ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

Laboratoire Européen pour la Physique des Particules
European Laboratory for Particle Physics

GENÈVE, SUISSE
GENEVA, SWITZERLAND

Group code: BE-BI-BL
EDMS No.: 1416786
Date: 10 August 2014

Author: Kamil Florek, HEIG-VD, REDS
Supervision: Michel Starkier, REDS
CERN Contact Person: Jonathan Emery

Ethernet High Speed with a SoC based FPGA for the Wire Scanner

(SoPC dans une FPGA avec Ethernet haute vitesse)

Abstract

Le CERN, organisation européenne pour la recherche en physique des particules, est l'un des plus grands et des plus prestigieux laboratoires scientifiques du monde dans ce domaine. Un grand nombre d'accélérateurs et détecteurs de particules sont développés et exploités pour la recherche scientifique.

Le sujet proposé a pour but de développer pour le groupe d'instrumentation un System-on-Programmable-Chip (SoPC) à base de FPGA Altera en utilisant une carte de développement commerciale. Un processeur (Nios II ou ARM) sera exploité dans la FPGA pour implémenter un port Ethernet haute vitesse, si possible en utilisant une interface avec module SFP cuivre ou fibre optique. Une investigation pourra être conduite sur le type de system d'exploitation minimum nécessaire pour l'implémentation d'un serveur connecté à la partie hardware de la FPGA générant les données à transmettre (la génération des données ne fait pas partie du projet). Une interface minimum sur PC permettant un contrôle de registres et transfert de donnée via le réseau Ethernet pourra également être réalisée (C, C++, Java, Qt, ..) pour valider la performance de l'interface.

Scontrol : SoPC dans une FPGA avec Ethernet haute vitesse

Bachelor of Science HES-SO

Génie Electrique, Electronique embarquée et mécatronique

Auteur :	Kamil Florek
Professeur responsable :	Michel Starkier
Expert :	Jonathan Emery
Proposé par :	CERN
Diplômé :	2014

1 Resumé

Le CERN, organisation européenne pour la recherche en physique des particules, est l'un des plus grands et des plus prestigieux laboratoires scientifiques du monde dans ce domaine. Un grand nombre d'accélérateurs et détecteurs de particules sont développés et exploités pour la recherche scientifique.

Le sujet proposé a pour but de développer pour le groupe d'instrumentation un System-on-Programmable-Chip (SoPC) à base de FPGA Altera en utilisant une carte de développement commerciale. Un processeur (Nios II ou ARM) sera exploité dans la FPGA pour implémenter un port Ethernet haute vitesse, si possible en utilisant une interface avec module SFP cuivre ou fibre optique. Une investigation pourra être conduite sur le type de system d'exploitation minimum nécessaire pour l'implémentation d'un serveur connecté à la partie hardware de la FPGA générant les données à transmettre (la génération des données ne fait pas partie du projet). Une interface minimum sur PC permettant un contrôle de registres et transfert de donnée via le réseau Ethernet pourra également être réalisée (C, C++, Java, Qt, ..) pour valider la performance de l'interface

2 Cahier des charges

Dans le cadre de ce projet, le diplômant doit effectuer les tâches suivantes :

- Mise en œuvre d'un processeur ARM Cortex-A9 intégré dans un SoC Altera Cyclone V
  - Apprentissage de l'utilisation des outils de développement et de debug
  - Implémentation d'un exemple simple de SoC dans une carte d'évaluation Altera
- Recherche et évaluation d'un OS, RTOS ou librairies, exécutés dans l'ARM et supportant des communications TCP/IP efficaces
- Développement d'un démonstrateur pour le CERN, effectuant des transmissions de paquets avec un débit minimal de 400 Mbits/s
  - Paquets de 257 Mbits envoyé en 0.7s (périodicité : 1 s)
  - Paquets de 500 bits, (périodicité : 1 ms).
  - IPv4 avec évolution possible vers IPv6
- Réalisation sous Windows d'un logiciel pour le contrôle distant de la carte avec affichage des données reçues

Table des matières

3 Introduction

3.1 CERN

L'organisation européenne pour la recherche nucléaire est composé d'une multitude de circuits sous-terrains qui servent au transport des faisceaux de particules. Pour mener leurs expériences à bien, le CERN a besoin d'un équipement de pointe afin de veiller au bon fonctionnement de l'ensemble du site. Qu'il s'agisse des détecteurs ou du "simple" guidage de faisceau l'ensemble de la chaîne doit fonctionner de manière fiable, sous peine d'infliger des dégâts à la structure.

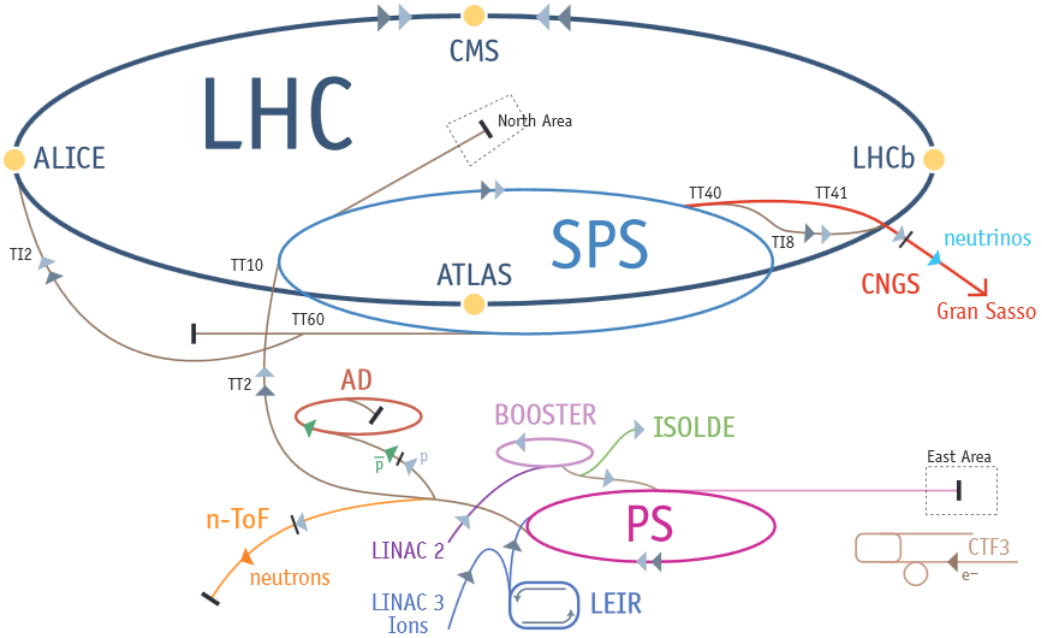

Figure 1 – Schéma des tunnels sous-terrains du CERN. Les différents injecteurs, accélérateurs et détecteurs sont visibles. [1]

Les faisceaux des particules sont accélérés a des vitesses proches de la lumière, de ce fait l'énergie emmagasinée dans ces derniers peut attendre des niveaux extrêmement élevés. Par exemple l'énergie stockée dans l'anneau du LHC peut attendre 350 $[MJ]$ ce qui correspond à l'énergie d'un TGV lancé à 150 $[km/h]$. Avec une telle quantité d'énergie il est possible de faire fondre 500 $[kg]$ de cuivre. A la vue de ces chiffres, il est évident qu'il faut à tout prix d'éviter que le faisceau frappe le bord de l'accélérateur.

3.2 Contrôle du faisceau

Pour éviter une telle situation, plusieurs systèmes de contrôles sont employés tout au long des tunnels. Un de ces systèmes est le Beam Wire Scanner (BWS). Il sert à mesurer la répartition du faisceau de particules dans le tunnel, et du coup de déterminer sa position à l'endroit de la mesure. Si la position du faisceau dévie de sa trajectoire normale, le système de contrôle déroute le faisceau dans un des tunnels d'arrêts où des barrages absorbent l'énergie emmagasinée en toute sécurité.

3.3 Beam Wire Scanner

Le principe de fonctionnement de cet instrument consiste à faire traverser un fil de carbone à travers le faisceau de particules. Cette action engendre une génération de particules secondaires qui traversent les parois de l'accélérateur ou de l'injecteur. Ces particules sont détectés par des détecteurs

placés à l'extérieur des parois. En relevant l'intensité des ces particules en fonction de la position de l'impact, il est possible de connaître la direction du faisceau principal dans l'accélérateur.

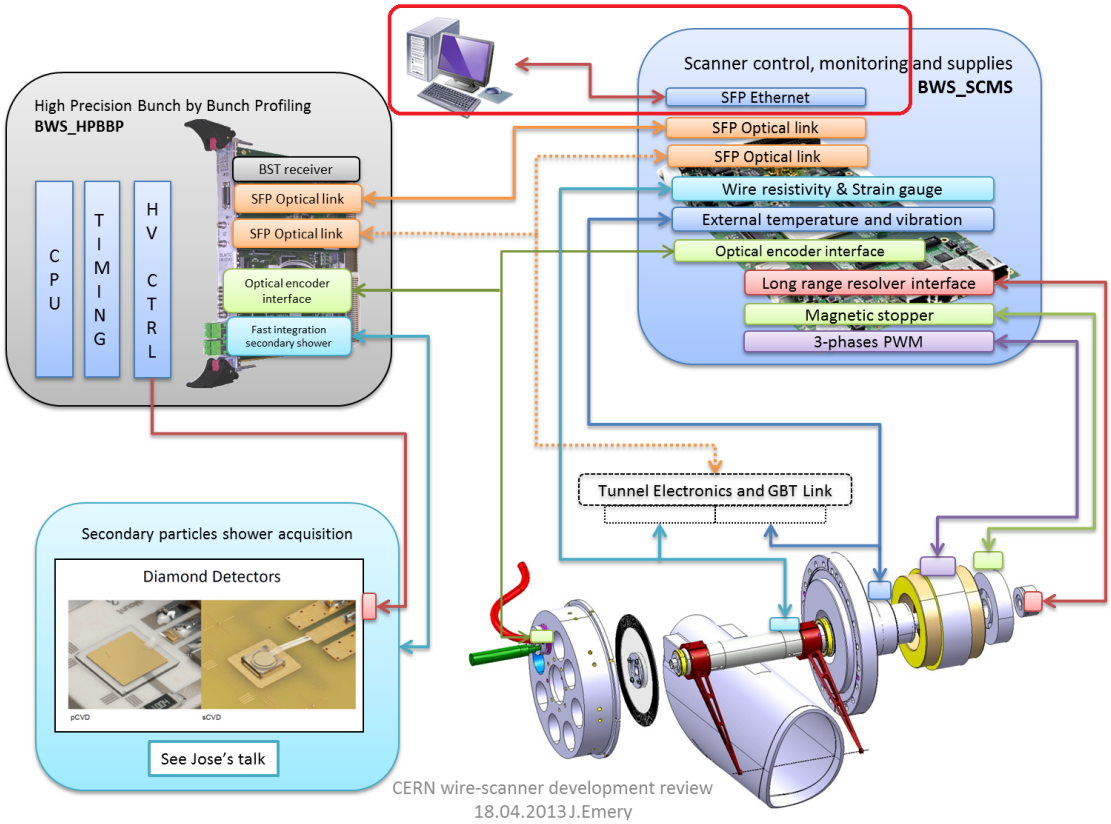

Figure 2 – Aperçu de l'ensemble du système BWS. La partie entourée en rouge correspond à la connexion Ethernet traité dans le cadre de ce dossier. [2]

Le système BWS est partagé en quatre éléments principaux. La carte électronique appelée Scanner Control Monitoring Supplies (SCMS) (En haut à droite de la figure 2), la carte électronique High Precision Bunch by Bunch Pofiling (HPBBP) (En haut à gauche de la figure 2), l'élément mécanique principal qui est le fil de carbone mobile (En bas à droite de la figure 2) et finalement le détecteur de particules secondaires (En bas à gauche de la figure 2).

3.3.1 SCMS

Cette carte électronique sert au contrôle direct de la partie mécanique du BWS (Moteur brushless triphasé). Outre le contrôle du mouvement du fil de carbone à travers le faisceau, cette carte gère aussi les tests automatiques. Elle est aussi chargée de veiller à ce que le système mécanique opère en toute sécurité (Magnetic Stopper). La carte SCMS relève une quantité importante d'informations durant le mouvement du fil (Vibration du fil, température, résistivité du fil, position angulaire) et une quantité moins importante lorsqu'aucun mouvement n'est effectué (Statut du système).

Ces données sont envoyés en premier lieu à la carte HPBBP via un lien optique, mais aussi, pour des raisons de maintenance et de développent, via une interface Ethernet à un ordinateur de contrôle (Cadre rouge sur la figure 2). Ce dernier est chargé de recevoir les données et les afficher, afin de simplifier la tâche aux techniciens travaillant sur le BWS. C'est ce lien Ethernet qui est concerné par cette thèse de bachelor. Il s'agit d'un lien Gigabit Ethernet avec une interface SFP. Idéalement les données transmises par ce lien sont une copie des données envoyées via l'interface optique à la carte HPBBP.

3.3.2 HPBBP

La deuxième carte électronique utilisée dans le système BWS est la carte HPBBP. Elle sert à effectuer l'acquisition des particules secondaires émises lorsque le scanner effectue une mesure ainsi que de relever la position du fil grâce au codeur optique. La deuxième fonction de cette carte est de stocker les données récupérés dans des bases de données ainsi que d'envoyer des données à la salle de contrôle. Cette carte permet aussi d'envoyer des signaux de trigger à la carte SCMS, ainsi que de configurer certains registres de cette dernière.

3.3.3 Partie mécanique

La partie mécanique du BWS forme le cœur de ce système de mesure. Comme dit précédemment, le principe de mesure consiste à faire déplacer un fil de carbone à travers le faisceau pour générer des particules secondaires. La partie mécanique de ce projet est extrêmement pointue. Pour commencer, la fourche avec le fil doit posséder une inertie très faible, afin de pouvoir accélérer suffisamment pour faire rentrer le faisceau à vitesse désirée dans le faisceau. De plus, une partie du montage se trouve dans le vide (vers le faisceau) et une autre partie se trouve dans le tunnel de maintenance, d'où l'utilisation d'un moteur brushless. Ainsi le stator et le rotor se trouvent dans deux espaces différents.

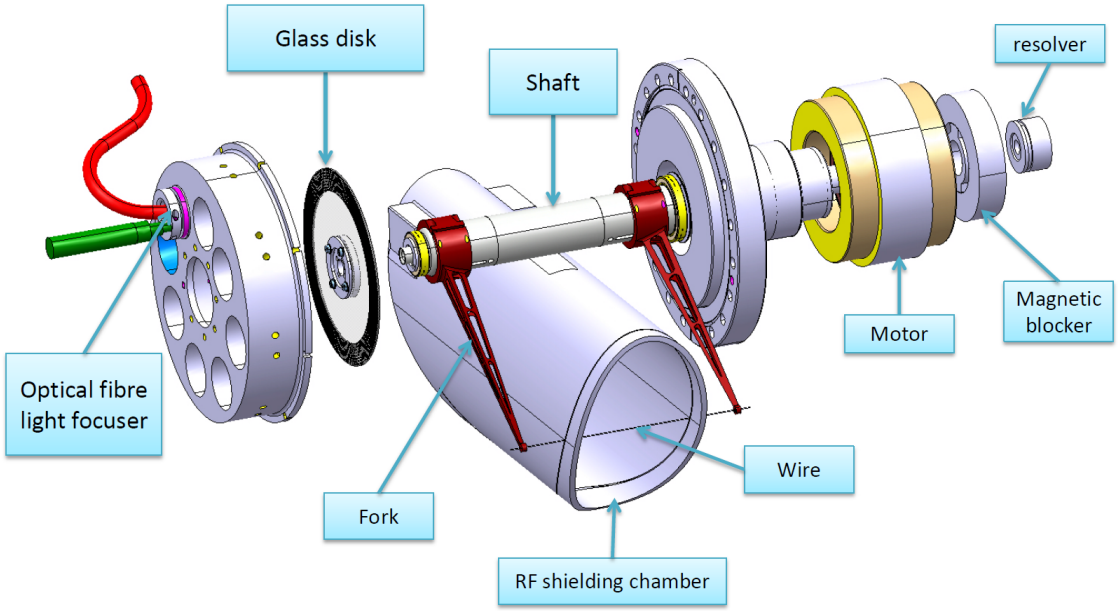

Figure 3 – Détail de la partie mécanique du Beam Wire Scanner. [2]

Le faisceau de particules voyage dans la chambre blindé. Le fil de carbone tendu peut traverser cette chambre pour effectuer une mesure. Il est tenu grâce à une fourche fixée sur un axe mobile. L'axe est mû par un moteur brushless, et sa position est contrôlée par un codeur optique ainsi que par un résolveur. Un bloquer magnétique est aussi présent pour bloquer l'axe en position de sécurité en dehors du faisceau. La figure 3 illustre la mécanique du système.

3.3.4 Détecteur de particules secondaires

Il s'agit d'un détecteur placé à l'extérieur du conduit de faisceau. Le particules générées durant le mouvement du fil dans le faisceau sont interceptés par ce détecteur. Les données collectées sont ensuite envoyés à la carte HPBBP.

3.4 Déroulement d'un processus de scan

Un processus de scan est un mouvement du fil à travers le faisceau. Ce processus est partagé en deux mouvement distincts. Pour commencer, la fourche est en position de repos au dessus du faisceau. Le moment venu, une commande de scan est envoyé depuis la carte HPBBP à la carte SCMS, ce qui a pour effet d'abaisser la fourche avec le fil à travers le faisceau. Ce mouvement est définit sous le nom de mouvement IN.

Après ça, un mouvement inverse est effectué, de bas en haut, pour faire revenir la fourche à sa position de départ. Ce mouvement est appelé le mouvement OUT.

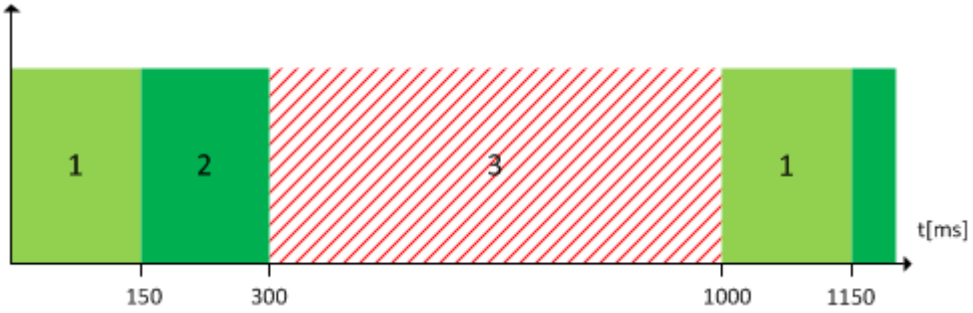

Figure 4 – Représentation graphique d'un processus de scan dans le pire cas de figure. La zone 1 correspond à un mouvement IN, la zone 2 à un mouvement OUT et la zone 3 à l'envoi des données. Le cycle se répète chaque seconde.

Chaque mouvement dure 150 $[ms]$, ce qui fait que le temps total du mouvent du fil est de 300 $[ms]$. Durant ce laps de temps, diverses données sont collectées, comme décrit lors de la description de la carte SCMS. Ces données sont stockées dans une mémoire de la carte SCMS, puis lorsque le mouvement est fini, elles doivent être envoyés à la carte HPBBP, ainsi qu'à l'ordinateur de contrôle via Ethernet.

Le temps alloué à l'envoi des ces données est déterminé par le délais auquel un prochain processus de scan sera engagé. Ce délais varie selon l'accélérateur, mais afin de garantir le bon fonctionnement du BWS, le pire cas de figure a été sélectionné. Il correspond à un cycle de répétition des scans toute les 1 $[s]$. Ce qui permet de déterminer le temps maximal qui peut être alloué à l'envoi afin de garantir le fonctionnement du BWS sur l'ensemble de la structure du CERN :

$$t_{env} = T - t_{scan} \rightarrow 1 - 0,3 = 0,7 \ [s] \tag{1}$$

La figure 4 résume sous forme graphique un processus de scan qui se répète avec la période minimale entre les scans (pire cas de figure).

3.5 Données envoyés

Comme stipulé, les données récoltés durant un processus de scan doivent être envoyés en un temps borné via Ethernet (paquet de données de scan). Cependant, il y a aussi un autre type de données qui est censé être envoyé à un taux beaucoup plus grand. Ces données renseignent sur le statut du système BWS et doivent être envoyés chaque milliseconde, peu importe si un processus de scan est engagé ou pas (paquet de données de statut). Le paquet de statut est bien plus petit que le paquet de scan, comme montré dans les sections suivantes.

