CONTRÔLE ET SYNCHRONISATION D'INSTRUMENTS EN MICROSCOPIE

Simon Duthoit

Polytech' Lille

IMA 5

Projet de Fin d'Études

Chef de projet : François Anquez

Tuteur : Samuel Hym

Tuteur : Jean Rodiguez

Table des matières

1. Remerciements

Je voudrais commencer ce rapport par quelques mots de remerciements aux personnes qui ont consacré de leur temps et apporté leur aide pour m'aider à la réalisation de mon projet.

Tout d'abord, j'adresse mes remerciements à Francois Anquez, du PhLAM qui a proposé le sujet de projet sur lequel j'ai travaillé et m'a accepté dans l'équipe. Je tiens également à remercier Jean Rodriguez, également du PhLAM, pour sa disponibilité et son aide très précieuse sur la partie technique. Il y a aussi Samuel Hym que je remercie, qui a été mon tuteur et m'a suivi tout au long de ce projet.

Enfin, je n'oublie pas Thierry Flamen de l'atelier électrique/électronique de Polytech' Lille pour ses bons conseils ainsi que sa présence pour me permettre d'accéder aux salles de Polytech'Lille dont j'avais besoin durant mon projet.

2. Introduction

Ce document a pour but de présenter le travail que j'ai réalisé pour mon PFE (Projet de Fin d'Etudes), en tant qu' élève-ingénieur IMA 5. La petite particularité liée à mon PFE est que je n'ai pas fait le S9 en France et n'ai donc commencé qu'à la rentrée 2015.

Le PFE a une importance capitale dans la formation de l'ingénieur. En effet, situé temporellement entre les derniers cours théoriques et le stage de fin d'études, il permet de mettre en pratique une dernière fois, avant de se lancer dans le monde de l'entreprise / recherche, les connaissances acquises tout au long de la formation sur une réalisation d'ampleur considérable au vu du temps alloué. Plus que ça, il requiert aussi des qualités comme l'autonomie, la persévérance et l'innovation pour le mener à bien.

Le sujet que j'ai choisi s'intitule “Contrôle et Synchronisation d'Instruments en Microscopie”. J'expliquerai en quoi il consiste dans la partie suivante. Je l'ai choisi car il touche au domaine médical, qui m'attire car j'ai suivi un cours sur le Traitement d'Images et les différentes techniques d'Imagerie Médicale aux USA au S9, et lie ce dernier à ma spécialité d'études.

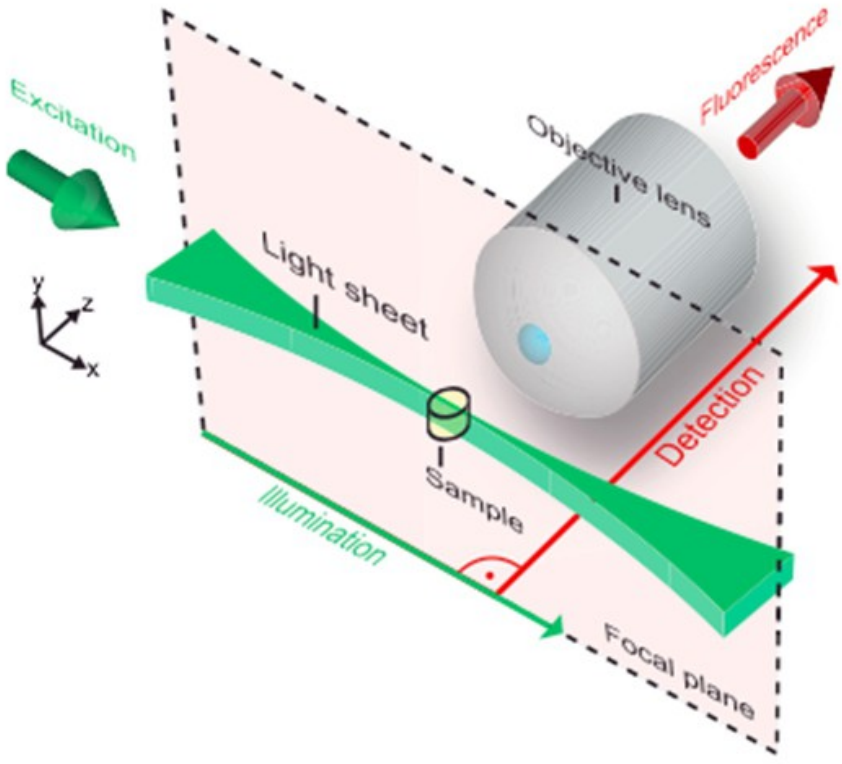
Pour exposer mon PFE et ce que j'ai réalisé, je vais tout d'abord expliquer le sujet dans son contexte et ce qu'il était attendu que je réalise. Je vais ensuite parler des moyens que j'avais à disposition pour travailler ainsi qu'expliciter la chaîne de développement retenue et le rôle de chacun de ses maillons. Je parlerai, après, des différentes réalisations effectuées ainsi que de ce qu'il reste à faire. Je dresserai enfin une conclusion au vu de cette expérience.

Il faut noter, que dans la partie où les réalisations sont exposées, des références à des programmes (C, VHDL) seront faites. Pour des raisons de confort visuel et de place, il a été décidé de ne pas les verser inutilement en annexe. En revanche, j'indique où ils sont visibles sur la page d'avancement Wiki de ce projet qui est le numéro 19 (http://projets-imasc.plil.net/mediawiki/index.php?title=P19_Contr%C3%B4le_et_synchronisation_d%27instruments_en_microscopie).

3. Présentation générale du projet

Pour rappel, le sujet de mon PFE est intitulé “Contrôle et synchronisation d'instruments en microscopie”. Le contexte est le suivant : au PhLAM (Laboratoire de Physique des Lasers, Atomes et Molécules), une équipe de chercheurs “Dynamique de la réponse cellulaire au stress” conduit des expériences sur la réponse à des stress de cellules vivantes. Plus précisément, des échantillons de cellules vivantes sont soumis à différents types d'excitation et les réactions sont étudiées pour comprendre comment ces cellules vivantes peuvent se défendre : quels moyens elles mettent en œuvre. Cela permet, notamment, des avancées dans la thérapie des cancers, et des tumeurs plus exactement.

Pour observer ces phénomènes, cette équipe de chercheurs assemble un microscope qui utilise la technologie de microscopie de fluorescence par feuilles de lumière. Le principe de ce type d'imagerie est le suivant : on produit une nappe de lumière infiniment fine, en théorie, sur une section de l'échantillon de cellules à étudier. Cette nappe est déplacée perpendiculairement à l'objectif du microscope qui image l'échantillon, sur d'autres sections de ce dernier. Ainsi, une image 3D peut être reconstituée au terme de l'opération, à condition bien sûr d'avoir imagé suffisamment de sections différentes de l'échantillon. Ce principe est résumé par la figure ci-dessous.



Les différents composants de ce microscope (micro-miroirs, caméra, objectif...) nécessitent d'être commandés par des signaux électriques et, également dans certains cas, d'être synchronisés entre eux (à l'ordre de la dizaine de µs). Lesdits signaux doivent pouvoir être choisis par l'opérateur qui peut les fabriquer à partir de motifs de base (triangles / trapèzes, carrés / rectangles, exponentiels) pour réaliser des signaux plus complexes.

En découlent, après cette première approche du projet, les tâches à réaliser :

-développer sur le PC hôte une application permettant à l'utilisateur qui va utiliser le microscope, de construire les signaux qu'il souhaite utiliser, à partir de motifs de base tels que précédemment mentionnés.

-implémenter sur une carte contenant un FPGA (Field Programmable Gate Array) -choix retenu au lancement du projet- une solution permettant, à partir du choix de l'utilisateur, d'interpréter ce dernier et de les envoyer à des DAC (Digital to Analog Converter) pour produire les signaux désirés et commander le microscope.

Ne travaillant qu'environ un mois trois quarts sur le projet, l'objectif que mes encadrants m'ont fixé est de permettre à un utilisateur de choisir un des trois motifs de base pour en faire un signal périodique (signal carré périodique par exemple) sans se soucier de la beauté de l'interface graphique (la console suffit). Il faut ensuite transmettre ce choix à la carte avec le FPGA qui va l'interpréter et l'envoyer à un DAC pour produire le signal. A noter que l'application sur le PC hôte devra être codée en langage C uniquement car elle doit être compatible, dans le futur, avec LabView.

Dans la partie suivante, je vais détailler la chaîne que j'ai à réaliser et expliquer le rôle de toutes ses constituantes en commençant par exposer les moyens que j'avais à disposition pour travailler.

