MACHEREZ Alexis

Année 2019

Rapport de PFE

Détection de menaces IOT sur FPGA

POLYTECH Lille

Boulevard Paul Langevin

Cité Scientifique

59655 VILLENEUVE D'ASCQ

Tuteurs : Alexandre BOE

Thomas VANTROYS

Sommaire

Introduction

L'IOT (Internet Of Things) est une technologie grandissante. C'est un concept orienté microservices, où une multitude de petits objets réalisant des tâches simples (capteurs, activateurs, ...) sont connectés par ondes radio entre eux ou à une passerelle vers le Cloud, dans lequel les données sont processées et des décisions sont prises. De nombreux domaines d'applications existent actuellement en industrie, dans le médical ou même chez les particuliers.

Les appareils utilisés pour l'internet des objets sont généralement très modulaires afin d'être utilisés pour différentes applications et de pouvoir s'appairer facilement à un réseau. Ils ont aussi des contraintes au niveau de la durée de vie de leur batterie et embarque donc très peu de puissance de calcul.

Ces deux points évoqués ci-dessus rendent donc les réseaux IOT relativement vulnérables aux attaques. En effet, la faible puissance de calcul des objets les rend incapables de se défendre en cas d'attaques ni d'ailleurs de les détecter. De plus l'interopérabilité des objets induit des vulnérabilités au sein même du réseau.

Il serait donc intéressant de mettre en place des moyens capables d'apporter plus de sécurité dans les réseaux IOT.

1. Présentation du projet de fin d'étude

Comme énoncé dans l'introduction, l'internet des objets est sensibles aux attaques. Ce projet a donc pour objectif de proposer un système permettant la surveillance d'un réseau IOT. Ce système ne sera pas en mesure de détecter toutes les menaces et attaques possible, il se concentrera uniquement sur celles qui sont le plus répandues.

Dans un premier temps, il s'agira donc de faire un état des lieux concernant les menaces existantes dans un réseau IOT en fonction des différentes couche du modèle OSI. Le but étant de dresser une liste de ces menaces à partir de laquelle il sera possible d'identifier quelles sont les menaces qui pourront être pris en charge par le système.

Le système proposé est constitué d'un processeur embarquant un OS de type Linux, couplé à un FPGA qui réalisera la majeure partie des calculs.

Il faudra donc commencer par étudier les moyens et méthodes à mettre en place afin de faire fonctionner le CPU et le FPGA en adéquation. Entre autre, le CPU devra être capable de communiquer avec le FPGA (et réciproquement, le FPGA avec le CPU), puisque la partie FPGA sera pilotée directement depuis le Linux embarqué. Il faut aussi prévoir un système d'interruptions qui permettra au FPGA d'envoyer des avertissements au CPU, ces interruptions devront donc être gérées dans Linux.

Ensuite, la logique sera mise en place dans le FPGA pour détecter les menaces identifiées dans la première partie du projet.

Les logiciels utilisés sont Vivado, Xilinx SDK et Petalinux.
Vivado et Xilinx SDK sont des environnements de développement tandis que Petalinux est un outil en ligne de commande.

Vivado est utilisé pour réaliser le code de la partie logique (VHDL ou Verilog) et les diagramme (ou block design) décrivant le matériel.

Xilinx SDK sert à créer des applications C ou C++ destinées à fonctionner sans OS. Il peut aussi générer des image bootable (encore une fois, sans OS) contenant les applications et le bitstream du FPGA.

Enfin, Petalinux permet de configurer et de compiler des images de type Linux et de générer des images bootable contenant l'image Linux ainsi que le bitstream.

2. Etat de l'art des menaces possibles

a. Liste non exhaustive

Cette liste d'attaques ne recense pas toutes les attaques possibles, elle liste celles qui sont couramment utilisées dans le cas d'un réseau IOT.

Couche Applicative :

Injection de code malicieux par buffer overflow : Cette attaque est réalisée en parvenant à faire écrire un processus dans une plage mémoire qui ne lui est pas allouée. Des instructions peuvent alors y être placées pour exécuter des opérations non désirées. C'est une attaque peu utilisée à cause de sa complexité.

Mot de passe par défaut : Ce n'est pas une attaque à proprement parlé, mais cela peut arriver fréquemment. Les logins permettant d'accéder au root du shell d'un objet peuvent être root / root ou admin / admin, compromettant alors grandement la sécurité.

Bloquer la mise en veille : Consiste à envoyer périodiquement des signaux destinés à maintenir les objets éveiller, épuisant donc leur batterie bien plus rapidement.

Envoi de données corrompues : Consiste à corrompre les données envoyées par un objets du réseau.

Couche de Session, Transport, Réseau et Liaison:

Déni de service (DOS) : L'attaquant utilise une machine (ou plusieurs dans le cas du DDOS) avec laquelle il va envoyer un grand nombre de requêtes dans le but de submerger le serveur, voire de le faire planter. Cette attaque est très répandue.