3.5.1 Paquet de données du scan

Le paquet de données du scan est généré lors du mouvent du fil à travers le faisceau de particules. Ce paquet a la structure suivante :

Source du signal	**Total des données [bit]**
PMT current	$10,8 \cdot 10^6$
Photodiode	$168 \cdot 10^6$
Laser monitoring	$420 \cdot 10^3$
Measures UVI	$28,8 \cdot 10^6$
Strain gauges	$19,2 \cdot 10^6$
Strain gauges	$9,6 \cdot 10^6$
Temperature	$9,6 \cdot 10^3$
Resolver position	$16,8 \cdot 10^6$
I sens stopper	$420 \cdot 10^3$
I sens motor	$2,52 \cdot 10^6$
V sens motor	$1,68 \cdot 10^6$
External GPI	$16,8 \cdot 10^6$
Supply voltages	$288 \cdot 10^3$
Total Memory	$275 \cdot 10^6$

Tableau 1 – Détail du paquet généré lors d'un processus de scan. La taille totale du paquet est de 275 $[Mbits]$. [2]

Le calcul de la taille totale du paquet permet de connaître le débit nécessaire afin de transmettre la quantité des données souhaitée, dans ce cas 275 $[Mbits]$. Connaissant le temps maximal alloué pour transmettre cette quantité de données (0,7 $[s]$), le débit nécessaire est de :

$$Debit = \frac{TaillePaquet\ [bits]}{TempsMax\ [s]} \rightarrow \frac{275 \cdot 10^6}{0,7} \approx 393\ [Mbits/s] \tag{2}$$

Avec un tel débit utile (Sans compter les donnes de contrôle propre au protocole) ou supérieur il est possible de transmettre le paquet voulu dans le temps. Pour la suite du projet, pour avoir une certaine marge, un débit minimal de 400 $[Mbits/s]$ devrait être atteint.

3.5.2 Paquet de statut

Comme stipulé précédemment, le paquet de statut est envoyé chaque milliseconde. Ce dernier est bien plus petit que le paquet de scan. Il est composé de la façon suivante :

Source du signal	**Total des données [bit]**
PMT current	18
Photodiode	28
Laser monitoring	14
Measures UVI	48
Strain gauges	32
Strain gauges	16
Temperature	32
Resolver position	28
I sens stopper	14
I sens motor	84
V sens motor	56
External GPI	28
Supply voltages	96
Total Memory	494

Tableau 2 – Détail du paquet de statut. La taille totale de ce paquet est de 494 $[bits]$. [2]

Ce paquet fait uniquement 494 bits, ce qui correspond à 62 bytes de données. Le challenge concernant le transfert de ce paquet n'est pas sa taille, mais la période avec lequel ce dernier doit être transféré. De plus, il faut que le paquet de statut soit envoyé même lorsque le paquet de scan es transféré.

3.6 Résumé

En résumé, dans le cadre de ce travail de bachelor, deux paquets doivent être envoyés par Ethernet à l'ordinateur de contrôle depuis la carte SCMS (Cette dernière sera simulée) :

- Le "grand paquet" de scan (275 $[Mbits]$), doit être envoyé en moins de 0,7 $[s]$ chaque seconde.
- Le "petit paquet" de statut (494 $[bits]$), doit être envoyé chaque milliseconde, même lorsque le paquet de scan est en cours de transfert.

La débit de la connexion Ethernet nécessaire pour ce faire est de 400 $[Mbits/s]$. Ceci-dit, si ce débit ne peut pas être atteint dans la suite de ce projet, un débit moins important sera quand même acceptable pour cette première version du projet. Ainsi, même un débit de l'ordre de 200 $[Mbits/s]$ sera utile pour la démonstration des possibilités du projet.

4 Arria V SoC

La carte SCMS, depuis laquelle l'envoi Ethernet concerné par ce projet de bachelor se fait, est basée sur le composant Arria V ST SoC [1] de chez Altera. Ce composant est une FPGA couplé à HPS (Hard Processor System) ARM Cortex-A9. Ce processeur 32 bit dualcore est largement repondu dans des applications électroniques grand public qui nécessitent des grandes puissances de calcul (Smartphones, TV box, etc).

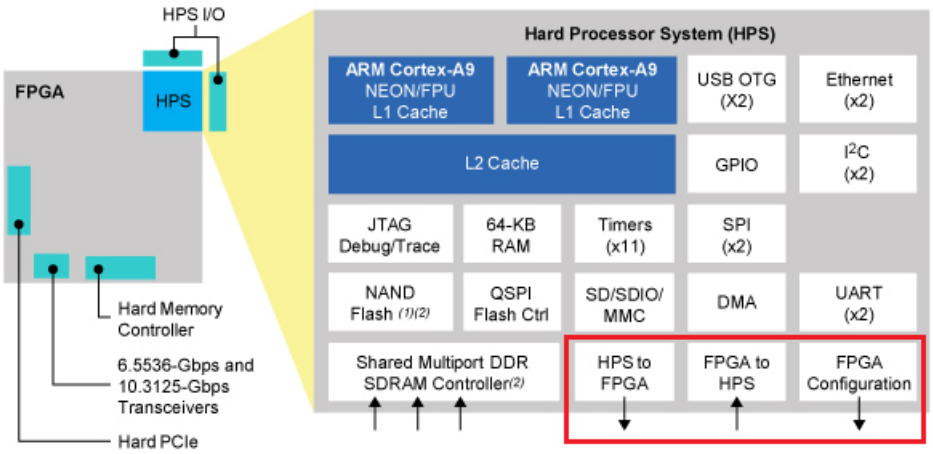

Figure 5 – Vue globale du Arria V SoC. Dans la partie FPGA les hard IP sont visibles, dans la partie HPS, les cœurs et les périphériques. Les bridges HPS-FPGA et FPGA-HPS sont entourés en rouge. [3]

La décision d'utiliser ce composant au sein du CERN a été prise après plusieurs tentatives d'implémentation des connexion Ethernet efficaces. Auparavant, des tentatives de génération de flux directement dans la partie FPGA ou d'utilisation d'un processeur softcore (Nios II) se sont avérés être des échecs. De ce fait, le choix s'est porté sur un composant intégrant un processeur plus performant.

De plus, l'usage d'un SoC est motivé par le fait que l'acquisition des nombreux signaux doit être faite, et dans ce domaine un composant programmable offre des avantages nets au niveau de la flexibilité ainsi que de la vitesse d'implémentation. En utilisant un SoC, des tâches simples mais répétitives peuvent être effectuées par la partie FPGA du composant, et des tâches de plus haut niveau par le HPS, libérant ainsi les ressources du processeur.

4.1 Cohabitation HPS-FPGA

Le processeur et la partie FPGA peuvent travailler de manière séparée. Ceci-dit, il est évidemment possible de les faire communiquer pour créer un système complet FPGA/HPS.

Pour ce faire, un système de bridges HPS-FPGA et FPGA-HPS au niveau du processeur est utilisé (Blocs entourés de la figure 5). Ce système permet d'accéder aux blocs IP dans la FPGA depuis le HPS, et d'accéder aux périphériques du HPS depuis un bloc IP de la FPGA.

Outre les bridges HPS to FPGA et FPGA to HPS, un autre existe. Il sert à la configuration de la partie FPGA du composant depuis le HPS. Cependant, dans le cadre de ce travail de bachelor, ce dernier sera quasiment pas utilisé tel quel.

4.1.1 Bus AMBA

Afin de bien comprendre le fonctionnement de ces bridges, il est nécessaire de rentrer plus en détail dans leur fonctionnement. Les bridges en question utilisent la norme de bus AMBA (Advanced

1. Dans le cahier de charges il est question du composant Cyclone V SoC, mais finalement la décision a été prise au CERN d'utiliser Arria V SoC.

Microcontroller Bus Architecture). Cette norme définit en réalité une famille de bus, en standard ouvert, qui est développée et améliorée par ARM depuis 1996. Elle est largement exploité dans les SoC ainsi que sur divers processeurs.

AMBA offre plusieurs types de bus, suivant l'application voulue et le type de périphériques à interconnecter. Les bridges HPS-FPGA et FPGA-HPS présents sur l'Arria V SoC utilisent uniquement le bus AXI (Advanced eXtensible Interface). C'est le dernier né de AMBA, et naturellement le plus performant. Lui même est partagé en deux sous-bus :

- AXI Stream - Utilisé là ou une bande passante importante est nécessaire
- AXI Lite - Utilisé lorsque une latence minimale est exigée, par exemple pour le contrôle des registres.

Malheureusement, dans la littérature d'Altera, ces deux sous-bus ne portent pas ces noms. Le bus AXI Stream est désigné simplement par AXI et AXI Lite est désigné par Lightweight AXI.

Afin de rester cohérent avec la documentation Altera, leurs désignations seront reprises dans la suite de ce dossier.

4.1.2 Bridges

Ayant une connaissance plus approfondie du bus AXI, il est désormais possible de voir le détail des bridges d'interconnexion entre le processeur et la partie FPGA (Désignée sous FPGA fabric dans la documentation d'Altera).

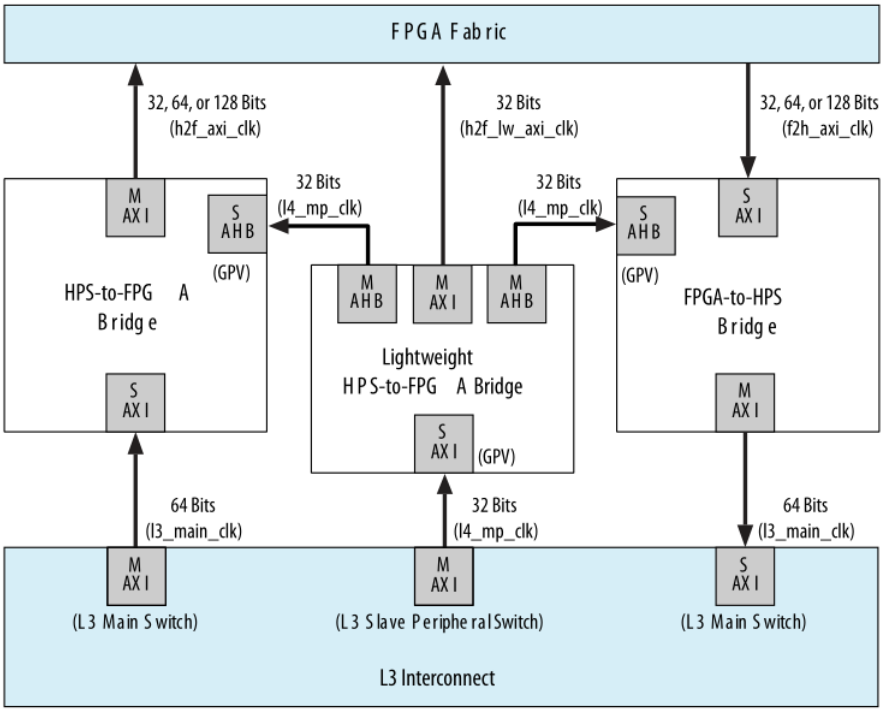

Figure 6 – Détail des bridges entre le HPS et la partie FPGA du composant Arria V SoC. Le sens du transfert de données se fait toujours depuis le maître (M) vers l'esclave (S). Les noms entre parenthèses indiquent le domaine du clock utilisé pour le bus. [4]

La figure 6 montre en détail le fonctionnement des bridges. Il y a trois blocs distincts, le premier qui utilise AXI pour envoyer des données du processeur à la partie FPGA, le deuxième qui permet de passer par Lightweight AXI pour envoyer des données à la partie FPGA et finalement le dernier qui permet d'envoyer des donnes depuis la partie FPGA aux périphériques du processeur par AXI.

Les deux bus AXI ont une largeur de données configurable. Elle peut être choisie entre 32, 64 et 128 bits. Par contre Lightweight AXI impose une largeur fixe de 32 bit de données.

Avant de pouvoir utiliser les bridges, il est nécessaire de les configurer via leurs registres GPV (Global Programmers View). Ces registres servent à contrôler les bridges et à déterminer leur comportement. Pour ce faire, il faut impérativement passer par Lightweight AXI. C'est le seul bus qui a accès au GPV des tous les bridges. Cette étape de configuration est effectuée par le programme implémenté par l'utilisateur dans le HPS.

Toujours sur la figure 6, il est montré que l'accès aux GPV des bridges AXI depuis le bridge Lightweight AXI n'est pas fait avec la norme AXI, mais AHB (Advanced High-performance Bus). C'est une révision plus ancienne et moins performante qu'AXI, mais qui reste employée pour l'accès à certains registres ne nécessitant pas de vitesses ou des débits importants. Ceci ne change rien pour le développeur, qui utilisera au final uniquement les deux variantes de la norme AXI.

4.1.3 Memory map

Afin d'accéder à un quelconque périphérique depuis le HPS, un système de memory map est utilisé (Figure 7). De cette façon, chaque périphérique possède une adresse de base ainsi qu'un espace dans la mémoire qui lui est propre. Il suffit que le programme de l'utilisateur écrive ou lise à l'adresse du périphérique souhaité pour y accéder.

A part les périphériques hardware du HPS (Addresse de 0xFF400000 à 0xFFFD0000), l'utilisateur peut accéder à ses blocs IP instanciés dans la partie FPGA de la même manière.

Deux espaces mémoires distincts sont alloués pour ce faire. Le premier, qui permet d'utiliser le bridge AXI, commence à l'adresse 0xC0000000 et finit à 0xFC000000 (Entouré en vert sur la figure 7). Ainsi, 960 $[MB]$ peuvent être adressés via AXI.

Le second, qui commence à 0xFF200000 et finit à 0xFF400000 (Entouré en rouge sur la figure 7), permet d'adresser 2 $[MB]$ d'IPs via le bridge Lightweight AXI.

Utiliser ces espaces mémoires est l'unique moyen pour communiquer avec la partie FPGA depuis le HPS.

Plusieurs autres espaces existent dans la mémoire, à commencer par les espaces STM (System Trace Macrocell) et DAP (Debug Access Port). Ces espaces servent essentiellement aux périphériques servant durant la phase du debug.

Ensuite l'espace SCU (Snoop Control Unit) et L2 Registers qui permet de configurer le fonctionnement des cœurs du processeur.

Le processeur possède bien évidemment de la mémoire SDRAM mappée. Celle-ci fait à peu de choses près 3 $[GB]$.

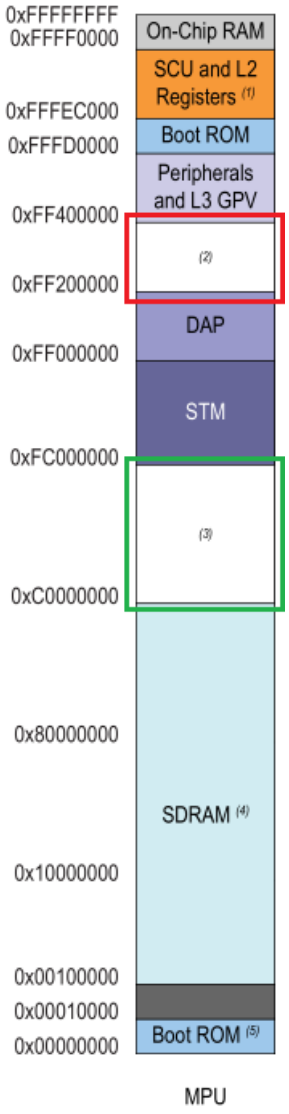

Figure 7 – Extrait de l'address map valable pour le HPS. Le cadre rouge montre l'espace de Lightweight AXI et le cadre vert celui de AXI. [5]

Finalement, entre l'adresse 0 et 0x10000, un espace de boot est alloué. Par défaut, dans cette zone est mappée la Boot ROM qui se trouve physiquement à l'adresse 0xFFFD0000. Elle contient un programme qui configure le HPS de manière à ce que celui-ci fonctionne correctement. Pour être plus précis, dedans se trouvent deux programmes, un exécuté dans le cas d'un cold reset (Démarrage physique du HPS), et l'autre exécuté dans le cas d'un warm reset (Reboot software).

Cependant, il est possible de mapper la mémoire On-Chip RAM (Se trouvant physiquement à l'adresse 0xFFFF0000) à cet espace pour booter à partie de celle-ci. Cette mémoire est accessible par un maître placée dans la partie FPGA, ce qui permet de changer le programme de boot du HPS par la FPGA.

Dans le cadre de ce projet le boot se fera uniquement depuis la Boot ROM.

4.2 Résumé

En résumé, pour accéder à la partie FPGA de l'Arria V SoC, il est nécessaire de passer par des bridges AXI. Pour ce faire, il faut écrire ou lire dans un des espaces mémoire lié aux bridges. Les espaces commencent aux adresses :

- 0xC0000000 - Pour le bridge AXI.
- 0xFF200000 - Pour le bridge Lightweight AXI.

Le bus AXI offre une bande passante importante, et le bus Lightweight AXI offre une latence minimale. Le bridge Lightweight AXI a une taille fixe de 32 bits, alors que AXI peut avoir une taille de 32, 64, ou 128 bits.

5 Carte de développement

Pour simuler la carte SCMS, une carte de développement choisie par le CERN est utilisée. Il s'agit d'une carte Altera Arria V SoC FPGA Developpement Kit qui permet d'exploiter les principaux éléments offerts par le composant Arria V SoC.

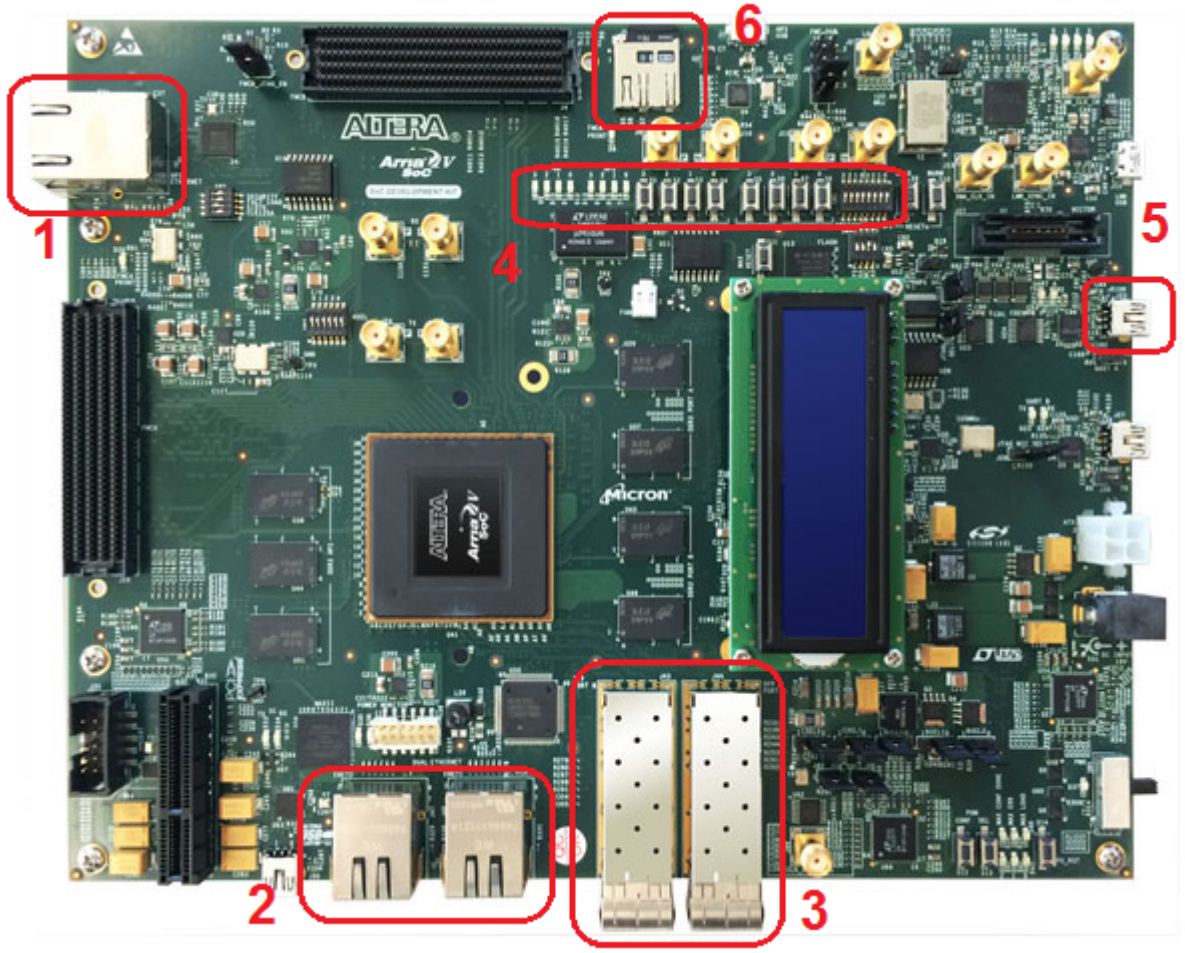

Figure 8 – Vue de la carte de développement Arria V SoC. Les principaux éléments utilisés dans le cadre de ce projet sont mis en évidence. [6]

La figure 8 montre la carte de développement ainsi que les éléments utiles directement ou indirectement dans le cadre de ce projet. Ces éléments sont :

1. Le port Gigabit Ethernet, il est directement connecté à l'interface Ethernet du HPS passant par un PHY externe. C'est ce port qui sera utilisé dans la suite du projet.
2. Deux ports Ethernet 10/100. Ils sont directement connectés à la partie FPGA de l'Arria V SoC. Ils ne sont pas utilisés dans le cadre de ce travail de bachelor.
3. Deux ports SFP+[2]. Ils ne sont pas utilisés dans ce travail de bachelor.
4. Divers éléments permettant d'interagir avec la partie HPS et FPGA de l'Arria V SoC (Quatre LEDs, BP et Switchs pour la partie HPS et autant pour la partie FPGA)
5. Connecteur mini-USB pour la sortie série de la partie HPS.
6. Emplacement pour la carte SD, offrant une source de boot pour le HPS.