4. Présentation technique du projet

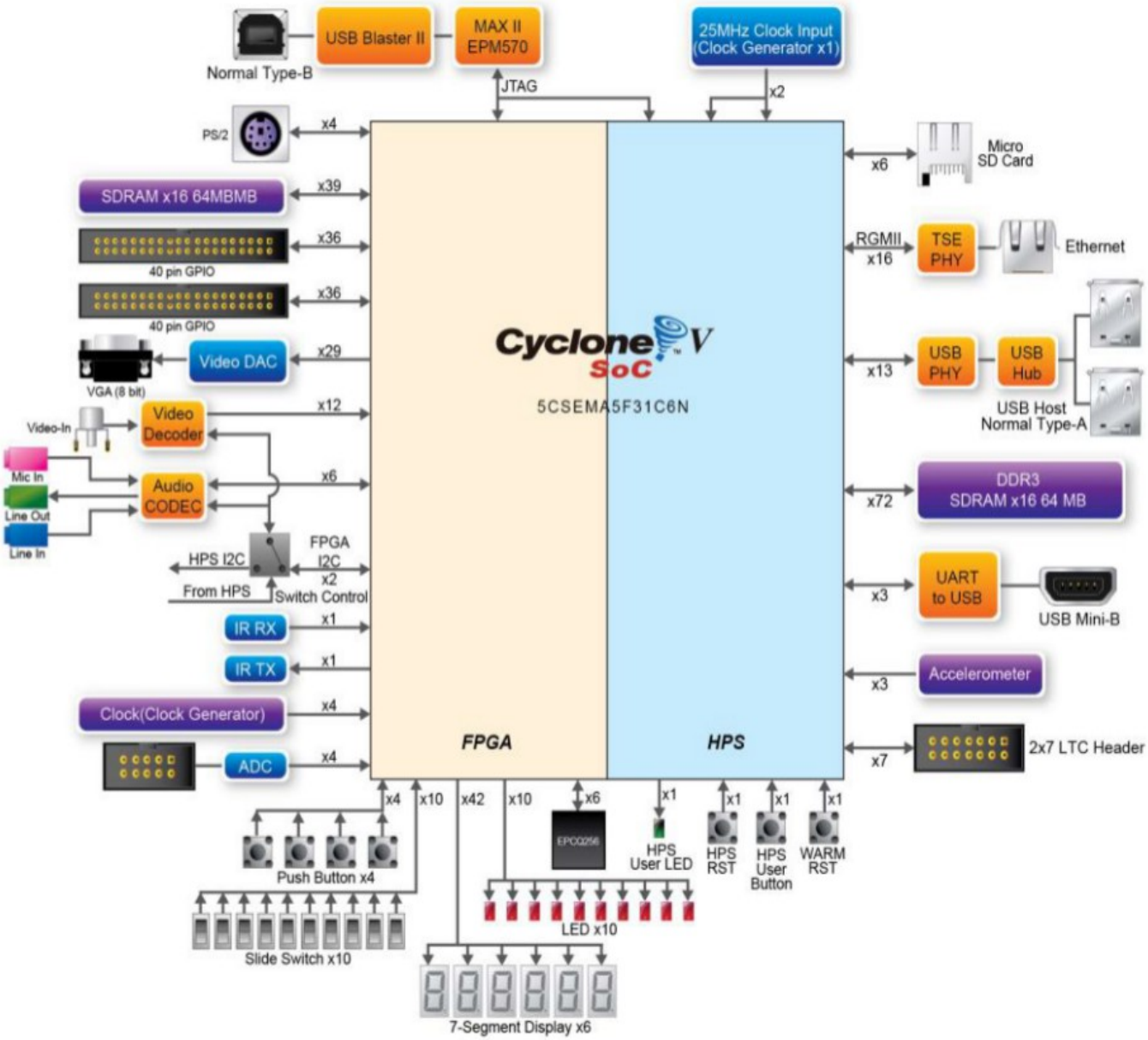
A. Moyens et ressources disponibles

Avant de commencer le projet, mes encadrants m'ont donné :

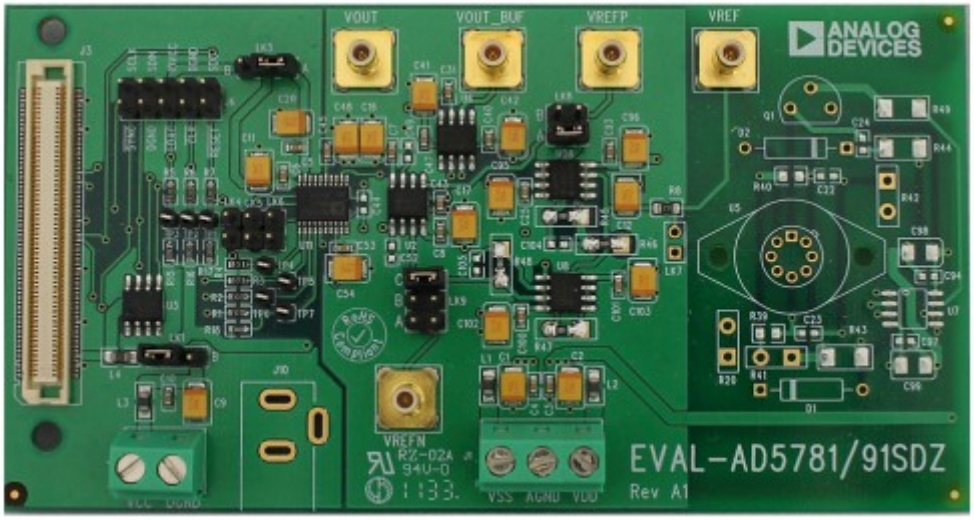
-une carte DE1 SoC développée par Altera et construite par Terasic

-une carte d'évaluation du DAC AD5791 d'Analog Devices.

La carte DE1 SoC est une carte dont le composant principal est un FPGA de la famille Cyclone V. Dans la puce qui contient le FPGA, se trouve également un processeur ARM Cortex-A9 double cœur. Autour de cette puce s'articulent une panoplie d'interfaces telles que des afficheurs 7 segments, de la RAM, un lecteur de carte micro SD, un port VGA … Elles sont reliées soit au processeur ARM ou au FPGA. La figure ci-dessous résume tout ce que propose la carte.



La carte d'évaluation du DAC AD5791 est, comme son nom l'indique, une carte permettant de découvrir le DAC AD5791 sans avoir à router sa propre carte. Ce DAC offre une conversion de mots de 20 bits vers une tension analogique qui peut s'étendre de -16.5 à 16.5 V. La communication avec la carte utilise le protocole SPI (Serial Peripheral Interface) avec une horloge pouvant monter jusqu'à une fréquence de 35 Mhz. La précision de conversion est de 1 ppm. Voici à quoi la carte ressemble ci-dessous.



En ce qui concerne les outils logiciels, j'ai utilisé :

-MinGW qui est un environnement Linux tournant sur Windows me permettant de me servir du compilateur GCC pour les programmes en C.

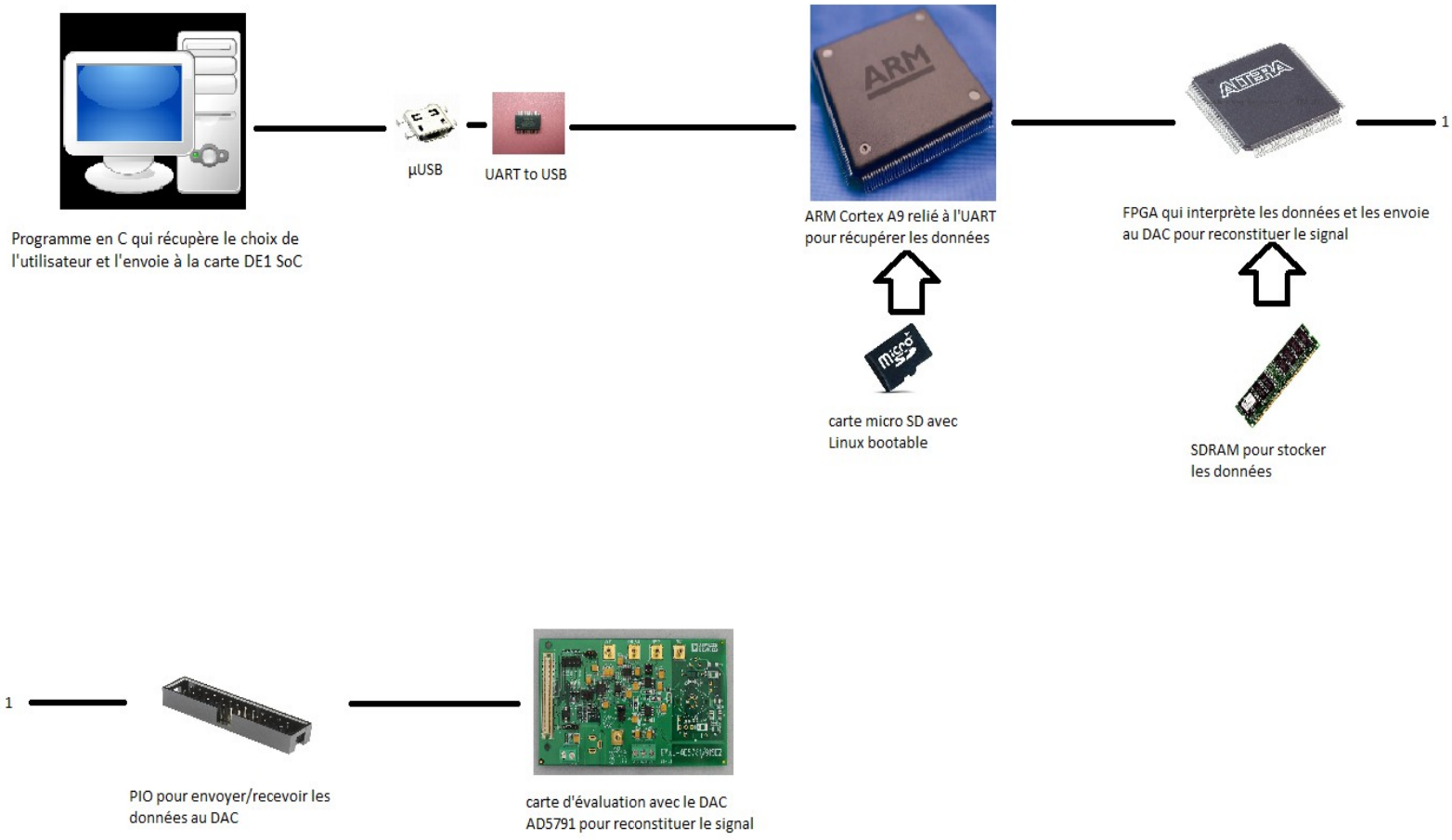
-La suite de logiciels d'Altera pour travailler sur la carte DE1 SoC, contenant Quartus pour les projets à implanter dans le FPGA et Embedded Command Shell pour compiler et transférer les programmes sur le processeur ARM.