Homme du milieu (MITM) : Comme son nom l'indique, cette attaque consiste à intercepter les requêtes entre un client et un serveur tout en continuant à les transmettre à leur destinataire. L'attaquant est donc invisible du point de vu du serveur ou du client. C'est une méthode très courante qui n'a d'utilité uniquement si les données ne sont pas encryptées.

Spoofing : Un attaquant qui dispose de suffisamment d'informations peut copier l'identité d'une machine (avec son adresse MAC par exemple), il obtient alors un accès au réseau.

Utilisation d'ARP : Les paquets ARP permettent à une machine de se connecter à un router. Il existe une attaque qui utilise ces paquets afin d'obtenir des informations sur le réseau, voire un accès. L'attaque est utilisé principalement pour ensuite faire du spoofing ou MITM.

Sniffing : Consiste à écouter le trafic et l'analyser, les informations obtenues peuvent ensuite être utilisées pour réaliser l'une des attaques citées ci-dessus.

Couche Physique :

Brouillage : C'est une attaque très utilisée et simple à réaliser. Il suffit d'émettre un signal de forte puissance à la même fréquence que celui que l'on veut brouiller pour gêner ou interrompre des communications hertziennes.

Sniffing : Il est aussi possible d'écouter directement le trafic sur la couche physique plutôt que sur les couches supérieures.

b. Menaces retenues

Compte tenu de la liste précédente, sur la couche applicative, l'attaque de blocage de la mise en veille des objets pourra être détectée en vérifiant le niveau de batterie des objets. La vérification des données reçues peut aussi être faite pour détecter la corruption de données.

Sur la couche physique, le brouillage de fréquence pourra être détecté en analysant le spectre des signaux reçus.

Sur les couches intermédiaires, le système se concentrera principalement sur l'attaque de déni de service.

3. OS Petalinux sur Zynq

Le FPGA utilisé pour ce projet est un Zynq-7000 monté sur une carte Zybo de Digilent. Distribué par Xilinx, ce processeur dispose d'une partie CPU ainsi qu'une partie FPGA, comme indiqué sur le diagramme suivant :

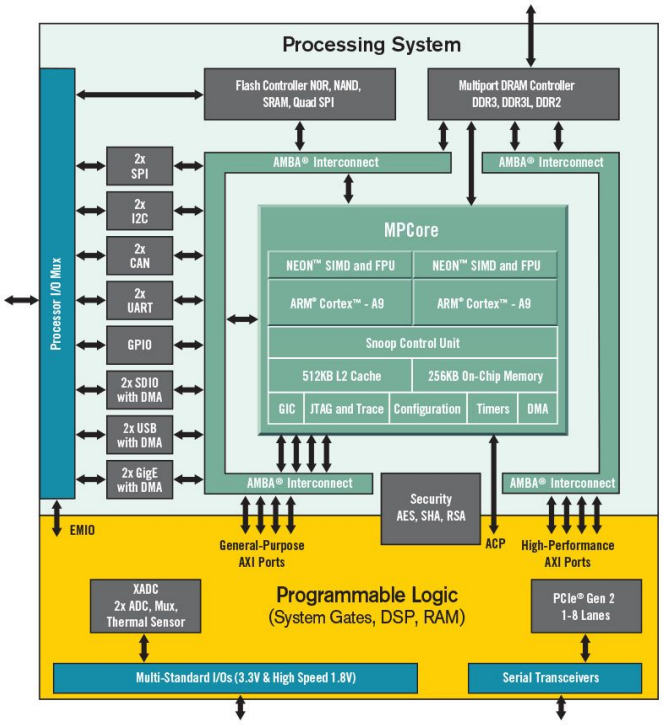


diagramme du processeur de type Zynq-7000

La partie CPU est constitué de deux coeurs ARM Cortex 9, Petalinux servira notamment d'ordonnancer les processus sur ces derniers.
D'autre part, on remarque les ports AXI (dans la partie logique) qui permettent au CPU et au FPGA de communiquer ensemble par le biais de la RAM.

Dans cette partie du rapport, je vais expliquer les recherches et démarches réalisées dans le but de piloter le FPGA depuis le CPU.

a. Présentation de Petalinux

Petalinux désigne un OS conçu pour fonctionner sur le type de processeur qui est utilisé pour le projet, il désigne aussi l'outil en ligne de commande (CLI) permettant de configurer et de compiler des images Linux. L'OS et le CLI sont tous deux maintenus par Xilinx. L'outil est lourd en terme de capacités, et est donc difficile à prendre en main.