2. Malgré que le cahier de charges stipule l'utilisation de l'interface SFP, elle ne sera pas utilisé, car elle n'est pas directement accessible depuis le HPS. Ce qui ne respecte pas ligne conductrice de ce travail de bachelor, qui est d'utiliser le HPS pour communiquer par Ethernet.

La figure 9 montre le schéma bloc de la carte de développement. Les blocs mis en évidence sont ceux qui sont employés dans la suite de ce dossier.

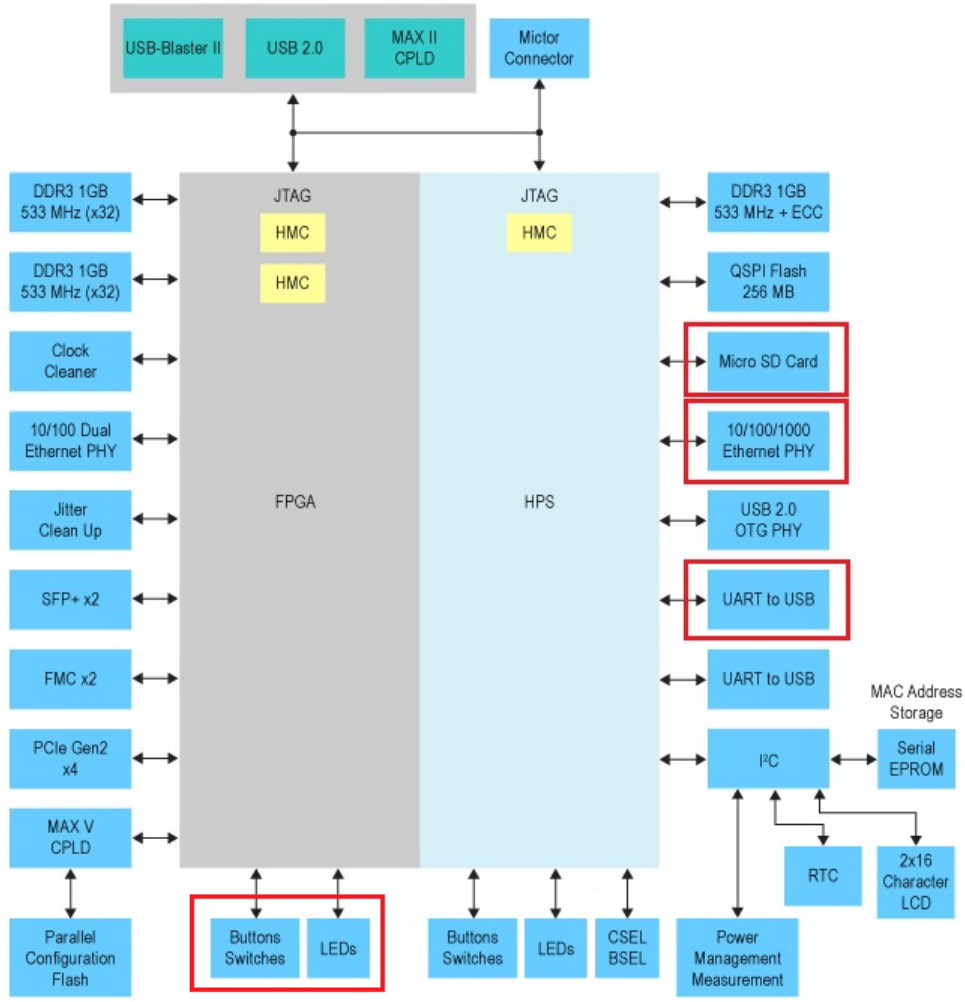

Figure 9 – Schéma bloc de la carte de développement. Les parties mises en évidence sont celles directement utilisées dans le cadre de ce travail de bachelor. [6]

5.1 Outils de développement

Pour travailler sur cette carte, il est nécessaire d'installer la suite d'outils permettant le développement aussi bien hardware que software. Cette suite d'outils est composée de :

- Quartus II v13.1 - L'environnement Altera pour le développement sur les composants programmables.
- Qsys - En réalité ce programme fait partie de Quartus II. Il permet de gérer l'interconnexion des sous systèmes dans la partie FPGA de manière graphique.
- Embedded Design Suite/ARM DS-5 Studio - Permet le développement software pour le HPS.
- Kit Installation - Suite de projets d'exemple et de documentation pour cette carte de développement.

Tous ces programmes sont à disposition sur le site d'Altera. Pour les utiliser une licence Altera et une licence ARM est nécessaire.

Pour commencer à travailler sur un nouveau projet, il faut partir d'un projet de référence appelée GSRD (Golden System Reference Design). Ce projet est préconfiguré pour être compatible sur cette carte de développement. Dedans se trouvent aussi bien le design hardware que software.

6 Système bare-metal

La première partie pratique de ce travail de bachelor consiste à faire fonctionner le processeur sans système d'exploitation (càd bare-metal). Programmer de cette manière à l'avantage de pouvoir optimiser le plus possible le code, mais nécessite de connaitre parfaitement le fonctionnement bas niveau du matériel, entre autres le fonctionnement des bridges AXI.

6.1 Hardware library

Lorsque l'utilisateur souhaite travailler sans OS, ou avoir accès au matériel bas niveau avec un OS, il est forcé de jongler avec les bits des différents registres. Ceci n'est pas une tâche facile, pour cette raison Altera a décidé de fournir une suite de libraires C ajoutant une couche d'abstraction pour l'utilisateur.

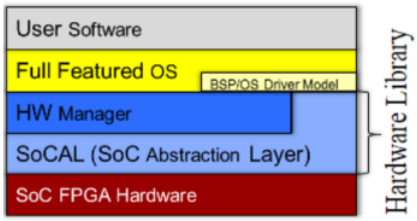

Figure 10 – La suite des libraires hardware se place entre le matériel et l'OS/le programme utilisateur. [7]

Cette suite est composée en réalité de deux APIs. La SoCAL (SoC Abstraction Layer) qui est la plus proche du matériel, et Hardware Manager API, qui se situe plus haut. La documentation complète de ces librairies se trouve dans les répertoires de l'installation de l'Embedded Design Suite :

```
[EDS install path]/embedded/ip/altera/hps/altera_hps/doc/socal/html/index.html
[EDS install path]/embedded/ip/altera/hps/altera_hps/doc/hwmgr/html/index.html
```

Ces librairies sont largement exploités dans le cadre de la programmation bare-metal, mais elles sont aussi utilisés avec un OS, comme il sera montré plus loin dans ce dossier.

6.2 Application bare-metal

> ⚠ **Attention !**
> Les manipulations décrites dans ce chapitre ont étés effectuées sur une carte de développement Cyclone V SoC. Le principe reste le même pour les deux cartes de développement, mais des erreurs peuvent surgir si le design est repris tel quel pour la carte Arria V SoC.

Connaissant le fonctionnement des bridges AXI et l'existence de l'Hardware library, un simple programme de démonstration peut être développé. Il aura pour but de :

1. Prendre le contrôle de la partie FPGA.
2. Ouvrir les bridges AXI et Lightweight AXI.
3. Lire le ID système par AXI.
4. Faire clignoter les LEDs 50 fois, en les allumant par AXI et les éteignant par Lightweight AXI.
5. Envoyer et lire une valeur inversée par le bloc inverseur avec Lightweight AXI.
6. Fermer les bridges.

Comme tout design sur un SoC, l'application est partagée en deux parties, une hardware avec divers blocs instanciés dans la FPGA, et une partie software pour le HPS.

6.2.1 Design hardware

Cette parte de l'application décris la partie hardware, c'est à dire l'ensemble de blocs instanciés dans la partie FPGA du composant. Les blocs ont été essentiellement instanciés avec Qsys, car il est bien plus facile et rapide d'instancier les composants de cette manière plutôt que de le faire dans le code VHDL (Dans Quartus II). Ceci-dit, il est toujours possible de ne pas passer par Qsys et tout faire dans Quartus II.

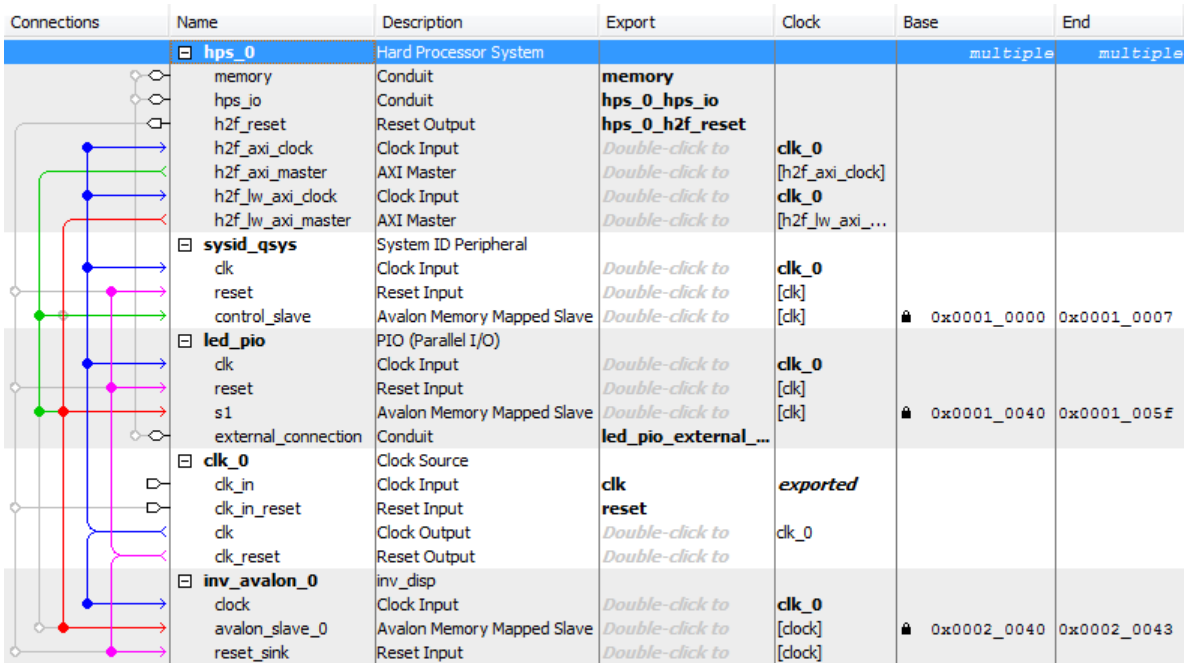

Figure 11 – Vue du projet bare-metal dans Qsys.

La figure 11 montre la composition et l'interconnexion du projet Qsys pour l'application bare-metal. Il y a cinq blocs distincts utilisés dans ce projet, en partant du haut en bas :

- HPS - Le processeur hardcore ARM et ses interfaces d'entrée/sortie.
- System ID - Permet d'assigner une valeur au système. Dans ce cas, elle vaut 0x11112222.
- LED Parallel I/O (PIO)- Conduit pour les LEDs de la partie FPGA (Voir figure 9).
- Clock Source - Indispensable dans tout design. Il permet de distribuer une clock ainsi que d'offrir une source de reset pour les autres IPs.
- Inverseur - Bloc inverseur chargé d'inverser une valeur qui lui est envoyé au niveau binaire. Il ne fait pas partie de la bibliothèque de composants Altera, car il a été crée de toutes pièces pour les besoins de ce projet.

L'interconnexion des blocs a été faite suivant les besoins de la partie software. Les IPs qui seront uniquement accédés par Lightweight AXI, ont étés connectés uniquement via ce bus au processeur. Lorsque l'HPS doit accéder via l'AXI (Bus vert sur la figure 11) et le Lightweight AXI (Bus rouge sur la figure 11) au même IP, alors il est possible de connecter les deux bus sur l'entrée de l'IP (C'est le cas de l'IP LEDs PIO). Le bus AXI a été configuré pour travailler avec une largeur de données de 64 bits. Le bus Lightweight AXI a une largeur fixe de 32 bits, comme stipulé précédemment.

Le signal bleu correspond à la clock pour la partie FPGA du SoC. Elle est utilisée pour clocker tous les IPs ainsi que les deux interfaces AXI maîtres dans le HPS. Le signal pourpre est le signal de reset général de l'Arria V SoC importé depuis le top.

Il faut être conscient que Qsys met une couche d'abstraction lors de l'interconnexion, simplifiant ainsi la tache au développeur. Néanmoins, ceci peut s'avérer dangereux pour certains designs (Pro-

blèmes de timing). Dans ce design, une couche d'abstraction est ajoutée à chaque connexion de bus AXI aux IP. En effet, tous les IP utilisés ont une interface d'entrée utilisant le bus Avalon.

Le bus Avalon a été développé par Altera dans le but de simplifier l'interconnexion dans leurs FPGAs. Pour cette raison c'est le bus par défaut utilisé dans les designs Altera. Dans ce design, des maîtres AXI (HPS) sont connectés sur des esclaves Avalon. Ceci ne peut pas être fait tel quel, les deux bus ont une architecture différente. Pour cette raison, lors de l'interconnexion, Qsys met de blocs de conversion AXI-Avalon normalement non visibles dans Qsys. Pour voir les blocs de connexion, il suffit de cliquer sur System → Show System With Qsys Interconnect dans Qsys.

Qsys permet d'assigner des adresses pour chaque IP instancié, si ce dernier utilise un système de bus. Dans le cas de ce projet, les blocs System ID, LEDs PIO et l'IP inverseur ont tous une adresse. Suivant l'espace mémoire accessible de chaque bloc l'adresse de fin peut changer. Cette adresse doit être multiple de la largeur du bus de données qui y accède. Ainsi, par exemple pour le bloc System ID un seul mot de 64 bits peut être lu, ce qui représente un espace mémoire de 8 bytes. Ces adresses seront reprises dans le programme software pour accéder aux IPs.

Une fois le design généré dans Qsys et compilé dans Quartus, le fichier `.sof` avec le bitstream du design sera utilisé par le HPS. De cette manière c'est le HPS qui programmera la partie FPGA.

Code VHDL du bloc inverseur Le bloc inverseur a été décrit en VHDL afin d'inverser une valeur au niveau binaire. Contrairement aux autres blocs utilisés dans le design, ce dernier a été ajouté manuellement à la libraire de composants de Qsys.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY inv IS
5     PORT
6     (
7        clk, resetn    : IN STD_LOGIC;      -- Clock and active low reset signal
8        read, write    : IN STD_LOGIC;      -- Read and write signals
9        readdata       : IN STD_LOGIC_VECTOR (31 DOWNTO 0); -- Data read from the master
10       writedata      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0) -- Data write to the master
11    );
12 END inv;
13
14 -- Description of the device
15 ARCHITECTURE Behavior OF inv IS
16
17 -- Signal to keep the data from the master
18 SIGNAL reg           : STD_LOGIC_VECTOR (31 DOWNTO 0);
19
20 BEGIN
21    PROCESS(clk)
22    BEGIN
23       IF RISING_EDGE(clk) THEN   -- On rising edge of the clock
24          IF resetn = '0' THEN
25             reg <= "00000000000000000000000000000000";
26          ELSE
27             -- If a write is desired just copy the data from the master
28             IF (write = '1') THEN
29                reg <= readdata;
30             END IF;
31
32             -- In case of a read invert the data stored in the register
33             IF (read = '1') THEN
34                writedata <= NOT(reg);
35             END IF;
36          END IF;
37       END IF;
38    END PROCESS;
39 END Behavior;
```

Code 1 – Code VHDL du bloc inverseur

Le code 1 a été crée en reprenant un exemple fourni par Altera utilisant le bus Avalon [8]. Les signaux dans l'entitée sont déclarés pour être compatibles avec le bus Avalon. De cette manière, lors de l'ajout dans Qsys, il sera possible d'instancier l'interface en tant que bus Avalon.

Le fonctionnement de ce code est très facile. Le composant intègre un registre de 32 bits (Peu importe si l'interface entre le HPS et l'IP est de 64 bits. Qsys connectera les deux composants en conséquence). La valeur de ce registre est mise à 0 lors d'un reset, puis lorsque un accès en écriture est demandée par un maître (HPS), la valeur transmise est stockée dans ce registre. Lors d'un accès en lecture, cette valeur sera inversée bit à bit, puis le résultat sera transmis au maître.

6.2.2 Programme software

Une fois la partie hardware en place, le programme pour le HPS peut être écrit. Ce dernier aura pour but de commander la partie hardware du design.

Le programme a été développé avec DS-5 Studio en partant d'un exemple bare-metal fourni dans le kit software de la carte de développement. Son fonctionnement a été décrit en grandes lignes au début de ce chapitre. Pour cette raison la suite de cette section décrira le code développé plus en détail. Tous les types et les fonctions non standard utilisés dans ce code font partie des librairies hardware Altera décrites plus tôt.

L'ensemble du code se trouve en annexe A.1.

Configuration de la partie FPGA La première fonction majeure utilisée dans le code est `test_config_full`. Elle sert à configurer la partie FPGA de l'Arria V SoC. Elle vérifie si les jumpers MSEL (Servants à choisir le type de configuration et le type du bitstream) de la carte Cyclone V SoC sont dans la bonne position, puis elle tente de charger le bitstream dans la FPGA.

Le chemin vers le fichier `.sof` généré avec le bitstream doit être précisé dans le Makefile lors de la compilation du programme software.

Accès aux bridges La fonction `test_bridge` est la plus intéressante dans ce programme, pour cette raison elle sera expliquée plus en détails. Elle permet d'initialiser les bridges ainsi que de tester la communication avec les IPs placés dans la partie FPGA de l'Arria V SoC.

Les premières lignes de la fonction définissent les adresses de départ des bridges dans le memory map, ainsi que les adresses de départ des IPs définies dans Qsys.

```
1  const uint32_t ALT_LW_H2F = 0xFF200000;
2  const uint32_t ALT_H2F = 0xC0000000;
3
4  const uint32_t ALT_LED_OFFSET = 0x00010040;
5  const uint32_t ALT_ID_OFFSET = 0x00010000;
6  const uint32_t ALT_INV_OFFSET = 0x00020040;
```

La première fonction appelé est `alt_bridge_init`. Comme son nom l'indique elle permet d'initialiser les bridges AXI en configurant leurs registres GPV.

```
1  ALT_STATUS_CODE status1 = alt_bridge_init(ALT_BRIDGE_H2F,   NULL, NULL);
2  ALT_STATUS_CODE status2 = alt_bridge_init(ALT_BRIDGE_LWH2F, NULL, NULL);
```

Si l'initialisation des deux bridges (AXI et Lightweight AXI) s'est bien passée, alors la communication peut se faire par le biais de ces bridges. Dans ce cas, le premier IPs à être lu est le System ID. Sa présence dans le design n'est pas uniquement pour la démonstration. Altera préconise d'instancier cet IP dans chaque design SoC, puis en premier lieu de lire sa valeur afin de vérifier que celle ci correspond bel et bien à la valeur escomptée. Cette action permet de vérifier que les bridges et les bus fonctionnent correctement.

Pour lire le System ID avec l'interface AXI, il faut utiliser la fonction `alt_read_word` des libraires hardware. Cette fonction lit un mot de 32 bits à l'adresse donnnée. Dans ce cas, l'adresse du composant est celle de base du bridge AXI plus l'adresse assignée dans Qsys.

```
1    uint32_t sysid = alt_read_word(ALT_H2F + ALT_ID_OFFSET);
2    printf("TEST 1: Get System ID (H2F) = 0x%x.\n", (unsigned int)sysid);
```

Pour un accès en écriture, le principe reste le même. Pour envoyer des valeurs aux LEDs la fonction `alt_write_word` est employée. D'autres fonctions d'écriture et de lecture existent suivant la taille du mot qui doit être écrite.

```
1    for (uint32_t i = 0; i < 50; i++)
2    {
3       alt_write_word(ALT_H2F + ALT_LED_OFFSET, 0); // Turn on LEDs AXI
4       delay(100000);
5       alt_write_word(ALT_LW_H2F + ALT_LED_OFFSET, 0xF);  // Turn off LEDs LwAXI
6       delay(100000);
7    }
```

Finalement une valeur est envoyée au bloc inverseur avec la même fonction d'écriture, puis la valeur inversée est lue.