-Win32 Disk Imager, logiciel permettant d'écrire des images sur une carte SD (micro SD ici) branchée sur un PC ayant Windows en tant qu'OS.

-Putty, terminal de communication permettant d'envoyer et de recevoir des données, avec notamment des ports COM, ce qui nous intéresse dans le cadre de ce projet.

B. Chaîne de développement

Je vais maintenant apporter plus de précisions sur les tâches que j'ai eu à réaliser lors de mon PFE. Avant de me lancer dans une quelconque réalisation, j'ai établi une chaîne de développement pour avoir une découpe précise de ce que j'avais à faire ainsi qu'une bonne vue d'ensemble. La voici ci-dessous.



Détaillons maintenant chaque maillon de la chaîne :

-le premier a déjà été évoqué précédemment. Le PC hôte va permettre à l'utilisateur, via un programme en langage C, de choisir les signaux qui vont commander les composants du microscope. Ils seront construits à partir de 3 motifs de base qui sont le trapèze / triangle, le carré / rectangle et l'exponentielle. En fonction du nombre de points choisi par l'utilisateur et bien évidemment du signal, une suite de points, correspondant aux mots de 20 bits à convertir par le DAC, et représentant le signal, sera générée avec d'autres informations inhérentes à ce dernier tels que les amplitudes crêtes, la période et la synchronisation éventuelle avec d'autres signaux. Une mise en forme des données doit être adoptée pour que leur interprétation sur la carte DE1 SoC soit possible.

-Ces données, une fois générées sont transmises à la carte DE1 SoC par une liaison d'un port USB du PC hôte vers le port µUSB de la carte auquel est connecté un convertisseur de niveaux logiques UART to USB (FTDI232R pour être plus précis), lui-même relié au processeur ARM de la carte.

-Au vu de la précédente connexion, il faudra écrire un programme pour le processeur ARM qui va permettre de réceptionner les données venant du PC hôte. Il faut noter que le processeur nécessite de fonctionner sur un environnement Linux dont l'image bootable doit être écrite sur une carte micro SD connectée à la carte DE1 SoC par son lecteur de cartes micro SD. Quoique programmable en assembleur, le langage retenu pour l'ARM a été le C du fait des librairies fournies par Altera qui implémentent déjà des fonctions pour exploiter l'UART. Une fois les données reçues, il faudra les transmettre au FPGA.

-Le FPGA va stocker les données dans la SDRAM qui lui est connectée en exploitant le pont physique entre lui-même et le processeur ARM. Une fois que l'opérateur choisit de commencer sa manipulation, le FPGA va interpréter les données selon le formalisme de données adopté et communiquer avec le DAC (protocole SPI).

-Les broches d'E/S servent évidemment à connecter les broches de la carte d'évaluation du DAC au FPGA pour que la communication soit possible.

-Enfin, cette même carte d'évaluation du DAC produit le signal que l'utilisateur a sélectionné au départ grâce aux trames (contenant les mots de 20 bits numériques) que lui envoie le FPGA.

Au vu de ce qui a été élaboré, on peut donc finalement distinguer les trois parties suivantes à réaliser :

-un programme en C sur le PC hôte qui va récupérer le choix de l'utilisateur quant aux signaux et envoyer les données sous un certain formalisme.

-un programme pour le processeur ARM en C également, qui tournera sous Linux, pour récupérer les données arrivant depuis le PC et vers l'UART (connecté à l'ARM).

-un programme pour le FPGA qui va ranger les données récupérées par l'ARM dans la SDRAM (qui est connectée au FPGA) et les lire puis les transmettre au DAC pour produire le bon signal.

Cela nous amène donc à la partie suivante qui va parler de ce que j'ai fait au niveau technique durant mon PFE.

5. Réalisations accomplies

A. Partie FPGA et DAC : Solution numéro 1

Je vais maintenant parler de la première solution réalisée pour que le FPGA envoie des données au DAC pour produire un signal. Comme expliqué précédemment, le FPGA doit interpréter les données stockées dans la SDRAM qui lui est connectée, décrivant un / des signaux, et donc envoyer des trames à un / des DAC pour qu'il(s) produise(nt) le(s) bon(s) signal / signaux.

Altera offre une possibilité avec ses FPGA, dont le Cyclone V, qui est d'utiliser la logique présente au sein du FPGA pour créer un microprocesseur (appelé Nios II), pouvant ensuite se programmer en assembleur ou en C et ayant la possibilité d'interagir avec les interfaces connectées au FPGA.

Qui plus est, Altera crée même une couche appelée HAL (Hardware Abstraction Layer) entre le Nios II et ses interfaces, ce qui permet au programmeur d'utiliser un langage C avec des fonctions standard tandis que la couche HAL s'occupe de la compatibilité avec les interfaces pour réaliser les instructions du programme. Un exemple parlant : si le programmeur veut stocker la valeur 30 à la case 5 d'un tableau d'entiers, il écrit : tab[5] = 30; . La couche HAL se chargera à partir de cette instruction, de générer toute la séquence d'instructions pour aller stocker 30 à la case 5 de ce tableau en mémoire (sélectionner la bonne adresse en écriture et le bon chip select, envoyer 30 sur le bus de données, …).

Pour maintenant comprendre comment se développe un système articulé autour du processeur Nios II, il est judicieux de regarder la figure qui se trouve à la page d'après.

En regardant cette figure, on peut voir qu'il faut d'abord déterminer ce qu'on utilise dans notre système. Il y a bien évidemment le processeur Nios II ainsi que les interfaces suivantes :

-la SDRAM pour stocker le programme et les données liées aux signaux.

-Les E/S parallèles pour la communication avec un / des DAC.

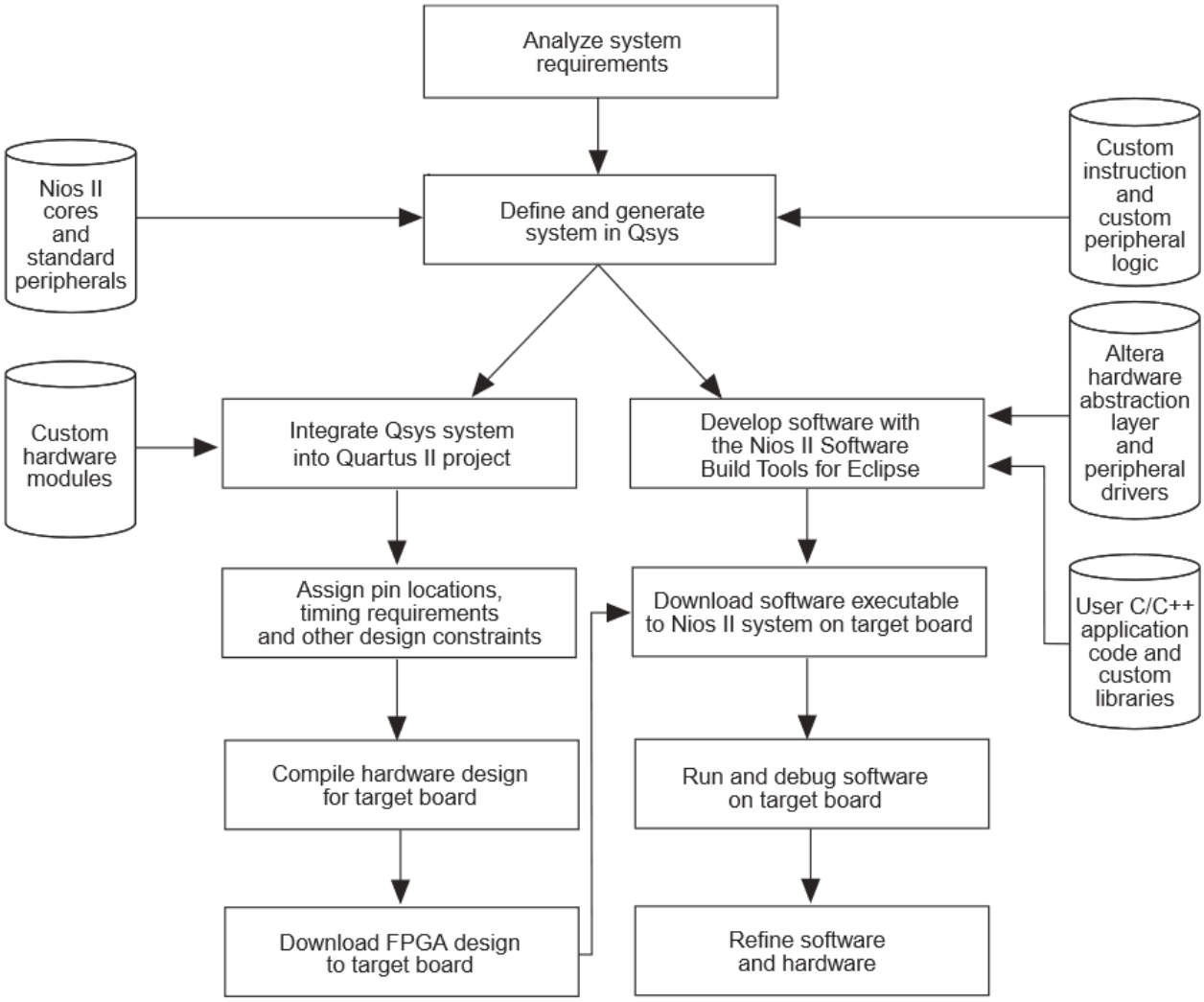
-un timer qui va nous servir d'horloge de référence pour cette même communication.

Après cela, en descendant jusqu'en bas de la colonne gauche de la figure, on peut voir qu'il faut utiliser Qsys, un outil intégré à Quartus, pour concevoir le système. Il suffit de simplement y sélectionner les composants que l'on vient d'énumérer et de les connecter graphiquement entre eux (cf Annexe 1) pour voir à quoi ressemble le modèle de ce système dans Qsys). Cela nous amène ensuite à la génération du modèle en Verilog / VHDL que l'on instancie et que Quartus va transformer en fichier à transférer dans le FPGA de la carte DE1 SoC. A l'issue de cette étape, notre

système est donc implémenté sur la carte.

