firmware devra être capable de piloter des DACs (Digital to Analog Converter) afin de piloter les interrupteurs. Pour cela, on utilisera des DAC121S101 12 bits de Texas Instrument avec un protocole SPI (Serial Peripheral Interface).

Le Firmware RAS devra aussi être capable de récupérer des signaux Hardware sur la carte. Pour cela, il faudra récupérer des signaux en sortie d'ADC (Analog to Digital Converter) et les stocker dans des registres que l'on pourra aller lire sur commande (de la même façon que les HK sur les registres soft). Cette évolution ne sera possible qu'une fois que le prototype de la carte Hardware du RAS sera mis en place.

Une fois l'ensemble de ces exigences respectées, testées et validées sur le banc de test, la fonction RAS sera prise en charge par un laboratoire de recherche de République Tchèque qui développera la version de vol de la fonction et qui sera intégrée sur l'instrument.

10. Perspectives personnelles

Cette année a été pour moi l'occasion de découvrir réellement le monde du travail dans l'électronique. Après les différents modules suivis en école, j'ai pu remarquer que l'électronique numérique est un domaine qui me plaît particulièrement, j'ai pu confirmer cela en travaillant sur FPGA tout au long de l'année et je pense également continuer à me spécialiser en électronique analogique pour pouvoir être polyvalente en électronique en général. La mission confiée ne constituait pas un travail d'équipe mais j'ai tout de même été intégrée dans l'équipe du module dans lequel se place la fonction que je devais développer. J'ai ainsi pu voir comment s'organisait le travail entre les différents membres et profiter du domaine d'expertise de chacun à la moindre de mes interrogations. Cette façon de travailler en équipe me paraît essentielle et je souhaite m'orienter vers des projets d'équipe plus tard.

La recherche du stage de césure a été un peu compliquée pour ma part puisque peu de temps avant de partir, le stage que je devais faire à l'étranger a été annulé à cause de la situation sanitaire. J'ai donc dû rechercher un nouveau stage très rapidement, c'est de cette façon que j'ai eu écho d'un stage dans un laboratoire de recherche en astrophysique. C'est un monde que je ne connaissais pas du tout et où j'avais tout à apprendre. Cette période de césure m'aura permis de découvrir le domaine du spatial vers lequel je pense essayer d'orienter ma carrière. Je ne pense cependant pas que ce sera dans la recherche mais plutôt dans l'industrie où j'aimerais réaliser mon stage de fin d'étude l'an prochain.

11. Conclusion

Lors de ce stage j'ai pu être intégrée dans l'équipe responsable du développement de l'électronique numérique de lecture (DRE) du projet Athena X-IFU. Ce stage a porté sur le développement de blocs FPGA pour la fonction Row Addressing and Synchronization du DRE ainsi que sur l'environnement de test qui permet la vérification du module. Il a ainsi consisté en le développement de fonctions VHDL pour le FPGA du RAS ainsi que de fonctions Python de test. Une fois ces développements réalisés j'ai pu tester par simulation dans un premier temps puis par configuration de la cible FPGA les modules. Durant cette période, j'ai également pu participer à l'amélioration du système de gestion de configuration en utilisant les outils proposés afin de trouver la meilleure méthode de gestion pour l'équipe permanente. Ce stage a donc couvert un large spectre d'activité allant d'aspects touchant à la gestion de configuration à d'autres plus techniques.

12. Bibliographie


<https://www.the-athena-x-ray-observatory.eu/resources/gallery.html>

<https://www.irap.omp.eu/>

<http://x-ifu.irap.omp.eu/>

<https://www.youtube.com/watch?v=JNUWOQwLkqk>

<https://www.youtube.com/watch?v=u369Otpg4pw>

| | | |
|---|--|---|
|  | 4 ^{ème} de couverture | Date du stage Du 08/09/2020 au 31/08/2021 |
| | Stage d'exécution <input type="checkbox"/> Stage élève – ingénieur <input type="checkbox"/> Stage année en entreprise : <input checked="" type="checkbox"/> 12 mois <input type="checkbox"/> 1 ^{er} semestre <input type="checkbox"/> 2 ^{ème} semestre Stage projet de fin d'études <input type="checkbox"/> | |
| | Nom et Prénom du stagiaire : ROLLAND Noémie | |

Spécialité : ☐ CGP ☒ ETI

RESUME (300 mots minimum, précisez : objectifs, moyens, résultats, conclusions, perspectives. Attention ce résumé ne doit pas contenir d'informations confidentielles)

Le stage effectué durant cette année de césure s'est déroulé au sein de l'IRAP (Institut de Recherche en Astrophysique et en Planétologie) dans le groupe GAHEC (Galaxies, Astrophysique des Hautes Energies et Cosmologie). J'ai pu être intégrée au sein de l'équipe responsable du développement de l'électronique numérique de lecture du projet Athena X-IFU.

L'observatoire Athena est une mission de l'ESA répondant aux problématiques de l'univers chaud et énergétique, il embarque à son bord deux instruments : le WFI et le X-IFU. Le spectro-imageur X cryogénique X-IFU utilise des détecteurs de type microcalorimètres TES opérés à 55 mK. Cet instrument est développé par un large consortium d'une trentaine de laboratoires en Europe et aux Etats-Unis.

L'objectif de ce stage est le développement d'un prototype de la fonction Row Addressing and Synchronization sur FPGA ainsi que de son environnement de test pour en démontrer la faisabilité. La fonction RAS aura pour rôle d'adresser chacun des pixels de la matrice de détection séquentiellement par la méthode TDM (Time Domain Multiplexing). Pour cela le travail s'est partagé autour de trois activités principales : le développement des modules pour le FPGA et de son environnement de test, la validation des modules par simulation et la validation des développements du FPGA par test. La technologie choisie pour le développement de ce prototype est le module d'intégration FPGA USB3.0 compact XEM7310-A75T Opal Kelly doté d'un FPGA Xilinx Artix-7. L'ensemble des modules pour le FPGA sont développés en VHDL et l'environnement de test est développé en Python.

Une fois les premiers développements finalisés, nous avons pu obtenir des premiers résultats, dans un premier temps par simulation puis directement sur FPGA, ces résultats nous ont permis de vérifier que la fonction RAS répondait effectivement aux contraintes imposées au départ et pourra être testée sur un banc de test pour piloter directement l'électronique contenant les TES provenant des Etats-Unis. Malgré une première observation positive de la fonction RAS quant à sa faisabilité, il reste encore certaines fonctionnalités à ajouter pour que la fonction réponde entièrement aux exigences. Une fois l'ensemble de ces exigences respectées, la fonction RAS sera prise en charge par un laboratoire de recherche de République Tchèque qui développera la version de vol de la fonction pour l'intégration sur l'instrument.

Cette année de césure a donc été pour moi l'occasion de découvrir le monde du spatial que je ne connaissais alors pas du tout, c'est un domaine qui m'a beaucoup plu et intéressé et où je souhaiterais

évoluer à l'avenir. Le stage m'a également confirmé mon attrait pour l'électronique numérique et l'électronique en général.