```
1    // Send a value to invert
2    alt_write_word(ALT_LW_H2F + ALT_INV_OFFSET, 0xAAAAAAAA);
3
4    // Then read the inverted value
5    printf("TEST 3: Inverted value: 0x%x\n", alt_read_word(ALT_LW_H2F + ALT_INV_OFFSET));
```

Finalement les bridges sont fermés et le programme se termine. Ce programme de test a été lancé en mode debug dans DS-5, ce qui permet de voir la sortie console sur le même programme.

Tester le fonctionnement du processeur et des bridges en bare-metal est très important, même si le concepteur souhaite embarquer un OS pour le processeur. Bien connaître le fonctionnement bas niveau des bridges permet d'éviter des pièges lorsqu'on travaille avec un OS, surtout que certaines fonctions de la libraire hardware sont réutilisées avec un OS.

7 Système d'exploitation et libraires bare-metal

Travailler en bare-metal est possible, cependant il peut s'avérer très difficile d'écrire un programme plus complexe sans OS. Créer un programme qui implémente TCP/IP de toutes pièces est très complexe. Pour cette raison, une étude des systèmes d'exploitation existants pouvant être facilement utilisée pour l'Arria V SoCa dû être menée. Le souhait du CERN était d'éviter l'utilisation de Linux et privilégier un Real Time OS plus léger ou même des librairies bare metal.

Pour comparer les diverses solutions trouvées, certains critères ont étés pris en compte. Ces derniers étaient, du plus influent au moins influent :

- La compatibilité - Le niveau de compatibilité avec l'Arria V SoC. Ce paramètre influe fortement sur le temps de développement du projet.
- Licence - Le type de licence avec laquelle la solution doit être employé.
- Prix.
- Dernière release - Donne une idée sur la pérennité de la solution et si elle est encore développée et à quel rythme.
- Débit - Information de débit obtenue. C'est une indication secondaire qui varie fortement en fonction du processeur, mais qui peut donner une certaine indication sur les performances de la solution.
- Références - Liste d'entreprises et organisations qui utilisent cette solution.

Afin d'obtenir les critères cités, des recherches sur internet, ainsi que des fabricants ont été contactés. Ceci à permis de lister un premier lot de solutions applicables pour ce projet. Les solutions primaires sélectionnes se trouvent dans les tableaux 4 et 5 en annexe.

Cette liste a été d'avantage affinée après des réflexions quand à la facilité d'être utilisée dans ce projet. Ainsi, trois solutions sont sorties du lot :

- Multicore Abassi - En tant que RTOS, il offre les possibilités les plus intéressantes.
- NicheStack - Librairie bare-metal intéressante. Malheureusement, un portage est nécessaire.
- Embedded Linux - Malgré qu'il n'est pas listé dans les tableaux, il a été pris en considération en tant que solution de secours.

Multicore Abassi a été le plus privilégié pour ses qualités ainsi que pour son faible prix. De plus le support s'est montré très réactif et compétant. Par contre c'est durant la demande de prix que cette solution a perdu son attrait. En effet, le support a demandé un prix de 24'000 $ au lieu des 8'000 $ comme indiqué dans leur brochure, sous prétexte que le prix de 8'000 $ s'appliquait uniquement aux systèmes mono cœurs. De ce fait cette solution a été abandonnée. Quand à la librairie NicheSack, après avoir fait une demande de prix, un prix de 10'000 € a été communiqué, et ce uniquement pour une version IPv4.

Finalement, après réflexion, la décision a été prise d'utiliser Embedded Linux, en tout cas dans le cadre dans ce travail de bachelor. Les autres solutions du tableau ont étés rejetées pour diverses raisons, surtout pour le manque de compatibilité. En effet, la durée d'un travail de bachelor ne permettait pas de s'attarder sur l'adaptation d'une librairie ou d'un RTOS.

Ceci-dit, dans un développement plus avancé de ce projet, pour éviter d'employer Embedded Linux, il est possible de consacrer du temps afin d'adapter une solution. Dans ce cas, il serait possible d'écrire les drivers pour la libraire lwIP, qui est une solution open source avec une importante communauté de développeurs actifs et qui est réputée pour ses bonnes performances. De plus elle est compatible IPv4 ou IPv6.

8 Embedded Linux

Désormais l'OS qui sera utilisé dans la suite de ce projet est Embedded Linux. Il s'agit d'une version fournie par Altera, qui est adaptée pour la carte de développement utilisée. Altera supporte et encourage vivement l'utilisation de Linux sur l'Arria V SoC. Pour cette raison, il existe un site dédié à l'utilisation de Embedded Linuxappelé RocketBoards [3]. Il contient l'essentiel des informations ainsi que des exemples.

8.1 Image Linux

C'est sur ce site que l'image précompilée de Embedded LinuxAltera est disponible. Une fois téléchargée, elle doit être écrite sur un périphérique. Dans le cadre de ce projet, il s'agira de la carte SD (Voir figure 8 pour son emplacement). En utilisant une carte mémoire, une flexibilité maximale est assurée lors de l'étape de développement.

Avant de pouvoir lancer Embedded Linuxsur la carte de ceveloppement, il est nécessaire de la configurer à booter sur la carte SD. Pour ce faire il faut suivre l'explication donnée sur RocketsBoards [9]. En particulier il faut veiller à ce que la position des switch soit correcte afin que le processus de boot débute bel et bien depuis la carte SD.

8.2 Boot

Utiliser une carte SD lors de la phase de développent sur la carte Altera est très pratique, cependant dans la phase d'implémentation sur une carte SCMS réelle, ce support est exclu.

L'Arria V SoC permet de booter Embedded Linux à partir des autres sources, comme par exemple la mémoire QSPI. Cette mémoire de 256 $[MB]$ externe au composant est prévue à cet effet. Il est possible d'y stocker le projet de base GSRD (Env. 150 $[MB]$). La figure 12 montre le processus dans sa globalité afin de booter Embedded Linux.

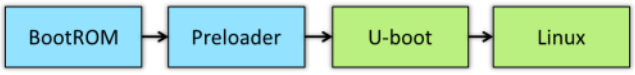

Figure 12 – Boot de l'Arria V SoC. Les blocs vert ne sont pas obligatoires dans le processus de boot. [10]

La première étape du boot de l'HPS est effectuée automatiquement après un allumage de l'Arria V SoC. C'est la mémoire Boot ROM qui s'en occupe, comme expliqué dans le chapitre dédié à l'address map. Ensuite, un premier programme se lance, le preloader. Ce dernier configure l'Arria V SoC puis charge U-Boot, un autre programme plus évolué qui lui chargera finalement Embedded Linux.

Ceci-dit, il n'est pas obligatoire de charger U-Boot et Embedded Linux. Seul les deux premières étapes du processus de boot sont obligatoires (En bleu sur la figure 12). Un autre OS peut être chargé directement après le preloader, voir même un programme bare-metal. Embedded Linux seul ne peut pas être chargé directement par le preloader, car il est bien plus complexe qu'un programme bare-metal.

8.2.1 Preloader

Suivant ce qui doit être chargé par le preloader, son comportement varie. Pour cette raison, il est nécessaire de lui indiquer certains paramètres. Le faire à la main serait très difficile, car il faudrait manipuler les fichier sources de celui-ci. Pour cette raison Altera a mis à disposition dans la suite EDS un utilitaire permettant de faire ceci de manière graphique. Il s'agit de BSP Editor. Pour le lancer il faut lancer la commande `bsp-editor &` dans le terminal de l'EDS.

3. http://www.rocketboards.org/foswiki/Documentation/AlteraSoCDevelopmentBoard

Dans la fenêtre de BSP Editor, il est possible d'indiquer à partir de quel support le restant du processus de boot se fera (QSPI, SDMMC, RAM), ainsi que d'autres options moins essentielles pour ce projet. Une fois le preloader modifié, un fichier `.bsp` est généré. C'est ce dernier qui sera utilise lors de la compilation du preloader. La figure 13 montre le processus de compilation du preloader.

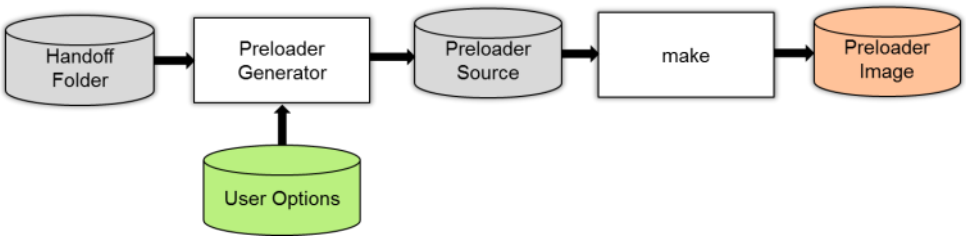

Figure 13 – Génération du preloader. Handoff folder est celui du projet GSRD, le preloader generator est le BSP Editor. Make correspond à l'étape de compilation du preloader. [11]

Une fois l'image du preloader obtenue, elle peut être utilisée dans le processus du boot. Comme ce travail de bachelor a été uniquement fait en utilisant la carte SD, la recompilation du preloader n'a pas été effectuée, car l'image Embedded Linux fournie par Altera était déjà prête à l'emploi avec un preloader par défaut.

Néanmoins, cette étape peut s'avérer utile dans le futur, si autre chose que Embedded Linux devra être booté ou d'un autre endroit que la carte SD.

8.3 Device tree

Un des points le plus crucial concerne la communication entre Embedded Linux et le matériel. Avec un programme bare-metal, l'accès au matériel se fait d'une façon directe. Avec Embedded Linux ce n'est plus possible. L'utilisateur n'a pas accès au matériel d'une telle manière. Habituellement, des drivers sont utilisés pour accéder à chaque périphérique matériel. Cette manière de faire offre peu de flexibilité et est source d'erreurs (Ecriture de driver pour chaque périphérique).

Un autre système d'accès existe. Il s'agit du device tree, une description de matériel de façon structurée qui permet à un kernel compilé une seule fois d'avoir accès au matériel. C'est ce mode de fonctionnement qui a été implémenté par Altera.

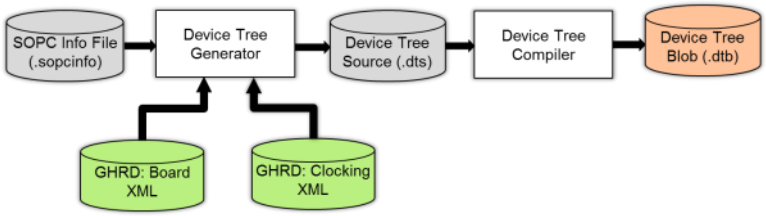

Figure 14 – Génération du device tree. Le fichier `.sopcinfo` est géneré par la compilation Quartus, device tree generator correspond à la commande `sopc2dts` et device tree compiler correspond à la commande `dtc`. [12]

Le conception hardware du design se fait de la même manière que pour un design bare-metal, via Qsys et Quartus. Une fois le design compilé, un fichier `.sopcinfo` est généré. Il contient la description du design. C'est à partir de ce fichier que le device tree sera créé au bout d'un processus (Voir figure 14). La suite de commandes à lancer via le terminal d'EDS est décrite sur le site RocketBoards [12].

Ceci peut très vite s'avérer ennuyeux, surtout si le device tree doit être generée plusieurs fois. Pour cette raison, un script batch a été écrit (Code 2). Il est à placer à la racine du répertoire du projet GSRD.

```
@echo off
echo ---------------------------------------------
echo -- Generating bitstream and device tree ... --
echo ---------------------------------------------

cd hardware/
quartus_cpf.exe -c output_files/ghrd_5astfd5k3.sof output_files/soc_system.rbf
sopc2dts --input ghrd_5astfd5k3.sopcinfo --output socfpga.dts --board
    ghrd_5astfd5k3_board_info.xml --board hps_clock_info.xml
dtc -I dts -O dtb -o socfpga.dtb socfpga.dts
```

Code 2 – Script batch automatisant le processus de génération du device tree.

Le fichier `.sof` avec le bitstream généré avec Quartus, est converti en fichier avec de bitstream compressé (Fichier `.rbf`).

Au final, deux fichiers nécessaires pour décrire le hardware à Embedded Linux seront générés :

- soc_system.rbf - Fichier contenant le bitstream de la FPGA compressé.
- socfpga.dtb - Fichier avec le device tree compilé.

Ces deux fichiers sont à placer à la racine de la seule partition accessible de la carte SD. De cette manière, au prochain boot du HPS à partir de la carte SD, U-Boot chargera le fichier `.dtb` pour donner l'accès au matériel à l'Embedded Linux. U-Boot programmera aussi la partie FPGA de l'Arria V SoC avec le fichier `.rbf`.

Ce système montre bien la facilité et la flexibilité de réconfiguration du matériel avec un device tree. Si un changement de matériel doit avoir lieu, seul la partie hardware du design est modifiée, puis les fichiers nécessaires sont placés sur la carte SD. Cette façon de faire est extrêmement appréciée lors de la phase de développement.

8.4 Communication HPS-FPGA

Dès que le device tree a été généré, il reste encore un programme software à écrire pour communiquer avec ce dernier. Pour ce faire il est nécessaire d'utiliser DS-5 et de partir d'un exemple donné utilisant Embedded Linux et le compilateur GNU du projet GSRD. De là, toutes les fonctions standard UNIX sont disponibles.

Supposant que le device tree a été correctement généré pour le même design hardware que celui de l'application bare-metal (Figure 11), et que le programme software est chargé de lire la valeur du bloc system ID.

```
int main(int agc, char *argv[])
{
   int fd;
   void *virtualBaseLwAxi;

   const uint32_t LW_AXI_BASE = 0xff200000;  // Beginning address of Lw AXI
   const uint32_t LW_AXI_SPAN = 0x200000;    // Span of Lw AXI in memory

   const uint32_t SYSTEM_ID = 0x10000;       // Address set in Qsys for sysID

   // Open file giving access to memory
   fd = open("/dev/mem", (O_RDWR | O_SYNC));

   // Map memory into user space
   virtualBaseLwAxi = mmap(NULL, LW_AXI_SPAN, (PROT_READ | PROT_WRITE), MAP_SHARED, fd,
       LW_AXI_BASE);

   // Read system ID
   printf("System ID: 0x%x\n", alt_read_word(virtualBaseLwAxi + LW_AXI_BASE + SYSTEM_ID);

   return 0;
}
```

Code 3 – Exemple de la manière d'accéder au matéreil dans un programme Embedded Linux.

Le code 3 montre le strict minimum pur lire le system ID du design hardware. La fonction `alt_read_word` de l'hardware library Altera déjà utilisée dans le programme bare-metal est réemployée. Ce qui diffère, c'est que l'adresse physique de l'IP n'est pas donnée directement, mais elle est ajoutée à un offset.

Ceci est dû au fait que Embedded Linux ne donne pas accès au matériel directement depuis l'user space. Pour avoir comme même un accès au matériel, il faut mapper le fichier d'Embedded Linux correspondant au memory map dans l'user space (/dev/mem). C'est la fonction `mmap` qui est chargée de faire ça. Faire un memory mapping de la mémoire physique dans l'user space, correspond en quelques sortes à utiliser une look-up table pour accéder aux adresses physiques.

Dans l'exemple actuel, il est demandé à la fonction `mmap` de donner une adresse d'offset virtuelle qui correspondra à l'adresse physique 0 pour accéder au bridge Lightweight AXI. Pour ce faire, il est nécessaire de donner l'adresse de départ et l'espace occupé dans la mémoire par ce bridge à `mmap` .

Une fois l'adresse d'offset obtenue, l'accès en lecture ou en écriture se fait de la même manière qu'en bare-metal.

9 Connexion Ethernet

Cette section traite un point essentiel de ce travail de bachelor, qui est la communication par Ethernet. Le cahier de charges stipulant qu'aucune perte de données n'est acceptable, le protocole TCP doit être utilisé. En effet, son fonctionnement intègre des mécanismes d'acquittement permettant d'indiquer si une perte des donnés ou des données corrompues ont étés reçues. Dans un tel cas, une partie des données est renvoyée afin de compléter les données manquantes. Il existe un autre protocole de connexion, UDP, mais celui-ci implique des pertes de paquets.

La connexion physique entre la carte (Voir figure 8 pour l'emplacement du port sur la carte) et l'ordinateur se fait par câble Ethernet en lien direct. L'ordinateur distinct doit utiliser une carte réseau Gigabit Ethernet.

Afin de faire communiquer deux périphériques (Dans ce cas, la carte de développement et un ordinateur), une architecture maître-esclave, ou client-serveur dans la terminologie du protocole internet est utilisé. Cette architecture permet d'envoyer des ordres au serveur puis d'attendre sa réponse.

9.1 Sockets

Les sockets sont un ensemble de fonctions uniformisées permettant la communication entre ordinateurs. Dans ce projet ils seront utilisés afin de mettre en place le système de communication entre la carte de développement et l'ordinateur distinct. Les fonctions du socket sont disponibles dans la quasi totalité de langages de programmation, ce qui les rend très portables et populaires.

Pour initialiser une connexion basique, quelques fonctions doivent êtres appelées du côté serveur ainsi que du côté client.

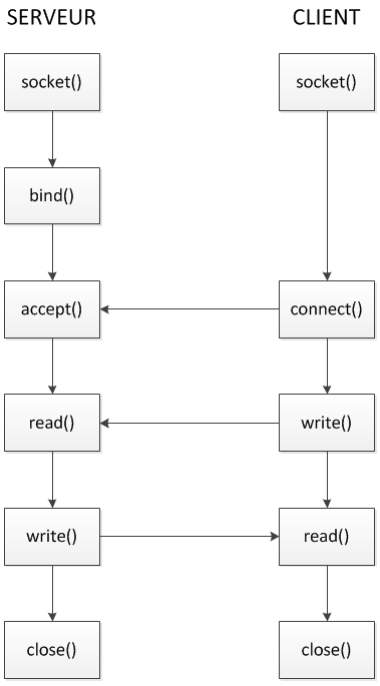

Figure 15 – Schéma typique de connexion entre serveur et client.

La figure 15 permet de voir le déroulement d'un processus de connexion entre le serveur et le client. Normalement, le programme serveur démarre avant le programme client. En premier lieu, il appelle la fonction `socket`, qui est chargé d'ouvrir un socket. Après quoi le serveur appelle la fonction `bind`, qui permet de lier une adresse IP avec le socket précédemment crée. Après quoi le serveur attend une connexion du client via la fonction `accept`. Cette fonction est bloquante tant qu'une requête de connexion du client n'est pas venue.

Une fois cette procédure achevée, le serveur est prêt à communiquer avec le client par le biais des fonctions `read` et `write`.

Du côté client, quasiment les mêmes fonctions sont utilisés, excepté la fonction `connect`, qui comme son nom l'indique, est chargée de se connecter à un serveur. Après quoi, il ne reste plus qu'à établir la communication avec le serveur.

Finalement, lorsque la connexion est finie, il ne reste qu'à fermer le socket via la fonction `close`.

10 Test de vitesse

Malgré le fait qu'une connexion Gigabit est disponible sur la carte de développement, un test de vitesse a été effectué afin de s'assurer que la vitesse réelle est suffisante pour attendre le débit exigé.

Pour ce faire, un outil de mesure de performances du réseau a été employé. Cet outil, appelé Iperf [4], permet de lancer divers tests, entre autre un test de la bande passante utile. C'est le test qu'il est nécessaire de faire dans le cadre de ce travail de bachelor.

Afin d'effectuer un test de vitesse plus proche de l'application finale, un programme à base de sockets a été écrit. Ce dernier est chargé d'envoyer une quantité de données définie au client ainsi que de mesurer le temps de transmission. De cette manière, il est possible de calculer le débit utile.

10.1 Iperf

Iperf est disponible en tant qu'exécutable pour divers plateformes et divers systèmes d'exploitation. Malheureusement, il n'est pas disponible directement pour le processeur Cortex-A9. Iperf étant open source, il est possible de cross-compiler les sources pour le processeur de son choix. Ceci a été fait dans le cadre de ce projet. Les manipulations décrites ci-dessous ont étés faites en utilisant un tutoriel adapté pour le processeur Cortex-A9 [13].

Le processus de cross-compilation doit être réalisé sur une machine tournant sur Linux (Dans ce cas Ubuntu 14.04 LTS) avec la bonne toolchain installée. La seule étape qui diffère par rapport au tutoriel cité, est l'étape de configuration des sources. A la place de la commande donnée sur le tutoriel, il faut utiliser :

```
./configure -build=arm-linux-gnueabihf -host=armv7-none-linux-gnueabihf
```

Cette commande préparera les sources pour être cross-compilées pour Cortex-A9. En effet, ce processeur possède une architecture ARMv7, et donc il utilise le jeu d'instructions associé. Finalement il ne reste qu'à lancer la commande make pour obtenir l'exécutable pour le processeur de l'Arria V SoC.

L'exécutable généré peut être transféré dans un des répertoires de Embedded Linux par ssh en utilisant DS-5.

10.1.1 Test de débit

Une fois l'exécutable de Iperf en place sur la carte de developpement, il est possible de lancer le test de vitesse. Pour ça, il faut lancer une instance de Iperf sur la carte de developpement en executant :

```
./iperf -s
```

Ceci permet de lancer un serveur sur la carte de développement attendant la connexion d'une instance client de Iperf. Pour lancer cette dernière, il faut lancer la commande suivante (Les x correspondent à l'adresse IP du serveur de la carte de développement) :

4. iperf.fr

```
./iperf -c xxx.xxx.xxx.xxx
```

Ceci aura pour effet de lancer le test de débit. Une fois le test fini, les résultats sont affichés. Il est possible de changer la fenêtre TCP en ajoutant l'argument -w dans la commande précédente. Par exemple :