Il ne faut pas oublier d'assigner les pins pour que les connexions entre le FPGA et les différentes interfaces soient valides. On peut également fixer des contraintes de temps comme par exemple la fréquence d'une horloge à une valeur bien définie.

Figure 1–2. Nios II System Development Flow



En regardant la colonne de droite de la figure ci-dessus cette fois, on voit qu'il faut ensuite écrire le programme, en C ici, en se servant de la couche HAL et si on le souhaite, des librairies Altera proposant déjà des fonctions pour les interfaces que l'on utilise. L'environnement de programmation Eclipse, intégré à Quartus, sert à programmer ainsi qu'à compiler et transférer le programme sur la carte pour compléter le développement de notre système.

Pour parler plus en profondeur de ce programme (qui se trouve sur le wiki à la fin de la semaine du 19 au 23 Janvier : CodeNiosSPIFonctionnel.txt), à l'appui sur un bouton poussoir, la routine d'interruption associée à ce bouton s’exécute et va activer les interruptions du timer, ainsi

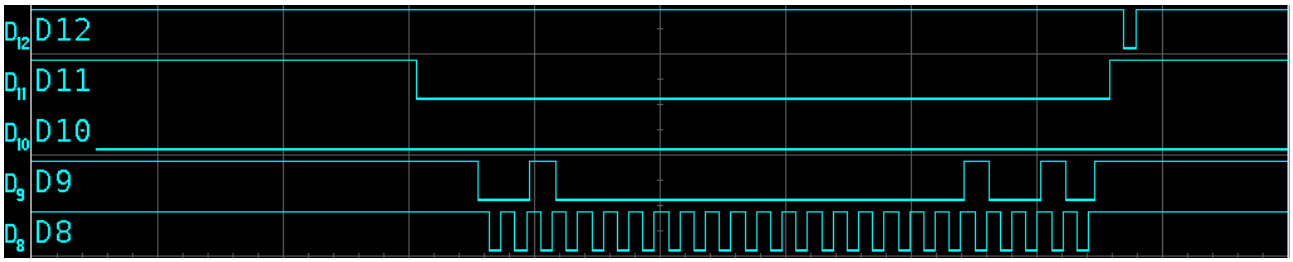
que l'initialiser, pour générer une horloge de fréquence donnée. On envoie ensuite une trame de 24 bits au DAC via une broche d'E/S de la carte DE1 SoC (1 bit de lecture / écriture, 3 bits de sélection de registre, 20 bits qui sont le mot à convertir). D'autres trames sont ensuite envoyées pour générer un signal périodique. Il faut aussi sur 3 autres broches envoyer :

-l'horloge.

-Le signal Slave Select qui indique si une transmission est en cours ou pas.

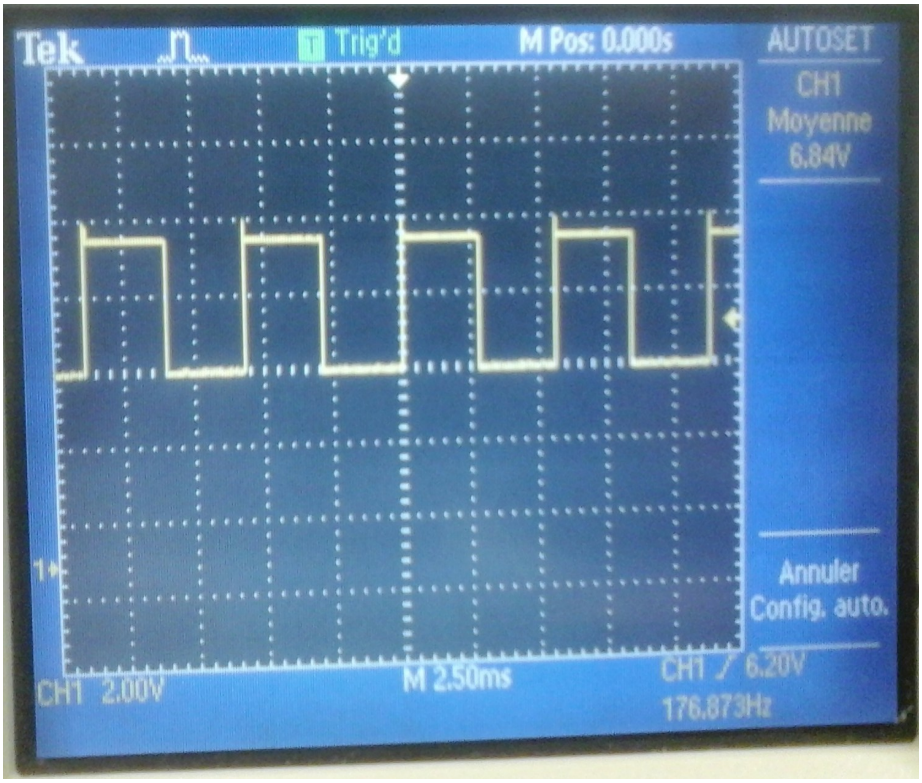
-Le signal LDAC, qui est l'entrée de mise à jour de la sortie du DAC.

Ci-dessous, on peut voir d'une part les 4 signaux générés par le système, qui illustrent une communication SPI, ainsi qu'un signal carré en sortie du DAC. En Annexe 2, on peut voir une image, tirée de la datasheet du DAC, qui montre les formes d'ondes à envoyer au DAC pour que la communication marche. Les temps annotés sont indiqués dans cette même datasheet mais non reportés ici, car non utiles à la compréhension.



D8 : Horloge **D9** : Trame de 24 bits **D11** : Slave Select **D12** : LDAC

Et pour illustrer le résultat, on peut voir ci-dessous un signal carré de rapport cyclique 50 % en sortie du DAC.



Malgré que cette méthode marche, on a décidé avec mes encadrants de l'abandonner. Bien que ce soit fonctionnel et, qu'une fois que la façon de développer un tel système est aisée quand les outils logiciels sont pris en main, cette solution présente des limites.

En effet, le premier gros désavantage constaté est en fait la rapidité. L'horloge générée par le timer ne dépassait pas quelques dizaines de kHz, ce qui est très lent. Un petit calcul rapide nous donne le résultat suivant : le signal le plus simple à générer est carré avec 2 points par période. Il faut pour cela, envoyer 48 bits par période au DAC. Avec une horloge à 50 kHz, cela signifie que la période du signal ne dépassera pas 1 kHz.

Enfin, le but est à terme de pouvoir contrôler plusieurs DAC en même temps car il ne faut pas perdre de vue qu'on contrôle plusieurs composants du microscope, chacun avec un DAC. Certains des signaux de commande devant être synchronisés entre eux à l'ordre de la dizaine de µs, une solution avec un microprocesseur est en fait difficilement envisageable. Car pour répondre à de telles contraintes temporelles, il faut que les communications avec les différents DAC soient faites en parallèle alors qu'un microprocesseur exécute son programme séquentiellement, instruction après instruction.

De telles limitations m'ont donc poussé à réaliser le système d'une manière différente, ce qui amène sa description dans la partie suivante.

B. Partie FPGA et DAC : Solution numéro 2

Abandonnant maintenant le processeur Nios II, il a été choisi de réaliser cette partie en VHDL. Le VHDL est un langage de description de circuits digitaux (logique combinatoire ou séquentielle). De fait, on s'affranchit donc de l'utilisation de Qsys et d'Eclipse même si le travail a été exactement le même. Il a fallu fabriquer le même « set » de 4 signaux que précédemment, selon le protocole SPI (cf Annexe 2 pour les caractéristiques requises par le constructeur) pour communiquer avec le DAC.

Le squelette d'un programme VHDL se présente toujours de cette forme :

-inclusion des librairies pour pouvoir utiliser certaines fonctions / fonctionnalités du langage.

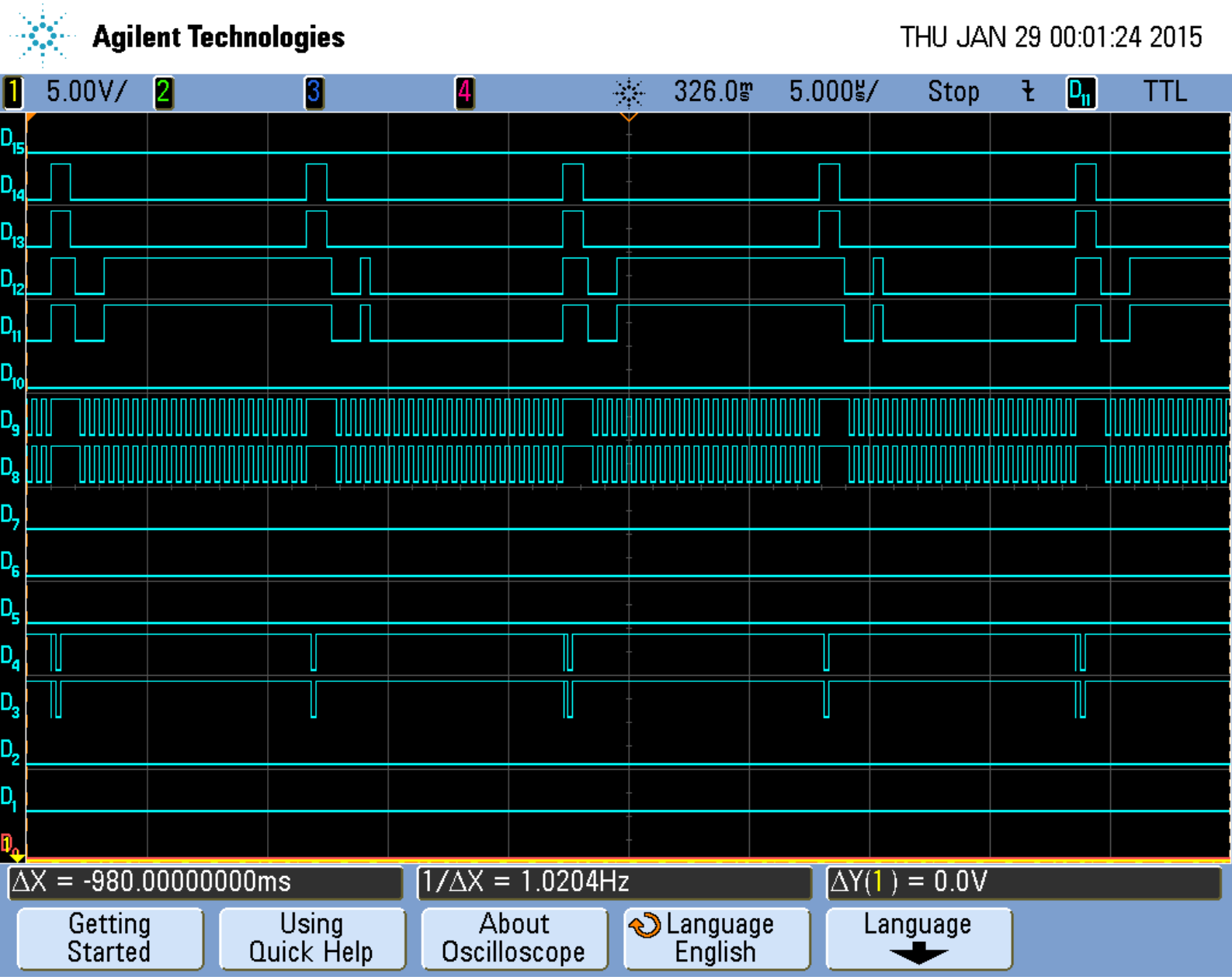
-L'entité où on déclare toutes les E/S du système.

-L'architecture où on décrit le comportement du système avec éventuellement, en son sein, des process qui s'éxécutent selon leurs listes de sensibilités pour décrire des comportements plus complexes (circuits séquentiels synchrones par exemple).

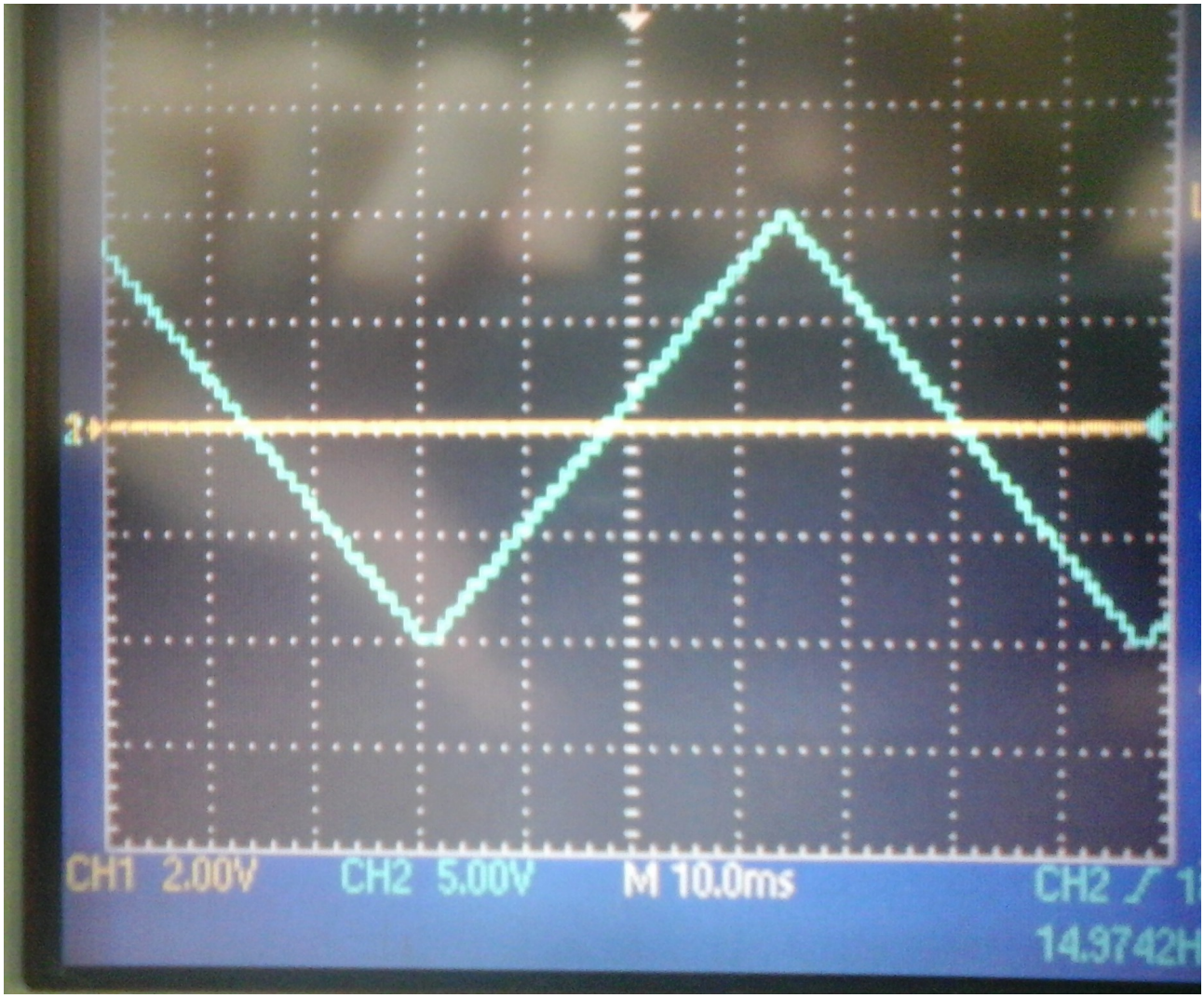
Cette partie a donc été implémentée selon cette mécanique. Le programme est disponible sur le Wiki à la fin de la semaine du 26 au 30 Janvier (SPISynchroBackup.txt). Un process sert à arbitrer l'envoi de trames (quand on appuie sur un bouton pour commencer l'opération ou quand on a fini d'envoyer une trame pour envoyer la suivante) et un autre gère la génération des signaux pour communiquer avec le DAC. Ce dernier peut être dupliqué autant de fois que l'on veut de DAC.

En ce qui concerne les résultats, on peut voir ci-dessous sur la première figure, la génération d'une paire de 4 signaux correspondant à la communication avec deux DAC différents pour qu'ils produisent le même signal. On remarque que la synchronisation n'est pas un problème et est respectée.

Sur la deuxième figure, ci-dessous, on peut voir en sortie du DAC un signal triangulaire à 60 points par période.



D3/D4 : LDAC **D8/D9** : Horloge **D11/D12** : Trame de 24 bits **D13/14** : Slave Select



On peut donc remarquer que cette solution n'utilise pas tant d'outils logiciels, mais juste Quartus lui-même pour décrire le système en VHDL. De plus les contraintes de synchronisation sont respectées grâce au fait que plusieurs process peuvent s'exécuter en parallèle sans problème et il n'y a pas de problème de limitation fréquentielle comme auparavant avec le Nios II.

Une partie de cette solution n'a pas été réalisée : la gestion de la SDRAM pour stocker et lire les données inhérentes aux signaux. Voici les raisons :

-la gestion de ce type de mémoire est compliquée à mettre en œuvre (machines d'états complexes) et est donc trop chronophage par rapport au temps que j'ai eu pour faire mon PFE.

-Il est possible d'utiliser la logique du FPGA pour stocker quelques données pour réaliser des tests (ce que j'ai fait à l'aide de constantes contenant les suites de points décrivant un signal).

C. Communication entre le PC et la carte DE1 SoC

Cette partie consiste de :

-un programme en C sur le PC hôte qui envoie des données (relatives aux signaux) à la carte DE1 SoC, et qui est une partie de toute l'application sur le PC hôte permettant à l'utilisateur de choisir ses signaux. Ce programme isolé est disponible sur le Wiki dans la première semaine de travail (Com.txt)

-Un second programme en C mais qui tourne sur le processeur ARM de la carte DE1 SoC, permettant de réceptionner les données envoyées sur le PC et arrivant sur l'UART.

Pour le premier programme en C, l'API (Application Programming Interface) Windows offre la possibilité d'envoyer et de recevoir des données via un port COM, ce qui est exactement ce dont on a besoin ici. Pour l'utiliser, il suffit d'inclure la librairie <windows.h> fournie avec le compilateur GCC de MinGW.

Il faut tout d'abord s'assurer que la carte est bien détectée dans les ports COM du PC hôte, sinon le programme ne marchera évidemment pas. Dans ce dernier, on commence par demander le contrôle sur le port COM correspondant à la carte, en précisant la volonté de lire, écrire ou les deux.

Si cette opération réussit, il faut ensuite définir les paramètres de la communication, à savoir :

-le baud rate (vitesse de communication) : 115200.

-La taille des données dans une trame : un octet ici pour l'UART.

-Le nombre de bits de stop : 1.

-La présence d'un bit de parité et si oui quel type (paire ou impaire) : pas de bit de parité ici.

-Différents timeouts car les fonctions d'écriture / lecture sont bloquantes : à la volonté de l'utilisateur.

Une fois ces opérations complétées avec succès, on peut écrire, lire ou les deux le port COM correspondant à la carte DE1 SoC. En exploitant les valeurs de retour des fonctions, on peut savoir si ces instructions ont été effectuées avec succès ou non.