```
./iperf -c xxx.xxx.xxx.xxx -w 200k
```

Aura pour effet d'effectuer le test avec une fenêtre TCP de 200 $[kB]$. La fenêtre TCP correspond à la quantité de données qui sera transmise avant que des paquets d'acquittement propres au protocole TCP soient transmis. Plus la fenêtre est petite, plus il y a des paquets d'acquittement transmis. Ces paquets utilisent une certaine quantité de la bande passante pour être envoyés, pour cette raison, pour ne pas gaspiller de la bande passante utile, ces paquets ont intérêt d'être les plus grand possibles.

Cependant, si la fenêtre est trop grande et une erreur de transmission est détecté, alors un bloc important d'informations devra être retransmis.

Résultat de test Le tableau 3 montre les résultats obtenus avec Iperf en fonction de la taille de la fenêtre TCP.

Fenetre TCP $[kB]$	**Débit $[Mbits/s]$**
64 (défaut)	487
100	500
128	735
200	735

Tableau 3 – Résultats du test de vitesse avec Iperf.

Avec une taille de fenêtre TCP par défaut de 64 $[kB]$, un débit d'environ 490 $[Mbits/s]$ est obtenu. Si la fentre est augmenté, le débit utile augmente à son tour. Au bout d'une certaine valeur (A savoir 128 $[kB]$), le débit n'augmente plus et il atteint sa valeur maximale de 735 $[Mbits/s]$.

Pour cette raison, connaissant le débit exigé pour ce travail de bachelor (400 $[Mbits/s]$), il n'est pas nécessaire d'augmenter la fenêtre TCP plus que la valeur par défaut. Ceci-dit, si dans le futur un débit plus important est demandé, il sera toujours possible d'augmenter la taille de la fenêtre TCP.

10.2 Programme de mesure

Comme stipulé, un programme de test de vitesse a été écrit. Il est chargé de mesurer le temps de transmission d'un bloc de données via Ethernet. Ce test à pour but de vérifier que les résultats avec un programme écrit et Iperf sont semblables. Les sources du programme du serveur (Tournant sur l'Arria V SoC) et du client se trouvent en annexe code 6 et code 7.

Le programme client doit être compilé et lancé sur un système Linux. Le programme serveur envoi une quantité de données passée en argument lors du lancement du programme, il faut aussi lui indiquer le port sur lequel ouvrir le socket. Le programme client nécessite l'adresse IP du serveur. Une fois ce dernier lancé, il envois une requête au serveur, puis les données de test sont transmises de ce dernier. Le client mesure le temps entre l'arrivée des données, puis il le somme. La taille totale du paquet est connue, donc le débit peut être calculé facilement par le client.

Résultats de débit Les test effectués avec les programmes de mesure, ont donné un débit d'environ **200** $[Mbits/s]$. C'est un résultat bien plus faible qu'avec Iperf. Malgré que le débit souhaité de 400 $[Mbits/s]$ n'est pas atteint, ce résultat n'est pas synonyme d'échec du projet. En effet, comme dit précédemment, un débit plus faible peut être accepté pour une première version de ce projet.

10.3 Analyse de résultats

Il y a une différence évidente entre les résultats de débit entre les deux manières de faire les mesures. Le test avec Iperf indique un débit utile maximal de 735 $[Mbits/s]$ et celui avec le programme de mesure 200 $[Mbits/s]$.

Le résultat d'Iperf semble correct. Les débits maximaux TCP pouvant être atteints avec un lien Gigabit Ethernet avoisinent 750 $[Mbits/s]$ et non 1 $[Gbits/s]$ comme il serait tentant de penser. Ceci à cause de l'overhead et des paquets de contrôle insérés dans le flux de donnés par TCP. De plus, Iperf est un outil largement utilisé pour les mesures des réseaux.

Pour ces raisons, le résultat obtenu avec le programme de mesure est à prendre à titre indicatif. Malgré que la taille de la fenêtre TCP n'a pas été changée, le résultat devrait être plus important. Il peut même s'agir d'une erreur de programmation.

De ce fait il faut considérer le résultat de l'outil Iperf comme correct.

11 Application de démonstration

Cette section décris l'application finale, qui permet de faire la démonstration grâce à une application tirant profit des tous les aspects vu précédemment. L'ensemble de l'application est partagée en deux parties :

- Partie serveur - Tourne sur l'Arria V SoC, imite le fonctionnement de la carte SCMS.
- Partie client - GUI permettant d'envoyer des commandes à la partie serveur et de recevoir le paquet de données de la partie serveur.

La partie serveur doit imiter grossièrement le fonctionnement de la carte SCMS ainsi qu'exploiter les aspects vu précédemment, càd la communication avec la partie FPGA. La partie client est un programme graphique qui arme le serveur et qui affiche les données reçues de la partie serveur.

L'ensemble du système fonctionne de la manière suivante :

1. Le client envoi une requête d'armement.
2. Le serveur remplit la mémoire avec des donnés aléatoires générées dans un IP de la partie FPGA.
3. Le serveur attend un signal de trigger externe pour commencer l'envoi des donnés au client.
4. Le serveur envoi le paquet scan (Données aléatoires) au client.
5. Le client affiche les données reçues.

Une fois le cycle terminé, il recommence depuis le début.

11.1 Serveur

La partie serveur de l'application de démonstration tire profit du processeur ainsi que de la partie FPGA de l'Arria V SoC. Le schéma bloc de la partie serveur se trouve sur la figure 16.

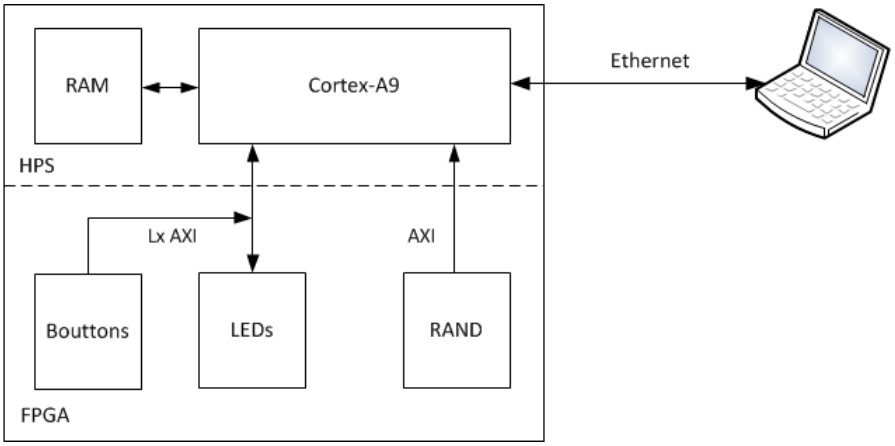

Figure 16 – Schéma bloc de la partie serveur de l'application de démonstration.

La requête d'armement envoyée depuis le client est déclenchée par un appui sur un bouton dans la GUI par l'utilisateur. Après cette dernière, le serveur lance une série d'accès à la partie FPGA via AXI (Interface de 64 bits) où un bloc IP générateur de nombres psydo-aléatoires a été instancié. Chaque valeur est placée dans la mémoire RAM. Lorsque 35 $[MB]$ (Taille du paquet de scan) de données ont étés placés dans la mémoire RAM du processeur, le paquet est prêt à être envoyé. Pour que le processus d'envoi commence, il faut qu'un signal de trigger le déclenche. Durant l'attente du trigger toutes les LEDs sont allumées. Le signal de trigger est simulé par l'appui sur le bouton 0 des boutons de la partie FPGA (Voir figure 9). Lorsqu'un appui sur le bouton est détecté, les LEDs sont éteintes, puis le paquet de scan stockés dans la RAM est envoyé au client. Les accès aux bouton et aux LEDs se font via Lightweight AXI.

11.1.1 Générateur de nombres pseudo-aléatoires

L'IP qui génère les valeurs psydo-aléatoires a été basé sur un exemple donnés sur un blog [14]. Il a été cependant adapté pour fonctionner avec le bus Avalon de 64 bits, d'autres portes XOR ont étés aussi ajoutés. Le code 4 montre son description en VHDL.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY rand IS
5     PORT
6     (
7        clk, resetn    : IN STD_LOGIC;            -- Clock and active low reset signal
8        read, write    : IN STD_LOGIC;                    -- Read signal
9        readdata       : IN STD_LOGIC_VECTOR (63 DOWNTO 0);   -- Data to be read
10       writedata      : OUT STD_LOGIC_VECTOR (63 DOWNTO 0)   -- Data to be write
11    );
12 END rand;
13
14 -- Description of the device
15 ARCHITECTURE Behavior OF rand IS
16    SIGNAL rand_temp  : STD_LOGIC_VECTOR(63 downto 0);
17    SIGNAL temp       : STD_LOGIC;
18 BEGIN
19    PROCESS(clk)
20    BEGIN
21       IF (RISING_EDGE(clk)) THEN
22
23          temp <= rand_temp(63) XOR rand_temp(62) XOR rand_temp(33) XOR rand_temp(7);
24          rand_temp(63 downto 1) <= rand_temp(62 downto 0);
25          rand_temp(0) <= temp;
26
27          IF resetn = '0' THEN
28             temp <= '0';
29             rand_temp <= (63 => '1', others => '0');
30          ELSE
31             IF (read = '1') THEN
32                writedata <= rand_temp;
33             END IF;
34          END IF;
35       END IF;
36    END PROCESS;
37 END Behavior;
```

Code 4 – Description VHDL du générateur de nombres psydo-aléatoires.

Le principe de fonctionnement de ce circuit peut se représenter avec la figure 17. Il s'agit d'un registre à décalage dans lequel l'entrée est la sortie d'une porte XOR. Les entées sont connectés à certains étages du registre à décalage, ce qui permet de générer une valeur psydo-aléatoire à chaque coup de clock (L'histogramme de ce générateur se trouve en annexe A.7).

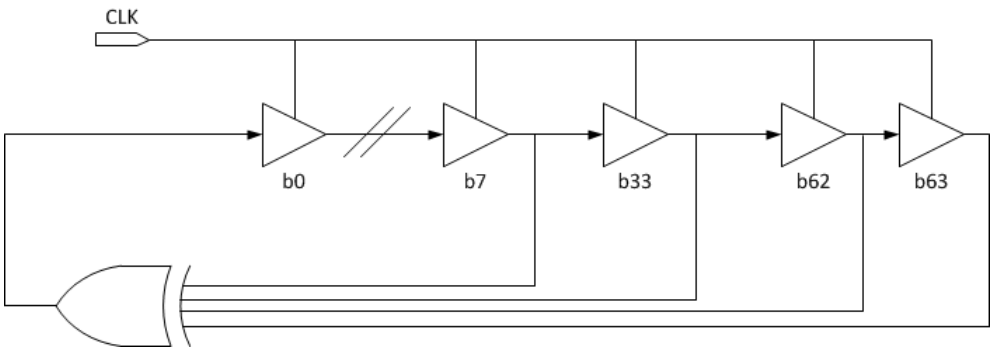

Figure 17 – Schéma bloc du fonctionnement du générateur de nombres psydo-aléatoires.

11.2 Client

Le programme client est un programme GUI fonctionnant sous Windows permettant d'armer le système, puis de recevoir le paquet de scan, ainsi que divers informations concernant la réception. Son interface a été crée en utilisant GTK+ [5]. Cette bibliothèque de création de GUI a été à la base utilisée afin de créer l'interface graphique de GIMP, puis les programmes sous environnement graphique Gnome sous Linux. Elle est multi-platforme et un grand nombre de langages de programmation peuvent être utilisés pour travailler avec, C y compris. C'est par ailleurs une de raisons principale pour laquelle cette bibliothèque à été choisie au lieu d'une autre. En effet, n'ayant pas de connaissances approfondies dans d'autres langages que le C, le choix s'est porté rapidement sur cette bibliothèque.

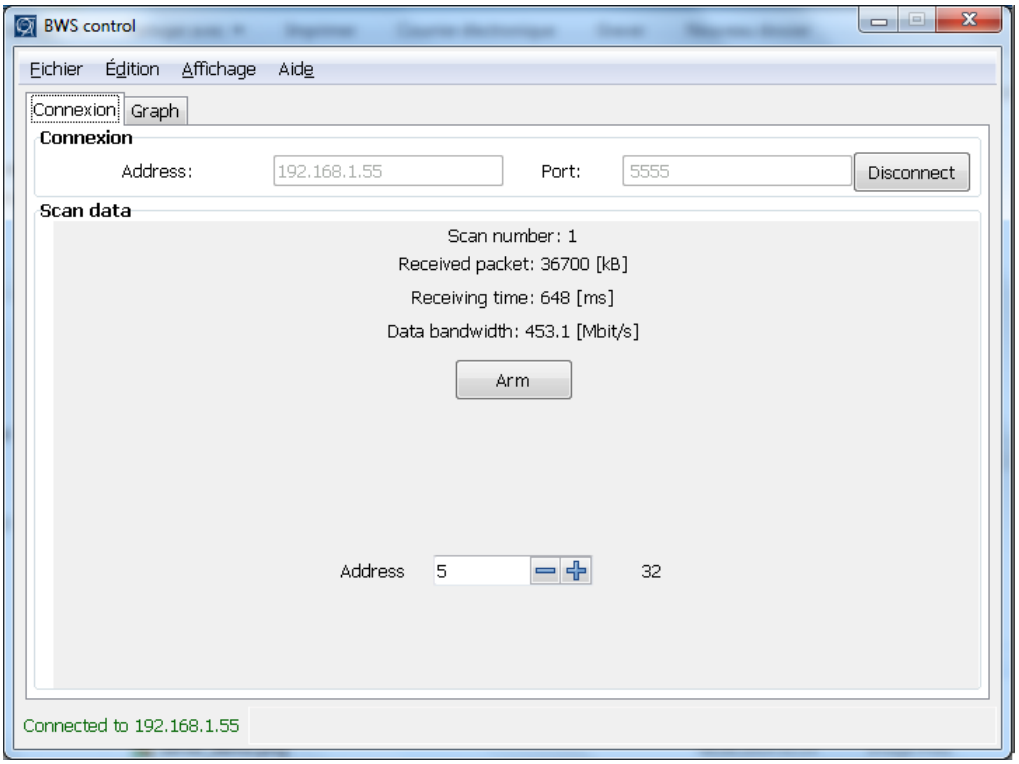

Figure 18 – Vue du onglet connexion du programme BWS control.

La figure 18 montre la vue de l'onglet connexion du programme BWS control. Dans cet onglet il est possible d'armer le serveur (Bouton Arm), puis lorsque le paquet est venu, plusieurs options deviennent visibles :

- Le numéro du paquet de scan reçu (Incrément de un à chaque paquet).
- La taille en $[kB]$ du paquet reçu.
- Le temps de réception du paquet.
- Le débit calculé de la connexion.
- L'entrée "Address" permet de visualiser la valeur du byte indiqué du paquet reçu.

Le deuxième onglet permet d'afficher de manière graphique les 12'000 premières valeurs du paquet reçu. La valeur de 12'000 a été choisie en tant que compromis entre la vitesse d'affichage et la quantité des données affichées. Le serveur, lorsqu'il envoi le paquet de scan simulé avec des valeurs aléatoires, en réalité envoi une courbe gaussienne fixe bruitée avec des valeurs psydo-aléatoires sur les 12'000 premiers échantillons. Le reste de ce paquet contient aucune donnée structurée. Les valeurs de la courbe sont lus à partir d'un fichier texte placé à côté de l'exécutable de démonstration dans un

5. http://www.gtk.org

des répertoires de l'Embedded Linux. La figure 19 montre l'onglet du graph de la courbe gaussienne bruitée.

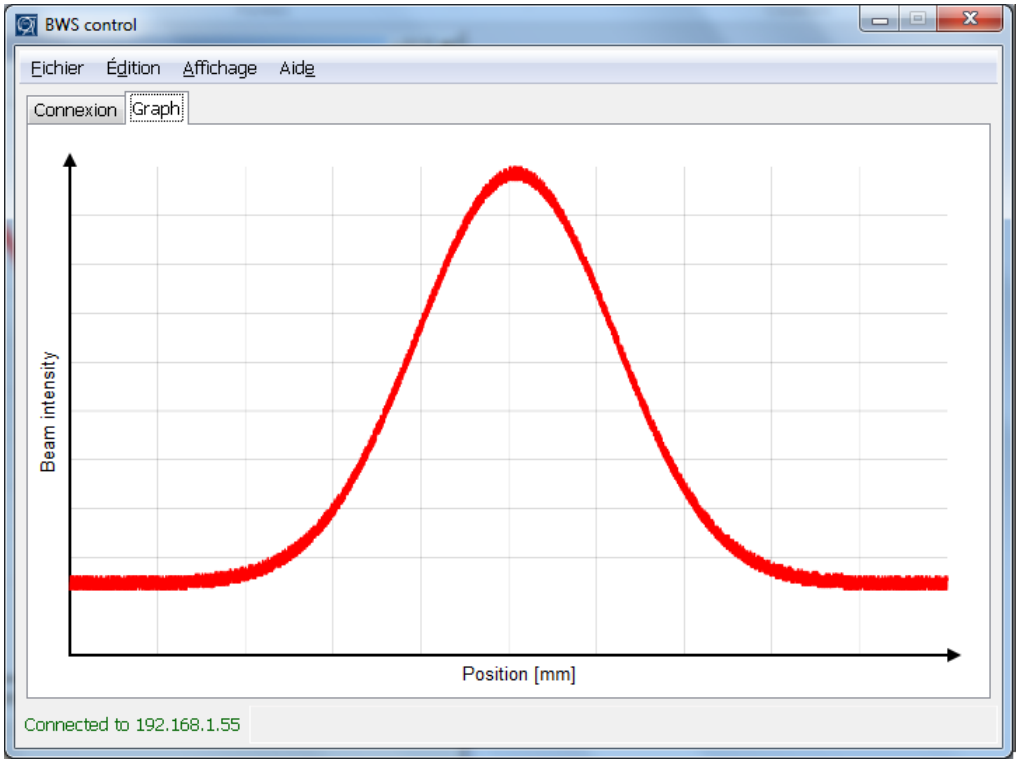

Figure 19 – Vue du onglet graph du programme BWS control.

Cette courbe correspond à un exemple de répartition du faisceau de particules dans le conduit. Normalement la carte SCMS ne collecte pas les données du détecteur de particules secondaires, mais pour les besoins de démonstration, ce type d'affichage a été choisi. Le code source complet de l'application client se trouve en annexe A.8.

11.3 Débit mesuré

En effectuant des mesures avec l'application de démonstration un débit utile de données entre 450 et 500 $[Mbits/s]$ est obtenu. Dans cette application le même système de sockets est employé, mais du coté client des winsocket sont utilisés et du côté serveur des sockets UNIX.

Malgré cela, un débit plus que satisfaisant est obtenu. De plus cette valeur de 450 à 500 $[Mbits/s]$ se recouvre avec les résultats de Iperf. En effet, avec une fenêtre TCP par défaut, le résultat obtenu était de 487 $[Mbits/s]$. Ce qui remet encore plus en question le résultat obtenu avec le programme de mesure de débit.

12 Conclusion

Ce projet de bachelor à permis de démontrer qu'il est tout à fait possible d'implémenter une connexion Ethernet haute vitesse sur la carte de développement Arria V SoC. Le débit escompté peut être atteint sans grandes difficultés (Les résultats du test non concluent sont à ignorer, une erreur a été commise quelque part), et donc le paquet de scan peut être transmis en respectant les temps alloués.

L'application de démonstration, surtout au niveau du serveur, est très basique et très mal optimisée. Par exemple, pour le moment le paquet de scan est placé en mémoire par le biais du HPS. Ceci est une aberration d'un point de vue du codesign. En effet, ayant accès à une FPGA il serait plus judicieux d'implémenter un système de DMA hardware qui placerait le paquet dans la mémoire vive du processeur, surtout qu'un bridge permettant l'accès au HPS en tant qu'esclave à un maître dans la partie FPGA existe (Il n'a pas été utilisé dans ce projet pour l'instant).

Malheureusement, l'envoi du paquet de statut n'a pas pu être implémenté à cause du manque de temps. Ceci pourra être fait en reprenant le programme de démonstration actuel, et en y ajoutant des threads, un pour chaque paquet à envoyer.

Outre ce point manquant du cahier de charges, le restant a été bel et bien effectué. Le composant Arria V SoC a été suffisamment décrit pour l'application dans ce projet. Concernant le système d'exploitation, si ce dernier ne correspond finalement pas aux attentes du personnel du CERN, il est encore possible de le changer.

Multicore Abassi pourrait être une bonne solution, surtout que durant les derniers jours de ce travail de bachelor, leurs responsables marketing se sont rendus compte de l'occasion qu'ils manquaient d'avoir le CERN dans leurs références. Pour cette raison ils étaient disposés à proposer un prix bien inférieur aux 24'000 $ exigés au départ.

Dans sa globalité, ce projet fut un succès qui ouvre des vois à un développent futur et qui offre des outils basiques pour aider les ingénieurs dans ce développement.

Yverdon-les-Bains, le 8 août 2014

Kamil Florek _____

Références

[1] Communication Group. *LHC Guide*. CERN, 2008.

[2] Jonathan Emery. *Beam Wire Scanner Control, Monitoring and Supplies part*. CERN, 2014.

[3] Altera. Arria v soc fpga hard processor system. http://www.altera.com/devices/fpga/arria-fpgas/arria-v/hard-processor-system/arrv-soc-hps.html.

[4] Altera. Hps-fpga bridges. http://www.altera.com/literature/hb/arria-v/av_54005.pdf.

[5] Altera. System interconnect. http://www.altera.com/literature/hb/arria-v/av_54004.pdf.

[6] Altera. Arria v soc development kit and soc embedded design suite. http://www.altera.com/products/devkits/altera/kit-arria-v-soc.html.

[7] Altera. *Altera SoC Embedded Design Suite User Guide*, 2013.