Dans mon cas, en plus de ces valeurs de retours de fonctions, j'ai pu voir la LED Tx de la carte DE1 SoC clignoter à chaque fois que j'envoyais des données.

Parlons maintenant du programme C du processeur ARM. Avant même de commencer à écrire une seule ligne de code, il a fallu, principalement, télécharger l'image de Linux bootable sur le site d'Altera, l'écrire sur la carte micro SD à l'aide de Win32 Disk Imager pour ensuite s'assurer que Linux bootait bien sur la carte DE1 SoC, via le terminal Putty.

Une fois que Linux était fonctionnel, j'ai commencé par écrire le programme le plus simple qui soit : Hello World. Pour compiler un programme C et le transformer en fichier exécutable par l'ARM, il faut utiliser le compilateur fourni dans Altera Embedded Shell, qui est un dérivé de GCC dédié à ce processeur.

La tâche suivante était de transférer cet exécutable depuis le PC sur lequel il a été généré, dans un dossier décidé à l'avance de l'environnement Linux, tournant sur la carte DE1 SoC. Pour ce faire, c'est la commande « scp » de l'Embedded Shell qu'on utilise. Cette dernière permet de transférer des données via une liaison Ethernet à un endroit donné.

De fait, il faut que la carte DE1 SoC et le PC soient sur le même réseau. Dans mon cas, j'ai créé un pont réseau entre la carte réseau Ethernet de mon PC, sur laquelle était branchée la carte DE1 SoC, et la carte réseau Wifi de mon PC connectée au réseau local de ma Box. Après avoir affecté une adresse IP, appartenant à la plage d'adresses du réseau local de ma Box, à la carte DE1 SoC via le fichier « interfaces » de l'interface « eth0 », j'ai pu utiliser la commande « scp » et transférer des fichiers exécutables sur la carte DE1 SoC. Ainsi, j'ai pu m'assurer que toute cette mécanique de développement fonctionnait en voyant sur Putty, Hello World s'afficher.

Après, j'ai pu commencer à développer le véritable programme pour l'ARM. Une librairie fournie par Altera offrait un ensemble de fonctions permettant de se servir de l'UART connecté à l'ARM.

Hélas, je n'ai pas pu tester une seule application mettant en œuvre l'UART. En effet, la carte après quelques jours, refusait de booter sur Linux bien qu'aucun changement de configuration n'ait été opéré. Quelques échanges avec le support se sont révélés infructueux quant à la résolution de ce problème. Après discussion avec mes encadrants, nous avons émis l'hypothèse que le bootloader de l'ARM ait pu être écrasé / endommagé, ce qui causerait ce souci. La supposition que toute la carte soit grillée était fausse étant donné que le FPGA marchait toujours avec ce que j'avais développé en VHDL.

Cependant, ayant déjà perdu quelques jours à essayer de dépanner le problème et n'ayant pas trouvé de solution immédiate pour réimplanter un bootloader au sein de l'ARM avec mes encadrants, j'ai décidé de travailler sur une autre partie du PFE, à savoir l'interface utilisateur pour le programme qui permet de choisir les signaux. Cela sera décrit dans la partie suivante.

D. Programme de génération des signaux sur le PC hôte

Initialement, je ne devais réaliser que l'algorithme en C qui permet à un utilisateur de choisir un signal parmi 3 motifs de base en console. Au vu des problèmes rencontrés avec le processeur ARM, j'ai également commencé à développer une interface plus ergonomique pour l'utilisateur.

J'ai donc réalisé, avant de rencontrer des problèmes avec la carte, un programme en C, utilisant des librairies standards, qui offre à l'utilisateur le choix parmi trois signaux comme on l'a déjà évoqué maintes fois. L'algorithme est assez simple. Une fois que l'utilisateur a choisi le type de signal qu'il veut (motif et ses caractéristiques) et la précision (le nombre de points par période), on calcule les valeurs du signal sur une période selon la précision choisie et les amplitudes crêtes (constantes dans le programme car liées à l'alimentation du DAC). Ensuite, il suffit de convertir toutes ces valeurs en entiers compris entre 0 et $2^{20} - 1$ (vu que le DAC reçoit des mots de 20 bits). Le signal est donc entièrement défini.

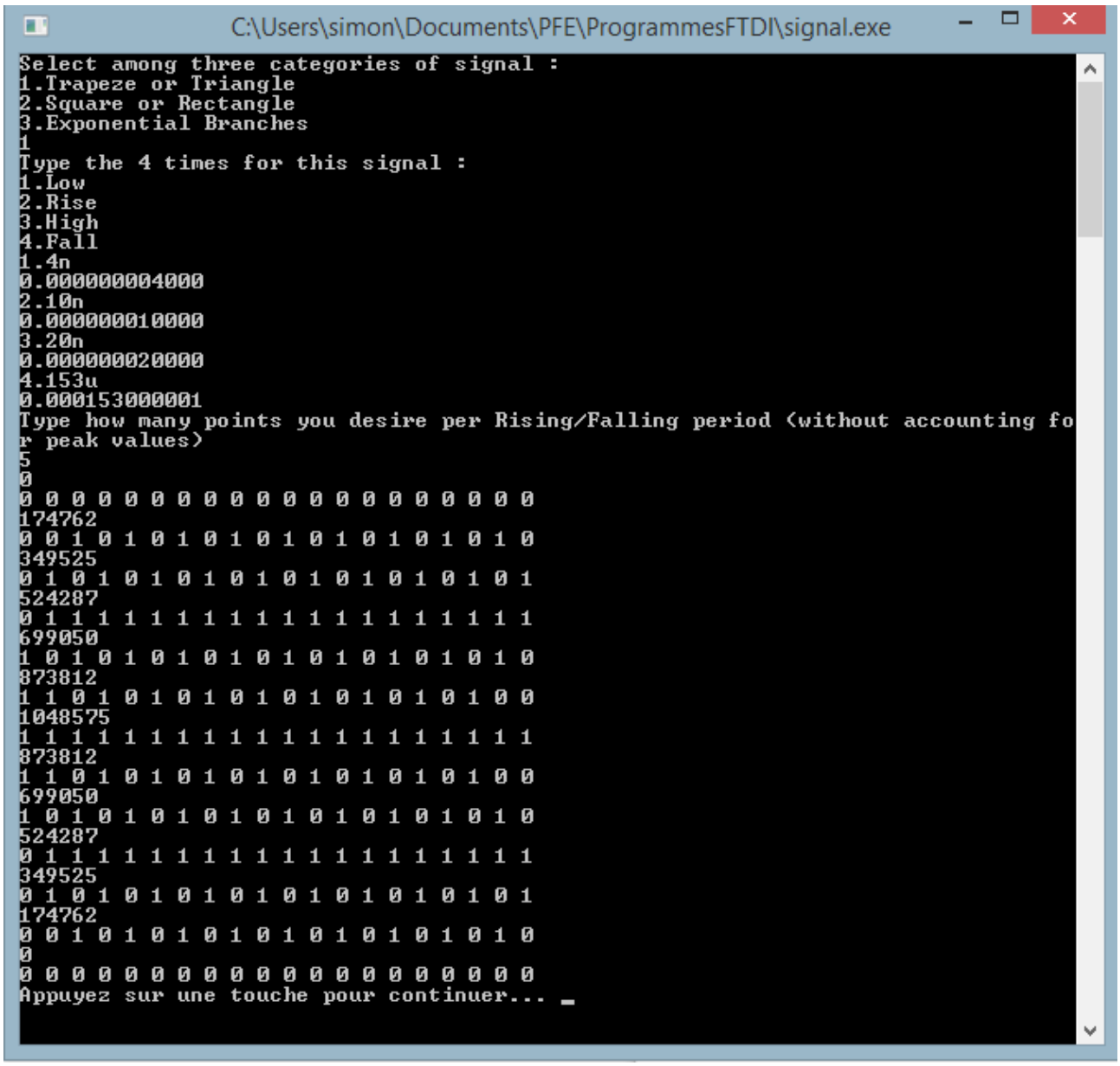
Voici un exemple d'exécution de ce programme ci-dessous avec un signal de type trapézoïdal / triangulaire :

-l'utilisateur sélectionne le motif.

-Il sélectionne les 4 temps correspondant aux temps bas, de montée, haut, de descente. Les préfixes peuvent être utilisés, la valeur en seconde est ensuite affichée.

-Il sélectionne le nombre de points qu'il veut pour les transitions montantes et descendantes.-Les points décrivant le signal sont ensuite affichés, respectivement en valeurs entières et en binaire.

Le programme peut être trouvé sur le wiki à la semaine de travail du 9 au 13 Février (signalConsole.txt).


C:\Users\simon\Documents\PFE\ProgrammesFTDI\signal.exe
Select among three categories of signal :
1.Trapeze or Triangle
2.Square or Rectangle
3.Exponential Branches
1
Type the 4 times for this signal :
1.Low
2.Rise
3.High
4.Fall
1.4n
0.000000004000
2.10n
0.000000010000
3.20n
0.000000020000
4.153u
0.000153000001
Type how many points you desire per Rising/Falling period (without accounting fo
r peak values)
5
0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
174762
0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
349525
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1
524287
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
699050
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
873812
1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0
1048575
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
873812
1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0
699050
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
524287
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
349525
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1
174762
0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Appuyez sur une touche pour continuer...

On va maintenant parler de la partie interface qui rend le programme plus ergonomique qu'une sobre console noire. Encore une fois, l'API Windows a été utilisé car il permet aussi de créer des programmes avec des fenêtres et différents contrôles (boutons, édition de texte, ascenseurs, menus déroulants etc.). On a donc couplé l'algorithme dont on a parlé précédemment avec ces fonctionnalités pour obtenir une interface plus accueillante afin que l'utilisateur choisisse ses signaux.

Expliquons maintenant les bases d'un programme pour créer une interface graphique avec l'API Windows. Elles sont suffisantes à la compréhension. Approfondir tous les détails d'un tel programme n'est ici pas dans l'optique de ce rapport car ce serait trop long. Voici comment les choses s'articulent :

-la fonction WinMain est équivalente au main d'un programme en C. On y définit la classe de la fenêtre principale qui régit son apparence (ce n'est pas une classe comme en langage objet mais une structure dont on remplit les champs). On y crée également la fenêtre principale en se basant sur la classe précédemment définie.

-Les contrôles sont créés comme la fenêtre principale à l'exception qu'ils sont des fenêtres filles dépendant de la fenêtre principale. Ils peuvent être créés dans le WinMain ou dans la procédure de fenêtre par exemple à la suite de la réception d'un message.

-La procédure de fenêtre (elle existe pour chaque fenêtre principale / mère) où sont gérés tous les messages que lui envoient les contrôles fils, l'utilisateur ou d'autres fenêtres. Ces messages servent à notifier la fenêtre de ce que veut l'expéditeur et sont donc traités pour agir en conséquence. Par exemple, un message WM_SIZE est reçu quand l'utilisateur redimensionne la fenêtre ce qui signifie qu'on voudra peut-être redimensionner les contrôles présents sur la fenêtre en conséquence; un message WM_COMMAND est reçu quand un bouton est actionné ce qui veut dire qu'il faudra peut-être effectuer des actions en conséquence.

-la boucle de message, incluse dans le WinMain, gère la file d'attente des messages destinés à la fenêtre principale, les traduit si nécessaire et les transmet à la procédure de fenêtre.

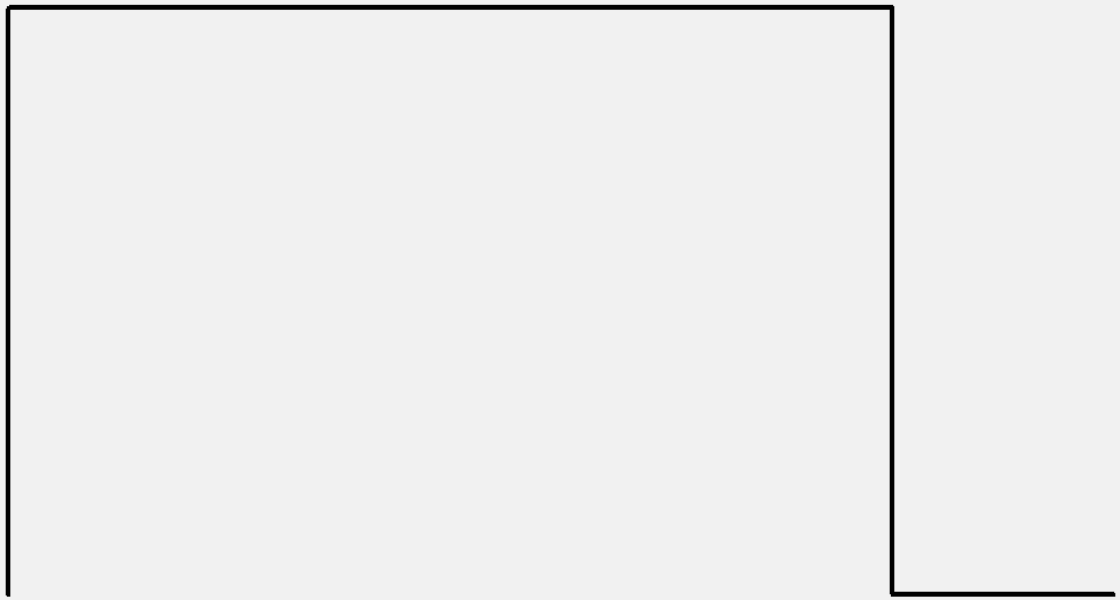
La figure ci-dessous illustre à quoi ressemble l'interface graphique que j'ai créée. Trois boutons en bas permettent de choisir parmi les 3 motifs de base. La place pour un quatrième est laissée, qui correspondra aux signaux composés de plusieurs motifs de base. En cliquant sur un des trois boutons, la forme d'onde s'affiche ainsi que les différents paramètres caractérisant le signal. Une fois choisis, on peut choisir de générer le signal, ce qui a pour effet de copier dans un fichier texte les informations décrivant le signal.

Dans l'exemple du signal carré ci-dessous, faire varier le slider du rapport cyclique (en haut à droite) actualise le signal sur la partie gauche pour voir quelle allure il a. On peut également saisir les amplitudes crêtes, à titre indicatif car elles dépendent de l'alimentation du DAC. Le dernier paramètre est la période du signal.

Le programme est disponible sur le Wiki à la semaine du 9 au 13 Février (Hmi.txt).

Signal Construction

Fichier Aide



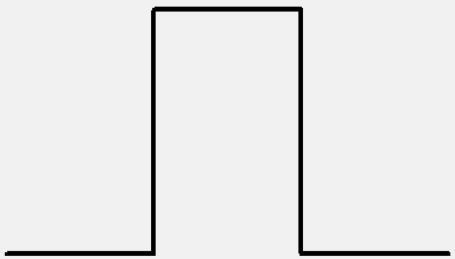
Duty Cycle

Amin (V) -5.5

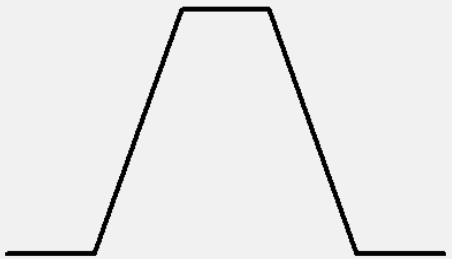
Amax (V) 5.5

Period 10.2u

Generate



Square/Rectangle

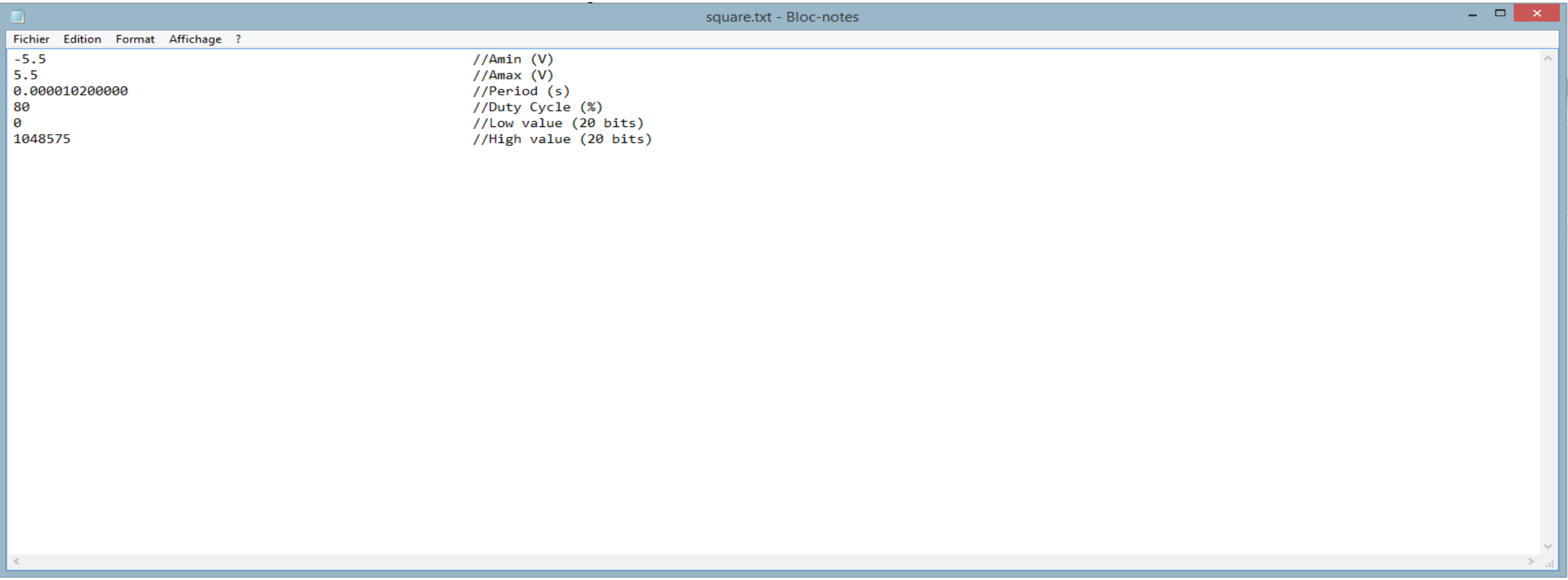


Trapeze/Triangle

Exponential

ENG FR 08:06 16/02/2015

Pour conclure cette partie, voici un exemple de fichier texte généré pour ce même signal carré et ses paramètres sur la figure ci-dessus. Les commentaires ont été rajoutés à titre indicatif pour comprendre quelles sont les valeurs.



6. Travail restant

Il n'était bien entendu pas possible en moins de deux mois et tout seul, de réaliser l'intégralité du projet.

Malgré les problèmes rencontrés avec le processeur ARM de la carte DE1 SoC, ce qui m'a empêché de réaliser toute la chaîne de développement initiale, j'ai quand même pu réaliser tous les autres maillons.

Ce qui se dessine à l'horizon quant au travail restant à présent est :

-finir l'interface graphique et permettre à l'utilisateur de construire des signaux à partir de motifs de base sur l'application du PC hôte. A plus long terme, pouvoir offrir la possibilité à l'utilisateur de choisir la synchronisation entre plusieurs signaux.

-Dépanner la carte DE1 SoC ou en racheter une nouvelle si ce n'est pas possible pour pouvoir écrire un programme pour le processeur ARM qui réceptionne les données envoyées par le PC.