[8] Altera. *Making Qsys Components*, 2012.

[9] RocketBoards. Booting linux using prebuilt sd card image. http://www.rocketboards.org/foswiki/Documentation/GSRD131BootLinuxSd.

[10] RocketBoards. Boot flow. http://www.rocketboards.org/foswiki/Documentation/GSRDBootFlow.

[11] RocketBoards. Generating and compiling the preloader. http://www.rocketboards.org/foswiki/Documentation/GSRD131Preloader.

[12] RocketBoards. Generating the device tree. http://www.rocketboards.org/foswiki/Documentation/GSRD131DeviceTreeGenerator.

[13] Leed Salim. Iperf for arm : cross-compile. http://embetek.blogspot.ch/2012/04/iperf-for-arm-cross-compile.html.

[14] Vipin. Random number generator in vhdl. http://vhdlguru.blogspot.ch/2010/03/random-number-generator-in-vhdl.html.

A Annexes

A.1 Code source bare-metal

```
#include <stdio.h>
#include "alt_clock_manager.h"
#include "alt_fpga_manager.h"
#include "alt_bridge_manager.h"
#include "socal/socal.h"
#include "socal/hps.h"
#include "socal/alt_fpgamgr.h"


ALT_STATUS_CODE test_config_full(void);
ALT_STATUS_CODE test_bridge(void);
void delay(uint32_t delay);


// MAIN
int main(int argc, char** argv)
{
   ALT_STATUS_CODE status = ALT_E_SUCCESS;      // Status of the FPGA

    // Verify power is on
    if (alt_fpga_state_get() == ALT_FPGA_STATE_POWER_OFF)
    {
        printf("ERROR: FPGA Monitor reports FPGA is powered off.\n");
        return 1;
    }

    // Get the control of the FPGA
    printf("INFO: alt_fpga_control_enable().\n");
    status = alt_fpga_control_enable();

    // Tries to program the bitstream into the FPGA
    if (status == ALT_E_SUCCESS)
      status = test_config_full();

    // If the FPGA is under HPS control test the AXI bridges
    if (status == ALT_E_SUCCESS)
      status = test_bridge();

    // Release the control under the AXI bridges
    printf("INFO: alt_fpga_control_disable().\n");
    alt_fpga_control_disable();


    if (status == ALT_E_SUCCESS)
    {
        printf("RESULT: All tests successful.\n");
        return 0;
    }
    else
    {
        printf("RESULT: Some failures detected.\n.");
        return 1;
    }
}


// This function program the bitstream into the FPGA
ALT_STATUS_CODE test_config_full(void)
{
   const uint32_t full_config_retry = 5;

    // Verify the MSELs are appropriate for the type of image we're using.
    ALT_FPGA_CFG_MODE_t mode = alt_fpga_cfg_mode_get();
    switch (mode)
    {
      case ALT_FPGA_CFG_MODE_PP32_FAST_AESOPT_DC:
      case ALT_FPGA_CFG_MODE_PP32_SLOW_AESOPT_DC:
         printf("INFO: MSEL configured correctly for FPGA image.\n");
```

```
69        break;
70
71        default:
72           printf("ERROR: Incompatible MSEL mode set. Cannot continue with FPGA programming.\
                n");
73           return ALT_E_ERROR;
74     }
75
76    // Variables declared in Makefile
77     extern char _binary_soc_system_dc_rbf_start;
78     extern char _binary_soc_system_dc_rbf_end;
79
80     // Use the above symbols to extract the FPGA image information.
81     const char *   fpga_image      = &_binary_soc_system_dc_rbf_start;
82     const uint32_t fpga_image_size = &_binary_soc_system_dc_rbf_end - &
           _binary_soc_system_dc_rbf_start;
83
84     // Trace the FPGA image information.
85     printf("INFO: FPGA Image binary at %p.\n", fpga_image);
86     printf("INFO: FPGA Image size is %u bytes.\n", (unsigned int)fpga_image_size);
87
88     // Try the full configuration a few times.
89     for (uint32_t i = 0; i < full_config_retry; ++i)
90     {
91         ALT_STATUS_CODE status;
92         status = alt_fpga_configure(fpga_image, fpga_image_size);
93
94         if (status == ALT_E_SUCCESS)
95         {
96             printf("INFO: alt_fpga_configure() successful on the %u of %u retry(s).\n", (
                    unsigned int)(i + 1), (unsigned int)full_config_retry);
97             return ALT_E_SUCCESS;
98         }
99     }
100
101    printf("ERROR: FPGA failed to program after %u attempt(s).\n", (unsigned int)
           full_config_retry);
102
103    return ALT_E_ERROR;
104 }
105
106
107 // This function test the usage of the bridges
108 ALT_STATUS_CODE test_bridge(void)
109 {
110    const uint32_t ALT_LW_H2F = 0xFF200000;
111    const uint32_t ALT_H2F = 0xC0000000;
112
113    // Offsets of the peripherals as seen in Qsys
114    const uint32_t ALT_LED_OFFSET = 0x00010040;
115    const uint32_t ALT_ID_OFFSET = 0x00010000;
116    const uint32_t ALT_INV_OFFSET = 0x00020040;
117
118     // Overall status of the test
119     ALT_STATUS_CODE status = ALT_E_SUCCESS;
120
121     // Attempt to bring up the used bridges in the Qsys design
122     ALT_STATUS_CODE status1 = alt_bridge_init(ALT_BRIDGE_H2F,   NULL, NULL);
123     ALT_STATUS_CODE status2 = alt_bridge_init(ALT_BRIDGE_LWH2F, NULL, NULL);
124
125
126     // Verify if the bridges could be initialized
127     if (status1 == ALT_E_SUCCESS)
128         printf("INFO: Bridge H2F successfully initialized.\n");
129     else
130     {
131         printf("ERROR: H2F bridge initialization failed; status = %d.\n", (int)status1);
132         status = ALT_E_ERROR;
133     }
134
135     if (status2 == ALT_E_SUCCESS)
136         printf("INFO: Bridge LWH2F successfully initialized.\n");
137     else
```

```
138     {
139         printf("ERROR: LWH2F bridge initialization failed; status = %d.\n", (int)status2);
140         status = ALT_E_ERROR;
141     }
142 
143     printf("\n");
144 
145 
146     // If all bridges are OK, then test the communication between the HPS and FPGA
147     if (status == ALT_E_SUCCESS)
148     {
149       // Get the ID number from the component System ID
150       uint32_t sysid = alt_read_word(ALT_H2F + ALT_ID_OFFSET);
151       printf("TEST 1: Get System ID (H2F) = 0x%x.\n", (unsigned int)sysid);
152 
153       // Then blink the LEDs 50 times. The LEDs are turned on with the H2F AXI bridge
154       // and turned off with the lightweight H2F bridge
155       printf("TEST 2: Toggle LEDs (Turn on with H2F, turn off with LWH2F)\n");
156 
157       for (uint32_t i = 0; i < 50; i++)
158       {
159          alt_write_word(ALT_H2F + ALT_LED_OFFSET, 0);
160          delay(100000);
161          alt_write_word(ALT_LW_H2F + ALT_LED_OFFSET, 0xF);
162          delay(100000);
163       }
164 
165       // Then send a value to be inverted
166       printf("TEST 3: Send the value 0xAAAAAAAA to be inverted (LWH2F)\n");
167       alt_write_word(ALT_LW_H2F + ALT_INV_OFFSET, 0xAAAAAAAA);
168 
169       // Then read the inverted value
170       printf("TEST 3: Inverted value: 0x%x\n", alt_read_word(ALT_LW_H2F + ALT_INV_OFFSET));
171 
172 
173       // End of tests
174       printf("\n");
175 
176       // When bridges are no more used, you have to close them
177       printf("INFO: Uninitalize H2F bridge.\n");
178       alt_bridge_uninit(ALT_BRIDGE_H2F, NULL, NULL);
179 
180       printf("INFO: Uninitalize LWH2F bridge.\n");
181       alt_bridge_uninit(ALT_BRIDGE_LWH2F, NULL, NULL);
182     }
183 
184     return status;
185 }
186 
187 
188 // A basic function to wait some time
189 void delay(uint32_t delay)
190 {
191    for (uint32_t i = 0; i < delay; i++)
192    {
193       __asm__("nop");
194       __asm__("nop");
195    }
196 }
```

Code 5 – Code C du programme bare-metal

A.2 Liste des librairies

Nom	Compatibilité	License	Prix	Release	Réferences	Commentaire
lwIP	Portage à faire (écriture du driver)	Open source (Modified BSD license)	Gratuit	17.12.12	Altera, Xilinx, Honeywell	Largement répandu, compatible IPv4 ou IPv6, mais pas les deux en même temps.
uIP	Portage à faire (écriture du driver)	Open source (Modified BSD license)	Gratuit	2013	Ciso	Version plus légère de lwIP, mais avec un débit plus faible. Ne gère pas IPv6 tel quel. (Ajout d'un paquet nécessaire)
CrossWorks TCP/IP	Compatible	Propriétaire	Sur demande	Pas spécifié	Pas spécifié	La librairie doit être utilisée avec l'environnement de développement de CrossStudio
NicheStack	Adaptation sur demande	Propriétaire, royalty free	Minimum 10'000 €	Pas spécefié	ABB, AT&T, Honeywell	Parrainé sur le site de ARM. Société spécialisée dans les OS et les stacks IP.
smxNS	Adaptation sur demande	Propriétaire, royalty free	Sur demande	Pas spécifié	GE Healthcare, Omicron	smxNS6 (Un autre de leur modules) doit être ajouté à cette librairie afin d'être compatible avec IPv6.

Tableau 4 – Liste des librairies pouvant être utilisés pour ce projet.

A.3 Liste des RTOS

Nom	Compatibilité	License	Prix	Release	Réferences	Commentaire
FreeRTOS	Portage à faire (écriture du driver)	Open source (Modified BSD license)	Gratuit	2013	Altera, Xilinx, Honeywell	La stack TCP/IP de ce système est basée sur lwIP, d'où la nécessité d'adaptation du driver
Fusion Embedded TCP/IP	Adaptation sur demande	Propriétaire, royalty free	Sur demande	Pas spécifié	ABB, Alstom, Rayathon	Librairie qui fonctionne avec un RTOS quelconque. Compatible avec IPv4 et IPv6. Doit être utilisée RTOS quelconque.
Multicore Abassi	Compatible directement	Propriétaire, royalty free	8'000$	Pas spécefié	ARM, Dolphin Integration, HCC-Embedded	Permet d'utiliser les deux cœurs du Cortex-A9. Parrainé sur le site d'ARM. La stack TCP/IP utilisée est lwIP (ils ont écrit les drivers). La license de base permet l'utilisation sur un type de processeur et compilateur et doit être utilisée pour un seul type de produit.
Nucleus RTOS	Compatible avec Cortex-A9, mais pas directement avec la carte de développement	Pas spécifié	Sur demande	Pas spécifié	ARM, NASA	Produit par Mentor Graphics. 3 milliards de devices utilisent ce RTOS. La stack TCP/IP est compatible IPv4 et IPv6.
µC/OS-II + µC/TCP-IP	Compatible directement	Propriétaire, bloqué pour un produit et un processeur	Sur demande	Pas spécifié	ARM, NASA	La librairie µC/TCP-IP fonctionne avec le RTOS µC/OS-II. La librairie peut fonctionner avec un autre RTOS, mais dans ce cas le SAV n'est pas offert. Compatible IPv4 et IPv6.

Tableau 5 – Liste des RTOS pouvant être utilisés pour ce projet.

A.4 Code source de mesure de vitesse du serveur

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>


void error(const char *msg)
{
   perror(msg);
   exit(EXIT_FAILURE);
}



int main(int argc, char *argv[])
{
   int socketFd, newSocketAdd, socketOptVal = 1, socketOpt;
   unsigned long dataSize, n, i, remainingData, dataAlreadySend = 0;
   char *bufferSend, bufferReceiv[256];
   socklen_t clientAddLen, optLen;
   struct sockaddr_in serverAdd, clientAdd;

   // Check the parameters
   if (argc < 3)
   {
      fprintf(stderr,"USAGE: %s [Port number] [Data size in MB]\n", argv[0]);
      exit(EXIT_FAILURE);
   }

   dataSize = atol(argv[2])*1024*1024;

   // Allocate the good buffer size
   bufferSend = (char*)malloc(dataSize * sizeof(char));

   // Open a new streaming socket (TCP) on internet
   if ((socketFd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      error("ERROR: Cannot opening socket");


   // Get the sending buffer size
   optLen = sizeof(socketOpt);
   if ((getsockopt(socketFd, SOL_SOCKET, SO_SNDBUF, &socketOpt, &optLen)) < 0)
      printf("ERROR: Cannot get buffer size info!\n");
   else
      printf("Old send buffer size: %d\n", socketOpt);

   // Set the new buffer size for the socket
   socketOpt = 102400;
   if ((setsockopt(socketFd, SOL_SOCKET, SO_SNDBUF, &socketOpt, sizeof(socketOpt))) < 0)
      printf("ERROR: Cannot set buffer size!\n");

   // Verify the new sending buffer size
   optLen = sizeof(socketOpt);
   if ((getsockopt(socketFd, SOL_SOCKET, SO_SNDBUF, &socketOpt, &optLen)) < 0)
      printf("ERROR: Cannot get buffer size info!\n");
   else
      printf("New send buffer size: %d\n", socketOpt);


   // Allow the immediate reuse of the socket
   setsockopt(socketFd, SOL_SOCKET, SO_REUSEADDR, &socketOptVal, sizeof(int));

   // Set the desired conection (address, port)
   serverAdd.sin_family = AF_INET;               // Used for TCP packets
   serverAdd.sin_addr.s_addr = INADDR_ANY;          // This server gonna use the local
       address
   serverAdd.sin_port = htons(atoi(argv[1]));       // Get the port number and set the port
       number in network format
```

```
70
71      // Bind the socket
72      if (bind(socketFd, (struct sockaddr*)&serverAdd, sizeof(serverAdd)) < 0)
73          error("ERROR: Cannot bind socket");
74
75      // Listen on the socket
76      listen(socketFd, 1);
77      clientAddLen = sizeof(clientAdd);
78
79      // Accept the new connection (client)
80      if ((newSocketAdd = accept(socketFd, (struct sockaddr*)&clientAdd, &clientAddLen)) < 0)
81          error("ERROR: Cannot accept connection");
82
83      // Read data from socket
84      if ((n = read(newSocketAdd, bufferReceiv, 255)) < 0)
85          error("ERROR: reading from socket");
86
87      // Check if the good request has been received
88      if (strcmp(bufferReceiv, "Ready"))
89          error("ERROR: Wrong request");
90
91      // Send the amount of data to send to client
92      sprintf(bufferSend, "%ld", dataSize);
93
94      if (write(newSocketAdd, bufferSend, strlen(bufferSend) + 1) < 0)  // Dont forget the \0
            char
95          error("ERROR: Cannot send data size to client");
96
97      // Fill buffer with dummy data
98      for (i = 0; i < dataSize; i++)
99          *(bufferSend + i) = (i%127);
100
101     // At the beggining all data has to be send
102     remainingData = dataSize;
103
104     // Loop for sending data
105     while(remainingData > 0)
106     {
107         if ((n = (unsigned long)write(newSocketAdd, bufferSend + dataAlreadySend,
                remainingData)) < 0)
108             error("ERROR: Cannot send packet to client");
109
110         remainingData -= n;
111         dataAlreadySend += n;
112     }
113
114     sleep(1);
115
116     // Close the sockets
117     close(socketFd);
118     close(newSocketAdd);
119
120     free(bufferSend);
121
122     return EXIT_SUCCESS;
123 }
```

Code 6 – Code du programme de mesure de vitesse du serveur.

A.5 Code source de mesure de vitesse du client

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <time.h>
#include <fcntl.h>


void error(const char *msg)
{
   perror(msg);
   exit(EXIT_FAILURE);
}



int main(int argc, char *argv[])
{
   int socketFd, n, socketOpt;
   struct hostent *serverInfo;
   struct sockaddr_in serverAdd;
   char buffer[256], *bufferReceiv;
   unsigned long dataSize, packetNumber = 0, totalData = 0, i;
   struct timespec *tDepart, *tFin; // Vectors for storing the delays
   float totalTime = 0.;
   socklen_t optLen;

   // Open a log file
   FILE *fileLog = fopen("packets.log", "w");
   FILE *fileData = fopen("data.log", "w");

   if (fileLog == NULL || fileData == NULL)
      error("ERROR: Cannot open log file!\n");

   // Check the arguments
   if (argc < 3)
   {
      fprintf(stderr,"USAGE: %s [Server IP address] [Port]\n", argv[0]);
      exit(EXIT_SUCCESS);
   }

   // Open a new socket
   if ((socketFd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      error("ERROR: Cannot open socket!\n");

   // Get the sending buffer size
   optLen = sizeof(socketOpt);
   if ((getsockopt(socketFd, SOL_SOCKET, SO_SNDBUF, &socketOpt, &optLen)) < 0)
      printf("ERROR: Cannot get buffer size info!\n");
   else
      printf("Old send buffer size: %d\n", socketOpt);

   // Set the new buffer size for the socket
   socketOpt = 102400;
   if ((setsockopt(socketFd, SOL_SOCKET, SO_SNDBUF, &socketOpt, sizeof(socketOpt))) < 0)
      printf("ERROR: Cannot set buffer size!\n");

   // Verify the new sending buffer size
   optLen = sizeof(socketOpt);
   if ((getsockopt(socketFd, SOL_SOCKET, SO_SNDBUF, &socketOpt, &optLen)) < 0)
      printf("ERROR: Cannot get buffer size info!\n");
   else
      printf("New send buffer size: %d\n", socketOpt);

   // Get infos about the serverInfo
   if ((serverInfo = gethostbyname(argv[1])) == NULL)
   {
```

```
72          fprintf(stderr,"ERROR: No such host\n");
73          exit(EXIT_FAILURE);
74      }
75
76      // Set the desired connection
77      serverAdd.sin_family = AF_INET;         // Required for a TCP connection
78      bcopy((char*)serverInfo -> h_addr, (char*)&serverAdd.sin_addr.s_addr, serverInfo ->
            h_length);  // Copy the address into serverAdd
79      serverAdd.sin_port = htons(atoi(argv[2]));    // Get the port number
80
81      // Connect to server
82      if (connect(socketFd, (struct sockaddr*)&serverAdd, sizeof(serverAdd)) < 0)
83          error("ERROR: Cannot connect to server");
84
85      // Send request to server to tell that the we're ready to continue
86      if ((write(socketFd, "Ready", 5)) < 0)
87          error("ERROR: Cannot write to socket");
88
89      // Get the size of data that is going to be received
90      if (read(socketFd, buffer, 256) < 0)
91          error("ERROR: Cannot read the data size from server");
92
93      dataSize = (unsigned long)atol(buffer);
94
95      // Allocate the good memory space
96      bufferReceiv = (char*)malloc(dataSize * sizeof(char));
97
98      // Allocate the good number of delays to be stord
99      tDepart = (struct timespec*)malloc(dataSize * sizeof(struct timespec));
100     tFin = (struct timespec*)malloc(dataSize * sizeof(struct timespec));
101
102     // Receive data send by server
103     while(totalData < dataSize)
104     {
105         clock_gettime(CLOCK_REALTIME, (tDepart + packetNumber));
106
107         if ((n = read(socketFd, bufferReceiv + totalData, (dataSize - totalData) * sizeof(
                char))) < 0)
108             error("ERROR: Cannot read data from server");
109
110         clock_gettime(CLOCK_REALTIME, (tFin + packetNumber));
111
112         // Log the results in the file
113         fprintf(fileLog, "Paquet %lu, Taille %d\n", packetNumber, n);
114
115         // Write data into results file
116         for (i = 0; i < (unsigned long)n; i++)
117             fprintf(fileData, "%d\n", *(bufferReceiv + i));
118
119         packetNumber++;
120         totalData += n;
121     }
122
123     fprintf(fileLog, "Finished receiving data from server\nTotal data received %lu\n",
            totalData);
124
125     close(socketFd);  // Close the socket
126
127     // Calculate the total sending time
128     unsigned long diff;
129     for (i = 0; i < packetNumber; i++)
130     {
131         diff = (unsigned long)(((tFin+i) -> tv_nsec - (tDepart+i) -> tv_nsec) + ((tFin+i) ->
                tv_sec - (tDepart+i) -> tv_sec)*1000000000);
132         totalTime += (float)diff;
133     }
134
135     // To get ms
136     totalTime /= 1e6;
137
138     // Print results
139     printf("%ld MB received within %.1f [ms]\n", totalData/1048576, totalTime);
140
```

```
141     // Calculate the throughout
142     printf("Throughout: %.1f [Mbits/s]\n", (float)((totalData/1000000)*8000)/totalTime);
143 
144     // Close the log file
145     fclose(fileLog);
146     fclose(fileData);
147 
148     free(bufferReceiv);
149 
150     return EXIT_SUCCESS;
151 }
```

Code 7 – Code du programme de mesure de vitesse du client.

A.6 Code source du serveur de l'application de démonstration

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/mman.h>

#include "include/hwlib.h"
#include "include/socal/socal.h"
#include "include/socal/hps.h"
#include "include/socal/alt_gpio.h"
#include "include/socal/hps.h"

#include "axi.h"
#include "comm.h"

// DEFINES
#define MEM_BLOCK_SIZE 4587520   // In words of 64 bits (35 [MB])


// File name with the bell curve
const char bellFile[] = "table.out";
const int bellSize = 12000, coeffRand[] = {40, 20, 33, 26};
const int numberRand = 4;


// Global variables
char *sendBuffer;
int socketFd, runs = 0;
uint64_t *randBuffer;


// Error Catching functioin
void error(const char *msg)
{
    perror(msg);
    exit(EXIT_FAILURE);
}



// MAIN
int main(int argc, char* argv[])
{
    int fdMem, commSocket, i;
    FILE *fdBell;
    void *virtualBaseLwAxi, *virtualBaseAxi;
    char tmpStr[20];


    // Argument number
    if (argc < 2)
    {
         fprintf(stderr, "USAGE: %s [Port number]\n", argv[0]);
         return EXIT_FAILURE;
    }

    // Allocate sending buffer
    if ((sendBuffer = (char*)malloc(MEM_BLOCK_SIZE * sizeof(uint64_t))) == NULL)
    {
       fprintf(stderr, "Cannot allocate sendBuffer\n");
         return EXIT_FAILURE;
    }


    if ((randBuffer = (uint64_t*)malloc(MEM_BLOCK_SIZE * sizeof(uint64_t))) == NULL)
    {
```

```
72          fprintf(stderr, "Cannot allocate randBuffer\n");
73          return EXIT_FAILURE;
74      }
75
76
77      // Open file giving access to memory
78      if ((fdMem = open("/dev/mem", (O_RDWR | O_SYNC))) < 0)
79      {
80          fprintf(stderr, "Cannot open /dev/mem file\n");
81          return EXIT_FAILURE;
82      }
83
84      // Open file with the bell curve
85      if ((fdBell = fopen(bellFile, "r")) == NULL)
86      {
87          fprintf(stderr, "Cannot open %s file\n", bellFile);
88          return EXIT_FAILURE;
89      }
90
91      // Map memory into user space
92      virtualBaseLwAxi = mmap(NULL, LW_AXI_SPAN, (PROT_READ | PROT_WRITE), MAP_SHARED, fdMem,
            LW_AXI_BASE);
93      virtualBaseAxi = mmap(NULL, AXI_SPAN, (PROT_READ | PROT_WRITE), MAP_SHARED, fdMem,
            AXI_BASE);
94
95      // Check if the space is remapped
96      if (virtualBaseLwAxi == MAP_FAILED || virtualBaseAxi == MAP_FAILED)
97      {
98          fprintf(stderr, "ERROR: mmap() failed\n");
99          close(fdMem);
100         return EXIT_FAILURE;
101     }
102
103     // Turn OFF leds
104     setLed(virtualBaseLwAxi, 0xFFFF);
105
106     // Check the systemID
107     if (checkSysId(virtualBaseLwAxi, 0xACD51311) == EXIT_FAILURE)
108         return EXIT_FAILURE;
109
110     // Open a new socket and wait for connection
111     if ((commSocket = serverInit(atoi(argv[1]))) == EXIT_FAILURE)
112     {
113         fprintf(stderr, "Cannot open socket\n");
114         return EXIT_FAILURE;
115     }
116
117
118     // Connection loop
119     while(1)
120     {
121         printf("Wait for an arm request...\n");
122         // Check if READY command has come
123         if (receiveCmd(commSocket, "READY") == DECON)
124             break;
125
126         // Read and store random values in RAM
127         printf("Filling RAM with random data...\n");
128
129         for (i = 0; i < MEM_BLOCK_SIZE; i++)
130         {
131             *(randBuffer + i) = readAxi(virtualBaseAxi, RANDOM_GENE_BASE);
132         }
133
134         // Copy values read in RAM to sending buffer
135         memcpy(sendBuffer, randBuffer, MEM_BLOCK_SIZE*sizeof(uint64_t));
136
137         // Add bell curve from file to random values
138         rewind(fdBell);
139
140         for (i = 0; i < bellSize; i++)
141         {
142             fgets(tmpStr, 20, fdBell);
```

```
143             *(sendBuffer + i) = atoi(tmpStr) + (*(sendBuffer + i) / coeffRand[runs %
                    numberRand]);
144         }
145
146         // Send data size to client
147         sendSize(commSocket, MEM_BLOCK_SIZE * sizeof(uint64_t));
148
149         // Wait OK command
150         if (receiveCmd(commSocket, "OK") == DECON)
151             break;
152
153         // Wait for the trigger to begin transmission
154         setLed(virtualBaseLwAxi, 0);
155         printf("Waiting for a trigger signal...\n");
156         while((readAxi(virtualBaseLwAxi, BUTTON_BASE) & 1) > 0);
157         printf("Trigger signal received!\n");
158         setLed(virtualBaseLwAxi, 0xFFFF);
159
160         // Semd trig command
161         sendCmd(commSocket, "TRIG");
162
163         printf("Sending data to client...\n");
164         sendPacket(commSocket, sendBuffer, MEM_BLOCK_SIZE * sizeof(uint64_t));
165         printf("Data sent!\n\n");
166
167         runs++;
168     }
169
170     printf("Finishing program...\n");
171
172     // Clean
173     free(sendBuffer);
174     close(fdMem);
175     fclose(fdBell);
176     close(socketFd);
177     close(commSocket);
178
179     munmap(virtualBaseLwAxi, LW_AXI_SPAN);
180     munmap(virtualBaseAxi, AXI_SPAN);
181
182     return EXIT_SUCCESS;
183 }
```

Code 8 – demo.c

```
#ifndef COMM_H
#define COMM_H

#define DECON 5

int serverInit(int port);
void sendPacket(int sock, char *buffer, int size);
void sendCmd(int sock, char cmd[]);
int receiveCmd(int sock, char cmd[]);
void sendSize(int sock, int size);

#endif
```

Code 9 – comm.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include "comm.h"

extern int socketFd;

int serverInit(int port)
{
   int sock, socketOptVal = 1;
   struct sockaddr_in serverAdd, clientAdd;
   socklen_t clientAddLen;

   // Open TCP socket
   if ((socketFd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      return EXIT_FAILURE;

   // Allow the immediate reuse of the socket
   setsockopt(socketFd, SOL_SOCKET, SO_REUSEADDR, &socketOptVal, sizeof(int));

   // Set desired connection
   serverAdd.sin_family = AF_INET;              // Used for TCP packets
   serverAdd.sin_addr.s_addr = INADDR_ANY;         // This server gonna use the local
       address
   serverAdd.sin_port = htons(port);      // Get the port number and set the port number in
        network format

   // Bind the socket
   if (bind(socketFd, (struct sockaddr*)&serverAdd, sizeof(serverAdd)) < 0)
      return EXIT_FAILURE;

   // Waiting for connection
   printf("Listening on the socket...\n");
   listen(socketFd, 1);
   clientAddLen = sizeof(clientAdd);

   // Accept the new connection (client)
   if ((sock = accept(socketFd, (struct sockaddr*)&clientAdd, &clientAddLen)) < 0)
      return EXIT_FAILURE;

   printf("Client connected!\n\n");

   return sock;
}

void sendPacket(int sock, char *buffer, int size)
{
   int dataAlreadySend = 0, n;

   do
   {
      if ((n = write(sock, buffer + dataAlreadySend, size - dataAlreadySend)) < 0)
      {
         printf("Cannot send packet\n");
         exit(EXIT_FAILURE);
      }

      dataAlreadySend += n;
   } while(dataAlreadySend < size);
}

void sendCmd(int sock, char cmd[])
{
   const int cmdLength = 20;

   int dataAlreadySend = 0, n;
   char bufferTmp[cmdLength];
```

```
72
73      strcpy(bufferTmp, cmd);
74
75      do
76      {
77          if ((n = write(sock, bufferTmp + dataAlreadySend, cmdLength - dataAlreadySend)) < 0)
78          {
79              printf("Cannot write command\n");
80              exit(EXIT_FAILURE);
81          }
82
83          dataAlreadySend += n;
84      } while(dataAlreadySend < cmdLength);
85  }
86
87  int receiveCmd(int sock, char cmd[])
88  {
89      const int cmdLength = 20;
90
91      int iResult, totalRecv = 0;
92      char buffer[cmdLength];
93
94
95      do
96      {
97          iResult = read(sock, buffer + totalRecv, cmdLength - totalRecv);
98
99          if (iResult < 0)
100         {
101             printf("Cannot receive command\n");
102             exit(EXIT_FAILURE);
103         }
104
105         totalRecv += iResult;
106     } while(iResult > 0);
107
108
109     if (strcmp(buffer, "DECON") == 0)
110     {
111         return DECON;
112     }
113     else if (strcmp(buffer, cmd))
114     {
115         printf("Wrong command received (Received: %s)\n", buffer);
116         return 0;
117     }
118     else
119         return 1;
120 }
121
122 void sendSize(int sock, int size)
123 {
124     const int cmdLength = 20;
125
126     int dataAlreadySend = 0, n;
127     char bufferTmp[cmdLength];
128
129     sprintf(bufferTmp, "%d", size);
130
131     do
132     {
133         if ((n = write(sock, bufferTmp + dataAlreadySend, cmdLength - dataAlreadySend)) < 0)
134         {
135             printf("Cannot write command\n");
136             exit(EXIT_FAILURE);
137         }
138
139         dataAlreadySend += n;
140     } while(dataAlreadySend < cmdLength);
141 }
```

Code 10 – comm.c

```c
#ifndef AXI_H
#define AXI_H


// Used for Leigthweigth AXI
#define LW_AXI_BASE (0xff200000)
#define LW_AXI_SPAN (0x200000)
#define LW_AXI_MASK (LW_AXI_SPAN - 1)

// Used for AXI
#define AXI_BASE (0xC0000000)
#define AXI_SPAN (0x3C000000)
#define AXI_MASK (AXI_SPAN - 1)

// SDRAM size to map
#define RAM_SIZE 0x6400000 // 100 [MB] of memory


// Addresses defined in Qsys
#define SYSID_BASE 0x10000
#define RANDOM_GENE_BASE 0x20000
#define RAM_BASE 0x100000//0x10000000
#define BUTTON_BASE 0x100C0


int readLwAxi(void *virtual_base , int addr);
uint64_t readAxi(void *virtual_base , int addr);
void writeAxiLw(void *virtual_base, int addr, int data);
void writeAxi(void *virtual_base, int addr, uint64_t data);

int checkSysId(void *virtual_base, int sysIdRef);
void setLed(void *virtual_base, int value);

#endif
```

Code 11 – axi.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>


#include "include/hwlib.h"
#include "include/socal/socal.h"
#include "include/socal/hps.h"
#include "include/socal/alt_gpio.h"
#include "include/socal/hps.h"

#include "axi.h"

int readLwAxi(void *virtual_base , int addr)
{
   int data;

   data = alt_read_word(virtual_base + ((uint32_t)(LW_AXI_BASE + addr) & (uint32_t)(
       LW_AXI_MASK)));
   return data;
}

uint64_t readAxi(void *virtual_base , int addr)
{
   return alt_read_dword(virtual_base + ((uint32_t)(AXI_BASE + addr) & (uint32_t)(AXI_MASK)
       ));
}

void writeAxiLw(void *virtual_base, int addr, int data)
{
   alt_write_word(virtual_base + ((uint32_t)(LW_AXI_BASE + addr) & (uint32_t)(LW_AXI_MASK))
       , data);
}


void writeAxi(void *virtual_base, int addr, uint64_t data)
{
   alt_write_dword(virtual_base + ((uint32_t)(AXI_BASE + addr) & (uint32_t)(AXI_MASK)),
       data);
}

int checkSysId(void *virtual_base, int sysIdRef)
{
   int sysId;

   if ((sysId = readLwAxi(virtual_base, SYSID_BASE)) != sysIdRef)
   {
      printf("ERROR: SysID is not matching the reference! (0x%X)\n", sysId);
      return EXIT_FAILURE;
   }
   else
      printf("SysID is matching the reference! (0x%X)\n", sysId);

   return sysId;
}

void setLed(void *virtual_base, int value)
{
   const uint32_t ledAddr = 0x10040;

   writeAxiLw(virtual_base, ledAddr, value);
}
```

Code 12 – axi.c

A.7 Histogramme du générateur de nombres psydo-aléatoires

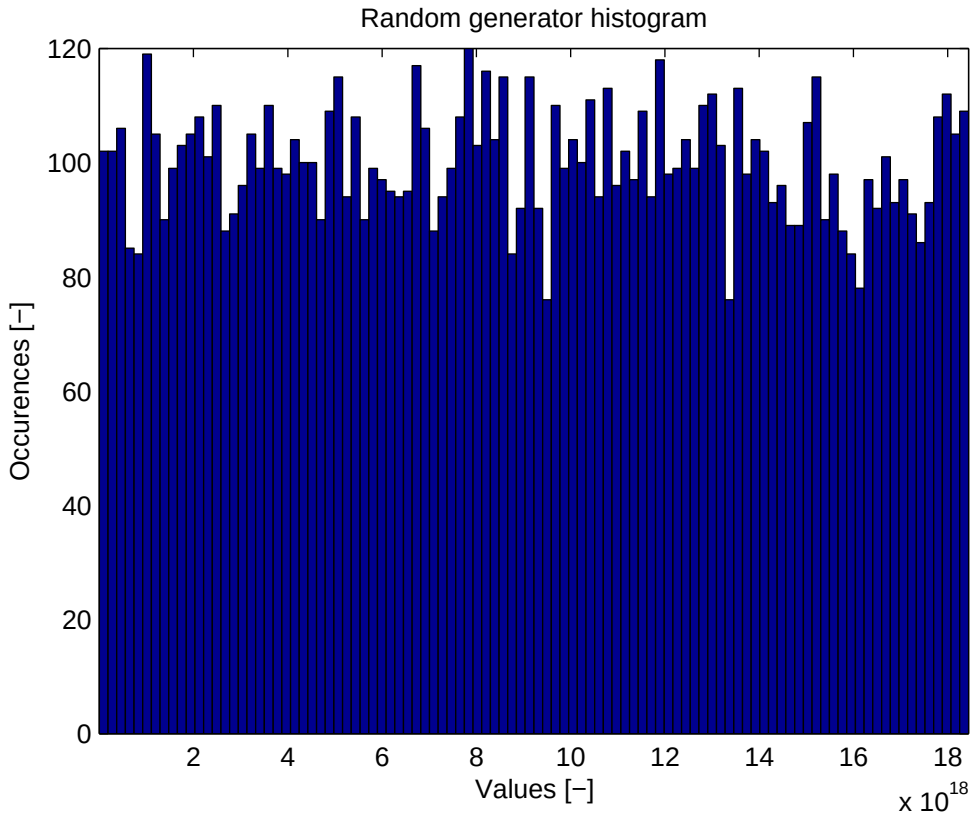

Figure 20 – Histogramme du générateur de nombres psydo-aléatoires sur 10'000 échantillons avec un bin de 100.

A.8 Code source du client de l'application de démonstration

```
#define WIN32_LEAN_AND_MEAN

#include <winsock2.h>
#include <ws2tcpip.h>

#include <stdlib.h>
#include <string.h>

#include <cairo.h>
#include <gtk/gtk.h>

#include "callbacks.h"
#include "comm.h"
#include "graph.h"

#pragma comment(lib, "Ws2_32.lib")  // Need this library for windows


// Global variables
GtkBuilder *builder;
GtkWidget *window;

SOCKET ConnectSocket = INVALID_SOCKET;
struct sockaddr_in clientService;

char *bufferReceiv = NULL;
_tGraph Graph;



int main (int argc, char *argv[])
{
   WSADATA wsaData;  // Used to initialize windows sockets
   const char builderFile[] = "demo.glade";
   GtkWidget *drawingArea;
   GtkSpinner *spinner;

   // Initialize GTK
   gtk_init (&argc, &argv);

   // Initialize static graph options
   Graph.plotReady = FALSE;   // No plot graph ready to graph
   Graph.plotlinked = TRUE;
   Graph.gridWidth = 0.1;
   Graph.plotWidth = 0.5;
   Graph.xBorder = 30.0;
   Graph.yBorder = 30.0;
   Graph.xLabel = "Position [mm]";
   Graph.yLabel = "Beam intensity";


   // Start windows socket
   if (WSAStartup(MAKEWORD(2,2), &wsaData) != NO_ERROR)
   {
      printf("ERROR: Cannot init Windows socket\n");
      return EXIT_FAILURE;
   }


   // Create window from Glade file
   builder = gtk_builder_new ();
   if ((gtk_builder_add_from_file (builder, builderFile, NULL)) == 0)
   {
      fprintf(stderr, "Error adding builder from file %s\n", builderFile);
      return EXIT_FAILURE;
   }


   // Connect signals to callbacks
   gtk_builder_connect_signals (builder, NULL);

```

```
72
73      // Load the main window with all the widgets
74      if ((window = GTK_WIDGET(gtk_builder_get_object(builder, "mainWindow"))) == NULL)
75      {
76          fprintf(stderr, "ERROR: Cannot load GTK main window\n");
77          return EXIT_FAILURE;
78      }
79
80      // Get the draving area
81      drawingArea = GTK_WIDGET(gtk_builder_get_object(builder, "drawingArea"));
82      g_signal_connect(G_OBJECT(drawingArea), "draw", G_CALLBACK(on_draw_event), NULL);
83
84      // Show window
85      gtk_widget_show(window);
86
87      // Main loop
88      gtk_main();
89
90
91      // Close socket
92      closesocket(ConnectSocket);
93      WSACleanup();
94
95      // Close GTK
96      return EXIT_SUCCESS;
97  }
```

Code 13 – main.c

```
#ifndef CALLBACKS_H_INCLUDED
#define CALLBACKS_H_INCLUDED

#include <gtk/gtk.h>

G_MODULE_EXPORT on_window_destroy (GtkWidget *object, gpointer user_data);
G_MODULE_EXPORT on_connectButton_clicked (GtkWidget *object, gpointer user_data);
G_MODULE_EXPORT on_draw_event (GtkWidget *widget, cairo_t *cr, gpointer user_data);
G_MODULE_EXPORT on_buttonOk_clicked(GtkWidget *object, gpointer user_data);
G_MODULE_EXPORT on_addressAdjustment_value_changed(GtkWidget *object, gpointer user_data);


#endif // CALLBACKS_H_INCLUDED
```

Code 14 – callbacks.h

```
#define WIN32_LEAN_AND_MEAN

#include <winsock2.h>
#include <ws2tcpip.h>

#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>

#include <cairo.h>
#include <gtk/gtk.h>

#include "various.h"
#include "callbacks.h"
#include "comm.h"
#include "graph.h"


extern GtkBuilder *builder;

extern SOCKET ConnectSocket;
extern struct sockaddr_in clientService;

extern char *bufferReceiv;
extern _tGraph Graph;

extern GtkWidget *window;

double *bufferReceivDouble, *bufferReceivDoubleOld;
int dataSize;

gboolean connected = FALSE;


// Number of points that forms the bell curve
const int numberBellPoints = 12000;

// Fermeture de la fenetre
G_MODULE_EXPORT on_window_destroy (GtkWidget *object, gpointer user_data)
{
   // Send deconnexion packet if necessary
   if (connected == TRUE)
      sendCmd(ConnectSocket, "DECON");

   free(bufferReceivDouble);
   free(bufferReceivDoubleOld);

   gtk_main_quit();
}


// Click sur le boutton Connect/Disconnect
G_MODULE_EXPORT on_connectButton_clicked (GtkWidget *object, gpointer user_data)
{
   gchar *addressStr, *portStr, tmpStr[60];
   int i = 0, port;
   char tempBuff[20];

   // Default label message
   gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "connexionStatusLabel")),
       "Not connected to a server");

   // Get the value from the address entry box
   addressStr = gtk_entry_get_text(GTK_ENTRY(gtk_builder_get_object(builder, "addressEntry"
       )));
   portStr = gtk_entry_get_text(GTK_ENTRY(gtk_builder_get_object(builder, "portEntry")));

   if (connected) // Want to disconnect
   {
      // Send disconect command
      sendCmd(ConnectSocket, "DECON");

```