-Écrire en VHDL la gestion de la mémoire SDRAM connectée au FPGA pour stocker les données venant de l'ARM et lire ces mêmes données. A plus long terme, prendre en compte l'aspect des synchronisations potentielles entre les signaux pour la communication avec les DAC.

-Établir un formalisme des données des signaux générés par le PC hôte pour que leur interprétation soit possible sur la carte DE1 SoC.

7. Conclusion

Je vais maintenant dresser un bilan de cette expérience qu'a été mon PFE en exprimant mon ressenti par rapport à celle-ci et ce que ça m'a apporté.

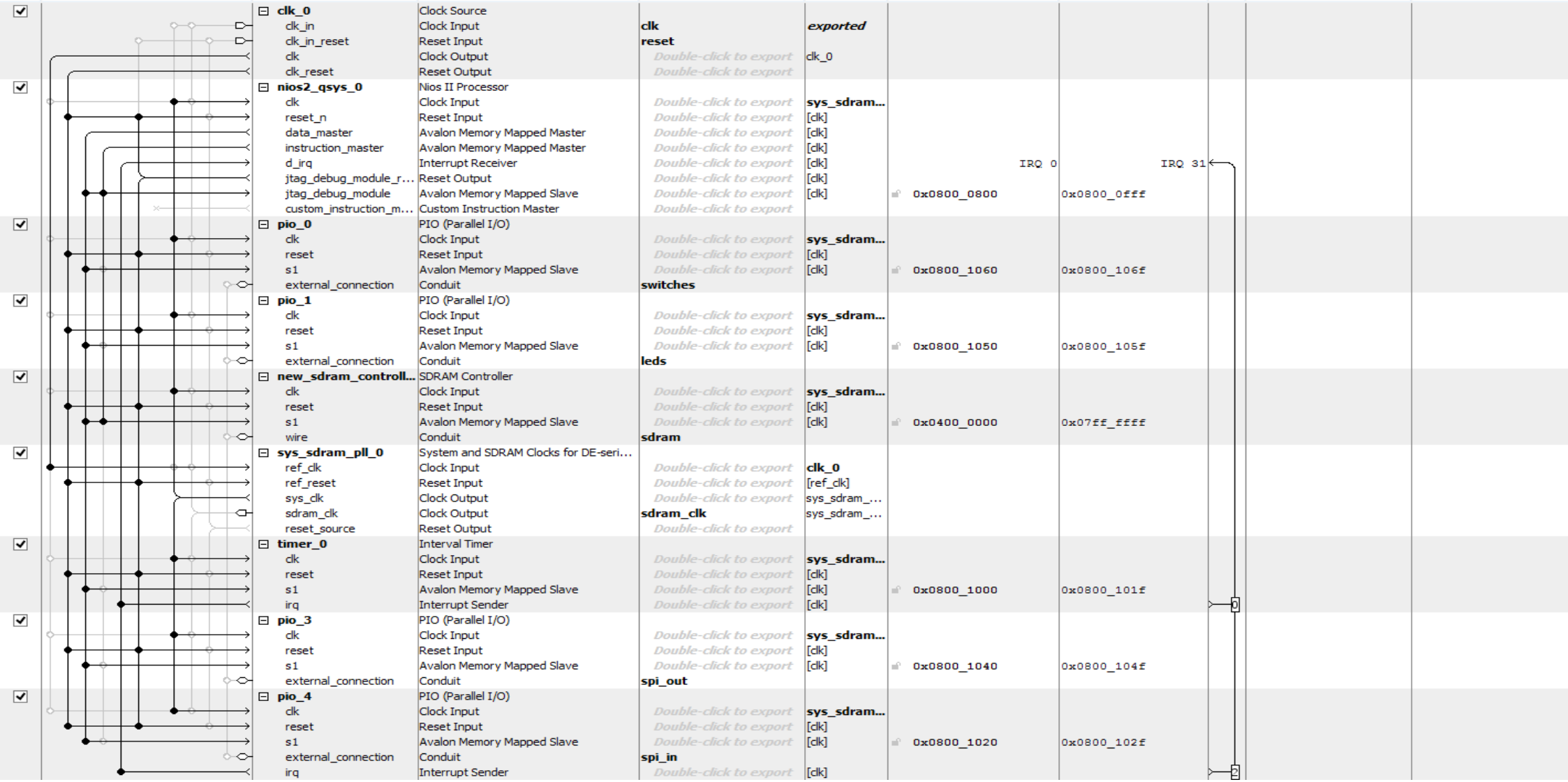
J'ai dû travailler seul sur un projet qui n'avait pas été lancé auparavant. Mes encadrants m'ont fourni du matériel sur lequel ils n'avaient jamais travaillé et que j'ai dû prendre en main par moi-même. Ces derniers points ont été les difficultés majeures de mon PFE. J'ai réussi à achever quelques réalisations intéressantes malgré que je n'aie pas pu boucler toute la chaîne de développement à cause des problèmes rencontrés sur la carte DE1 SoC. Cela va permettre aux personnes qui reprennent le projet après moi d'avoir une bonne base de travail sur laquelle s'appuyer.

Ayant dû travailler en totale autonomie sur un matériel inconnu, j'ai vraiment pu améliorer mes connaissances théoriques acquises en IMA, en engranger de nouvelles et développer une organisation efficace dans mon travail ainsi qu'une solide persévérance.

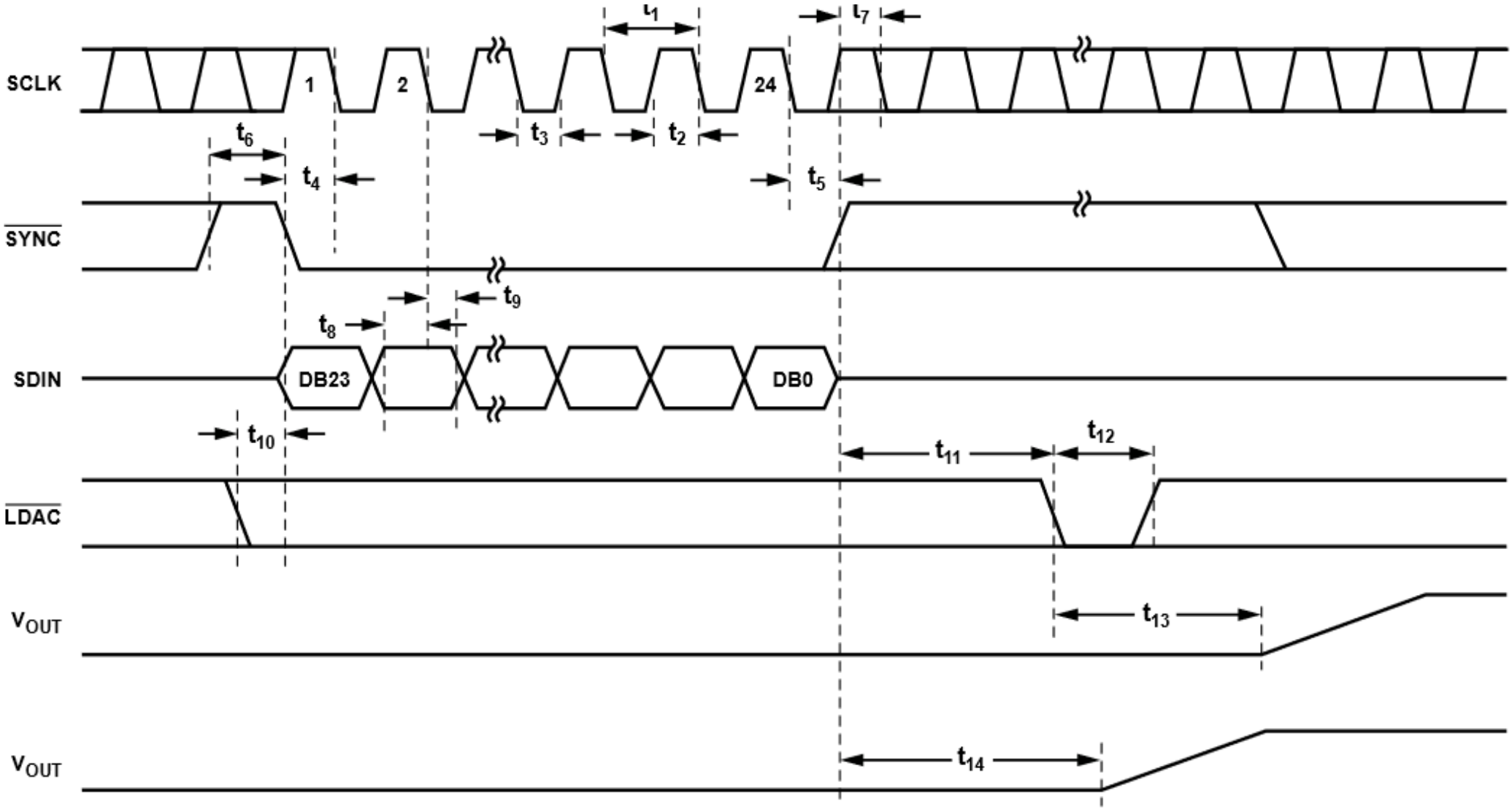
Je pense donc que globalement cette expérience aura été à la fois formatrice et enrichissante, juste avant la clôture de mes études par un stage long. Elle constitue un bon complément de formation pour ce dernier. En dépit de quelques points négatifs liés aux problèmes matériels, souvent indépendants de notre volonté, j'ai vraiment apprécié plancher sur ce sujet de PFE, espère que la personne qui va le continuer le mènera à bien, et souhaite une dernière fois réitérer mes remerciements à toutes les personnes qui ont participé de près ou de loin à cette expérience.

8. Annexes

A. Annexe 1



B. Annexe 2


SCLK
SYNC
SDIN
LDAC
VOUT
VOUT
1
2
24
DB23
DB0
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13
t14