```
72          // Disconnect from server
73          closesocket(ConnectSocket);
74
75          // Allow IP and port modification
76          gtk_widget_unset_state_flags(GTK_WIDGET(gtk_builder_get_object(builder, "addressEntry
                ")), GTK_STATE_FLAG_INSENSITIVE);
77          gtk_widget_unset_state_flags(GTK_WIDGET(gtk_builder_get_object(builder, "portEntry"))
                , GTK_STATE_FLAG_INSENSITIVE);
78
79          // Change button label
80          gtk_button_set_label(GTK_BUTTON(gtk_builder_get_object(builder, "connectButton")), "
                Connect");
81
82          // Change label text
83          gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "connexionStatusLabel")
                ), "<span foreground=\"black\">Not connected to a server</span>");
84
85          free(bufferReceivDouble);
86          connected = FALSE;
87      }
88      else  // Try to connect to server
89      {
90          // Buffer for data plot
91          if ((bufferReceivDouble = (double*)malloc(numberBellPoints * sizeof(double))) == NULL
                )
92          {
93              sprintf(tempBuff, "Error: Cannot allocate %d bytes of data for the
                    bufferReceivDouble!", numberBellPoints * sizeof(double));
94              showErrorPopup(window, "Allocation error", tempBuff);
95          }
96
97                 // Buffer for data plot
98          if ((bufferReceivDoubleOld = (double*)malloc(numberBellPoints * sizeof(double))) ==
                NULL)
99          {
100             sprintf(tempBuff, "Error: Cannot allocate %d bytes of data for the
                    bufferReceivDouble!", numberBellPoints * sizeof(double));
101             showErrorPopup(window, "Allocation error", tempBuff);
102         }
103
104
105         // Converts the entered address into numerical format
106         if (inet_addr(addressStr) == INADDR_NONE)
107         {
108             gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "
                    connexionStatusLabel")), "<span foreground=\"red\">Wrong IP format!</span>");
109             return;
110         }
111
112         // Verify if theres no letters in port
113         while (*(portStr + i) != '\0')
114         {
115             if (!isdigit(*(portStr + i)))
116             {
117                 gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "
                        connexionStatusLabel")), "<span foreground=\"red\">Wrong port provided!</
                        span>");
118                 return;
119             }
120
121             i++;
122         }
123
124         port = strtol(portStr, NULL, 10);
125
126         // Verify if port is in the bounds
127         if (port < 1 || port > 65535)
128         {
129             gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "
                    connexionStatusLabel")), "<span foreground=\"red\">No port number or port
                    number out of bounds!</span>");
130             return;
131         }
```

```
132
133
134         // Create socket
135         ConnectSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
136         if (ConnectSocket == INVALID_SOCKET)
137         {
138             WSACleanup();
139             showErrorPopup(window, "Socket error", "Error: Cannot create valid socket!");
140         }
141
142
143         // Parametres du serveur
144         clientService.sin_family = AF_INET;
145         clientService.sin_addr.s_addr = inet_addr(addressStr);
146         clientService.sin_port = htons(port);
147
148         // Connection au serveur
149         if (connect(ConnectSocket, (SOCKADDR*)&clientService, sizeof(clientService)) ==
                SOCKET_ERROR)
150         {
151             showWarningPopup(window, "Server error", "Cannot connect to the server, are you
                    sure that server is running?");
152         }
153         else
154         {
155             // Disable Address and Port entries
156             gtk_widget_set_state_flags(GTK_WIDGET(gtk_builder_get_object(builder, "
                    addressEntry")), GTK_STATE_FLAG_INSENSITIVE, FALSE);
157             gtk_widget_set_state_flags(GTK_WIDGET(gtk_builder_get_object(builder, "portEntry")
                    ), GTK_STATE_FLAG_INSENSITIVE, FALSE);
158
159             // Enable scan frame
160             gtk_widget_unset_state_flags(GTK_WIDGET(gtk_builder_get_object(builder, "scanFrame
                    ")), GTK_STATE_FLAG_INSENSITIVE);
161
162             // Change button label
163             gtk_button_set_label(GTK_BUTTON(gtk_builder_get_object(builder, "connectButton")),
                     "Disconnect");
164
165             sprintf(tmpStr, "<span foreground=\"dark green\">Connected to %s</span>",
                    addressStr);
166             gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "
                    connexionStatusLabel")), tmpStr);
167             connected = TRUE;
168         }
169     }
170 }
171
172
173 // Genere pour afficher le graphe
174 G_MODULE_EXPORT on_draw_event (GtkWidget *widget, cairo_t *cr, gpointer user_data)
175 {
176     Graph.cr = cr;
177     Graph.xArea = gtk_widget_get_allocated_width (widget);
178     Graph.yArea = gtk_widget_get_allocated_height (widget);
179
180     drawGraph(Graph);
181 }
182
183
184 // Click sur le boutton Arme
185 G_MODULE_EXPORT on_armButton_clicked(GtkWidget *object, gpointer user_data)
186 {
187     gchar tempBuff[100];
188     int i;
189
190     static int scanNumber = 0;
191     clock_t tBegin, tEnd;
192
193     // Send READY command to server
194     sendCmd(ConnectSocket, "READY");
195
196     // Wait for the data to be transmitted
```

```
197     dataSize = receiveSize(ConnectSocket);
198
199     // Allocate receive buffer
200     if ((bufferReceiv = (char*)malloc(dataSize * sizeof(char))) == NULL)
201     {
202         sprintf(tempBuff, "Error: Cannot allocate %d bytes of data for the bufferReceiv!",
                dataSize * sizeof(char));
203         showErrorPopup(window, "Allocation error", tempBuff);
204     }
205
206     // Send OK command
207     sendCmd(ConnectSocket, "OK");
208
209     // Enable address visualisation spin
210     gtk_widget_unset_state_flags(GTK_WIDGET(gtk_builder_get_object(builder, "addressSpin")),
            GTK_STATE_FLAG_INSENSITIVE);
211
212
213     // Wait for TRIG command
214     receiveCmd(ConnectSocket, "TRIG");
215
216     // Receive data from server
217     tBegin = clock();
218     receivePacket(ConnectSocket, bufferReceiv, dataSize);
219     tEnd = clock();
220
221     for (i = 0; i < numberBellPoints; i++)
222         *(bufferReceivDouble + i) = (double)((unsigned char)*(bufferReceiv + i));
223
224
225     // In order to plot correctly data change the values
226     Graph.maxVal = getMax(bufferReceivDouble, numberBellPoints);
227     Graph.yEveryPts = Graph.maxVal / 10;
228     Graph.xEveryPts = numberBellPoints / 10;
229     Graph.dataSize = numberBellPoints;
230     Graph.dataTab = bufferReceivDouble;
231     Graph.plotReady = TRUE;
232
233     scanNumber++;
234
235
236     // Displays information labels
237     // Scan number
238     sprintf(tempBuff, "Scan number: %d", scanNumber);
239     gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "scanNumberLabel")),
            tempBuff);
240
241     // Packet size
242     sprintf(tempBuff, "Received packet: %d [kB]", dataSize/1000);
243     gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "receivedDataLabel")),
            tempBuff);
244
245     // Calculate sending time
246     float receivingTime = (float)(tEnd - tBegin) / (float)CLOCKS_PER_SEC;   // in ms
247     sprintf(tempBuff, "Receiving time: %.0f [ms]", receivingTime * 1e3);
248     gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "receivingTimeLabel")),
            tempBuff);
249
250     // Calculate bandwith
251     sprintf(tempBuff, "Data bandwidth: %.1f [Mbit/s]", (float)(dataSize * 8) / (
            receivingTime * 1e6));
252     gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "bandwidthLabel")),
            tempBuff);
253
254
255     // Display first data value in label
256     sprintf(tempBuff, "%d", (unsigned char)*(bufferReceiv));
257     gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "addressLabel")), tempBuff
            );
258 }
259
260
261 // Called when amount of data is changed
```

```
262 G_MODULE_EXPORT on_addressAdjustment_value_changed(GtkWidget *object, gpointer user_data)
263 {
264    gdouble value;
265    char tempBuff[30];
266 
267 
268    // Sets the upper value for the up down box
269    gtk_adjustment_set_upper(GTK_ADJUSTMENT(gtk_builder_get_object(builder, "
           addressAdjustment")), dataSize);
270 
271 
272    // Get value from the up down box
273    value = gtk_adjustment_get_value(GTK_ADJUSTMENT(gtk_builder_get_object(builder, "
           addressAdjustment")));
274 
275    // Check if there is data avaible
276    if (bufferReceiv != NULL)
277    {
278       sprintf(tempBuff, "%d", (unsigned char)*(bufferReceiv + (int)value));
279       gtk_label_set_label(GTK_LABEL(gtk_builder_get_object(builder, "addressLabel")),
              tempBuff);
280    }
281 }
```

Code 15 – callbacks.c

```
#ifndef COMM
#define COMM

// Prototypes
int receivePacket(SOCKET sock, char *buffer, int size);
void sendCmd(SOCKET sock, char cmd[]);
int receiveCmd(SOCKET sock, char *buffer);
int receiveSize(SOCKET sock);


#endif
```

Code 16 – comm.h

```
#define WIN32_LEAN_AND_MEAN

#include <winsock2.h>
#include <ws2tcpip.h>

#include <stdlib.h>
#include <string.h>

#include <gtk/gtk.h>

#pragma comment(lib, "Ws2_32.lib")  // Need this library for windows


int receivePacket(SOCKET sock, char *buffer, int size)
{
   int iResult, totalRecv = 0;

   do
   {
      gtk_main_iteration_do(FALSE);

      iResult = recv(sock, buffer + totalRecv, size - totalRecv, 0);

      if (iResult > 0)
         totalRecv += iResult;
      else if (iResult == 0)
         g_print("Connection closed\n");
      else
         g_error("Receive failed: %d\n", WSAGetLastError());

   } while(totalRecv < size);

   return totalRecv;
}


void sendCmd(SOCKET sock, char cmd[])
{
   const int cmdLength = 20;
   int dataAlreadySend = 0, iResult;
   char bufferTmp[cmdLength];

   strcpy(bufferTmp, cmd);

   do
   {
      iResult = send(sock, bufferTmp + dataAlreadySend, cmdLength - dataAlreadySend, 0);
      if (iResult == SOCKET_ERROR)
      {
         g_error("Cannot write command\n");
         closesocket(sock);
         WSACleanup();
         exit(EXIT_FAILURE);
      }

      dataAlreadySend += iResult;
   } while(dataAlreadySend < cmdLength);
}


int receiveCmd(SOCKET sock, char *buffer)
{
   const int cmdLength = 20;

   int iResult, totalRecv = 0;
   char bufferTmp[cmdLength];

   do
   {
      gtk_main_iteration_do(FALSE);

      iResult = recv(sock, bufferTmp + totalRecv, cmdLength - totalRecv, 0);

```

```
74          if (iResult > 0)
75             totalRecv += iResult;
76          else if (iResult == 0)
77             g_print("Connection closed\n");
78          else
79             g_error("Cannot receive command: %d\n", WSAGetLastError());
80
81       } while(totalRecv < cmdLength);
82
83
84       // Check the command received
85       if (strcmp(buffer, bufferTmp))
86       {
87          g_warning("Wrong command received (Received: %s)\n", buffer);
88          return 0;
89       }
90       else
91          return 1;
92    }
93
94
95    int receiveSize(SOCKET sock)
96    {
97       const int cmdLength = 20;
98
99       int iResult, totalRecv = 0;
100      char bufferTmp[cmdLength];
101
102      do
103      {
104         iResult = recv(sock, bufferTmp + totalRecv, cmdLength - totalRecv, 0);
105
106         if (iResult > 0)
107            totalRecv += iResult;
108         else if (iResult == 0)
109            g_print("Connection closed\n");
110         else
111            g_error("Cannot receive command: %d\n", WSAGetLastError());
112
113      } while(totalRecv < cmdLength);
114
115
116      // Return the size
117      return atoi(bufferTmp);
118   }
```

Code 17 – comm.c

```
#ifndef VARIOUS_H_INCLUDED
#define VARIOUS_H_INCLUDED

double getMax(double tab[], int sizeTab);
void showWarningPopup(GtkWidget *parent, gchar title[], gchar msg[]);
void showErrorPopup(GtkWidget *parent, gchar title[], gchar msg[]);

#endif // VARIOUS_H_INCLUDED
```

Code 18 – various.h

```c
#include <stdlib.h>
#include <gtk/gtk.h>
#include "various.h"


double getMax(double tab[], int sizeTab)
{
   int i;
   double valMax = tab[0];

   for (i = 1; i < sizeTab; i++)
   {
      if (tab[i] > valMax)
         valMax = tab[i];
   }

   return valMax;
}


void showWarningPopup(GtkWidget *parent, gchar title[], gchar msg[])
{
   GtkWidget *dialog = gtk_message_dialog_new(GTK_WINDOW(parent), GTK_DIALOG_MODAL,
       GTK_MESSAGE_WARNING, GTK_BUTTONS_OK, msg);

   gtk_window_set_title(GTK_WINDOW(dialog), title);
   gtk_dialog_run(GTK_DIALOG(dialog));

   gtk_widget_destroy(dialog);
}


void showErrorPopup(GtkWidget *parent, gchar title[], gchar msg[])
{
   GtkWidget *dialog;
   dialog = gtk_message_dialog_new(GTK_WINDOW(parent), GTK_DIALOG_MODAL, GTK_MESSAGE_ERROR,
        GTK_BUTTONS_CLOSE, msg);

   gtk_window_set_title(GTK_WINDOW(dialog), title);
   gtk_dialog_run(GTK_DIALOG(dialog));

   gtk_widget_destroy(dialog);
   exit(1);
}
```

Code 19 – various.c

```c
#ifndef AXIS_H_INCLUDED
#define AXIS_H_INCLUDED

#include <cairo.h>
#include <gtk/gtk.h>

#define HOR 1
#define VER 0

#define RED 3
#define GREEN 4

typedef struct
{
    cairo_t *cr;          // Cairo draw area

    gboolean plotReady;   // TRUE when data can be plotted
    gboolean plotlinked; // TRUE if dots should be linked

    double xArea;         // X working area (Border included)
    double yArea;         // Y working area (Border included)
    double xBorder;       // X border
    double yBorder;       // Y border

    double gridWidth;     // Grid width (Dots)
    double plotWidth;     // Plot trace width
    double xEveryPts;     // Number of every points to place a line on X
    double yEveryPts;     // Number of every points to place a line on Y

    int dataSize;         // Size of the dataTab
    double maxVal;        // Maximum value in the dataTab
    double *dataTab;      // Data to be plot on graph

    char *xLabel;         // Axis label X
    char *yLabel;         // Axis label Y
}_tGraph;


void drawGraph(_tGraph Graph);


// DO NOT USE THESE FUNCTIONS
static void plot(cairo_t *cr, double valTable[], int tabSize, double xArea, double yArea,
    double xBorder, double yBorder, double dotWidth, gboolean linked, int color);
static void drawAxis(cairo_t *cr, double xArea, double yArea, double xBorder, double
    yBorder);
static void drawArrow(cairo_t *cr, double xBase, double yBase, double h, double l, gboolean
     horizontal);
static void drawGrid(cairo_t *cr, double xArea, double yArea, double xBorder, double
    yBorder, double xEveryPts, double yEveryPts, double maxVal, int tabSize);
static void insertLabels(cairo_t *cr, char xLabel[], char yLabel[], double xArea, double
    yArea, double xBorder, double yBorder);

#endif // AXIS_H_INCLUDED
```

Code 20 – graph.h

```c
#include <cairo.h>
#include <gtk/gtk.h>
#include <math.h>

#include "graph.h"


void drawGraph(_tGraph Graph)
{
   drawAxis(Graph.cr, Graph.xArea, Graph.yArea, Graph.xBorder, Graph.yBorder);

   if (Graph.plotReady)
   {
      drawGrid(Graph.cr, Graph.xArea, Graph.yArea, Graph.xBorder, Graph.yBorder, Graph.
          xEveryPts, Graph.yEveryPts, Graph.maxVal, Graph.dataSize);

      plot(Graph.cr, Graph.dataTab, Graph.dataSize, Graph.xArea, Graph.yArea, Graph.xBorder
          , Graph.yBorder, Graph.plotWidth, Graph.plotlinked, RED);

      insertLabels(Graph.cr, Graph.xLabel, Graph.yLabel, Graph.xArea, Graph.yArea, Graph.
          xBorder, Graph.yBorder);
   }
}


static void drawGrid(   cairo_t *cr,
                double xArea,
                double yArea,
                double xBorder,
                double yBorder,
                double xEveryPts,
                double yEveryPts,
                double maxVal,
                int tabSize)
{
   const double dashPattern[] = {3.0};
   const int dashPatternLen = 1;

   double xPixPerPts, yPixPerPts, tabMax;

   int xLines, yLines, i;
   double posGrid;




   // Line width
   cairo_set_line_width(cr, 0.1);

   // X points per pixel
   xPixPerPts = (xArea - 2.0 * xBorder) / (double)(tabSize - 1);
   yPixPerPts = (yArea - 2.0 * yBorder) / maxVal;
   xLines = tabSize / xEveryPts;    // Number of X lines on grid
   yLines = maxVal / yEveryPts;

   // Draw grid in X
   for (i = 1; i < xLines; i++)
   {
      cairo_move_to(cr, xPixPerPts * xEveryPts * i + xBorder, yBorder);
      cairo_rel_line_to(cr, 0., yArea - 2.0*yBorder);
   }

   // Draw grid in Y
   for (i = 1; i < yLines; i++)
   {
      cairo_move_to(cr, xBorder, yPixPerPts * yEveryPts * i + yBorder);
      cairo_rel_line_to(cr, xArea - 2.0 * xBorder, 0.);
   }

   cairo_stroke(cr);
   cairo_set_dash(cr, NULL, 0, 0);
}

```

```
71 static void drawAxis(cairo_t *cr, double xArea, double yArea, double xBorder, double
       yBorder)
72 {
73    cairo_set_source_rgb(cr, 0, 0, 0);
74    cairo_set_line_width(cr, 2.0);
75
76    cairo_move_to(cr, xBorder, yBorder);
77    cairo_line_to(cr, xBorder, yArea - yBorder);
78    cairo_rel_line_to(cr, xArea - 2.0*xBorder , 0);
79
80    cairo_stroke(cr);
81
82    drawArrow(cr, xBorder, yBorder, 10.0, 5.0, VER);
83    drawArrow(cr, xArea - xBorder, yArea - yBorder, 10.0, 5.0, HOR);
84 }
85
86 static void drawArrow(cairo_t *cr, double xBase, double yBase, double h, double l, gboolean
       horizontal)
87 {
88    if (!horizontal)
89    {
90       // Fleche haut
91       cairo_move_to(cr, xBase - l, yBase);
92       cairo_line_to(cr, xBase, yBase - h);
93       cairo_line_to(cr, xBase + l, yBase);
94    }
95    else
96    {
97       cairo_move_to(cr, xBase, yBase - l);
98       cairo_line_to(cr, xBase + h, yBase);
99       cairo_line_to(cr, xBase, yBase + l);
100   }
101
102   cairo_fill(cr);
103   cairo_stroke(cr);
104 }
105
106
107 static void plot( cairo_t *cr,
108             double valTable[],
109             int tabSize,
110             double xArea,
111             double yArea,
112             double xBorder,
113             double yBorder,
114             double dotWidth,
115             gboolean linked,
116             int color)
117 {
118    int i;
119    double xPixSpace = (xArea - 2.0 * xBorder) / (double)(tabSize - 1), yPixSpace;
120    double yOffset = yArea - yBorder;
121    double angle = 2.0*M_PI;
122    double tabMax = valTable[0];
123
124
125    // Search the max in array
126    for (i = 1; i < tabSize; i++)
127    {
128       if (valTable[i] > tabMax)
129          tabMax = valTable[i];
130    }
131
132    // Plot the new plot
133    yPixSpace = (yArea - 2.0 * yBorder) / tabMax;
134
135    if (color == RED)
136       cairo_set_source_rgb(cr, 255, 0, 0);
137    else
138       cairo_set_source_rgb(cr, 0, 255, 0);
139
140    cairo_set_line_width(cr, 1.0);
141
```

```
142
143     // Points
144     for (i = 0; i < tabSize; i++)
145     {
146         cairo_arc(cr, xBorder + ((double)i * xPixSpace), yOffset - ((double)valTable[i] *
                yPixSpace), dotWidth, 0.0, angle);
147         cairo_fill(cr);
148         cairo_stroke(cr);
149     }
150
151     // Lines if needed
152     if (linked)
153     {
154         for (i = 0; i < tabSize-1; i++)
155         {
156             cairo_move_to(cr, xBorder + ((double)i * xPixSpace), yOffset - ((double)valTable[i
                    ] * yPixSpace));
157             cairo_line_to(cr, xBorder + ((double)(i + 1) * xPixSpace), yOffset - ((double)
                    valTable[i + 1] * yPixSpace));
158             cairo_stroke(cr);
159         }
160     }
161 }
162
163
164 static void insertLabels(cairo_t *cr, char xLabel[], char yLabel[], double xArea, double
        yArea, double xBorder, double yBorder)
165 {
166     cairo_text_extents_t extents;
167
168     cairo_set_source_rgb(cr, 0, 0, 0);
169     cairo_select_font_face (cr, "Sans", CAIRO_FONT_SLANT_NORMAL, CAIRO_FONT_WEIGHT_NORMAL);
170
171     // Xlabel
172     cairo_text_extents(cr, xLabel, &extents); // Used to center text
173     cairo_move_to(cr, (xArea / 2.0) - (extents.width/2 + extents.x_bearing), (yArea -
            yBorder / 2.0) - (extents.height/2 + extents.y_bearing));
174     cairo_set_font_size(cr, 13);
175     cairo_show_text(cr, xLabel);
176
177     cairo_stroke(cr);
178
179     // Ylabel
180     cairo_text_extents (cr, yLabel, &extents);   // Used to center text
181     cairo_move_to(cr, (xBorder / 2.0) - (extents.height/2 + extents.y_bearing), (yArea /
            2.0) + (extents.width/2 + extents.x_bearing));
182     cairo_rotate(cr, -M_PI/2.0);
183     cairo_set_font_size(cr, 13);
184     cairo_show_text(cr, yLabel);
185
186     cairo_stroke(cr);
187 }
```

Code 21 – graph.c