Utilisé en combinaison d'un projet Vivado, Petalinux permet de décrire la partie matériel au Linux embarqué grâce à un device tree. Un device tree est un fichier (ou un ensemble de fichiers) détaillant les différentes parties matériel au kernel afin qu'il puisse les utiliser. On y retrouve par exemple, l'identité du matériel, son adresse mémoire, la taille de sa plage mémoire, la configuration de ses interruptions et la compatibilité, par exemple :

```
{
        amba_pl: amba_pl {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "simple-bus";
                ranges ;
                mymodule_0: mymodule@43c00000 {
                        compatible = "xlnx,myint-1.0";
                        interrupt-parent = <&intc>;
                        interrupts = <0 29 4>;
                        reg = <0x43c00000 0x10000>;
                        xlnx,s00-axi-addr-width = <0x4>;
                        xlnx,s00-axi-data-width = <0x20>;
                };
        };
};
```

Sans ce device tree, Linux serait incapable de communiquer avec le FPGA.

L'outil permet aussi de générer le First Stage Boot Loader (FSBL, qui peut d'ailleur aussi être généré dans Vivado). Le FSBL est un binaire exécuté à la mise sous tension du processeur, il va alors charger en mémoire les programmes faisant fonctionner le kernel et envoyer le bitstream au FPGA pour le configurer.

En plus de ces deux fonctionnalités importantes, Petalinux peut générer une image bootable, configurer le rootfs et le kernel, compiler et ajouter des applications (c, c++ et python) et des modules kernel dans le système de fichier de l'image.

Dans mon cas, l'image est bootée depuis une carte micro sd. Cette dernière doit comporter une partition FAT32 bootable d'au moins 1 Go, et et une seconde partition EXT4 d'au moins 3 Go pour accueillir le système de fichier. Pour réaliser le formatage des outils tels que fdisk sur Linux ou diskpart sur Windows peuvent être utilisés. La partition boot contiendra BOOT.BIN et image.ub (décrivant le système de fichiers), générés par Petalinux.

Ci-dessous, une listes des commandes proposées par Petalinux qui sont à retenir.

- Création d'un projet :

```
$ petalinux-create -t project --template zynq --name mypetalinux
```

Crée un nouveau projet Petalinux dans le dossier mypetalinux qui servira à créer une image pour un processeur Zynq.

- Configuration :

```
$ petalinux-config
$ petalinux-config -c rootfs
$ petalinux-config -c kernel
```

Permettent de configurer le projet, par exemple supprimer des applications, ajouter des librairies ou modifier le kernel.

- Import de la partie matériel :

```
$ petalinux-config --get-hw-description wrapper.hdf
```

Génère le device tree de base (qui peut ensuite être modifié à notre convenance) à partir du wrapper.hdf exporter depuis un projet Vivado.

- Création de modules :

```
$ petalinux-create -t modules --name mymodule --enable
$ petalinux-create -t apps --template c --name myapp --enable
```

Crée un module kernel ou une application, l'option enable configure le rootfs pour l'inclure.

- Nettoyage :

```
$ petalinux-build -x mrproper
$ petalinux-build -x distclean
```

La première supprime les fichiers temporaires et le caches, tandis que la seconde réécrit les chemins des fichiers.

- Compilation :

```
$ petalinux-build
```

Compile le kernel Linux, le rootfs, le devicetree et le bootloader.

- Création de l'image :

```
$ petalinux-package --boot --format BIN --fsbl fsbl.elf \
  --fpga bitstream.bit --u-boot --boot-device sd
```

Génère le fichier BOOT.BIN

b. Protocole AXI

AXI est un protocole de communication adopté par Xilinx facilitant la communication entre les différents modules FPGA (appelés IP) et entre le CPU et le FPGA. Les bus de communication sont donc standardisés et améliore l'interopérabilité et la portabilité des IPs.

Il existe trois sous instances de ce protocol : AXI-lite, AXI-full et AXI-stream.

AXI-lite et AXI-full sont très similaires, AXI-full a une plus grande bande passante. Ils utilisent tous les deux le principe de memory mapping (que nous allons voir par la suite) qui permet d'allouer des registres en mémoire pour écrire et lire la données.

AXI-stream apporte en plus la possibilité de streamer de la données en continu, et ne réserve pas de plage mémoire.

Dans ce projet, j'ai utiliser les protocoles AXI-lite et AXI-stream. Il ne m'a pas été nécessaire d'utiliser AXI-full.

Ces trois protocoles disposent d'interfaces maître / esclave implémentant plusieurs signaux pour assurer le bon fonctionnement.

Dans le cas d'AXI-lite et d'AXI-full, l'interface maître indique à l'interface slave l'adresse de la plage mémoire à laquelle il veut accéder avant d'écrire ou de lire dedans. Dans le cas d'une écriture, l'interface esclave envoie un signal confirmant l'écriture à l'interface maître. Il y a plusieurs échanges de signaux entre le maître et l'esclave pour synchroniser les deux et ainsi éviter un overflow ou la perte de données.

Dans le cas d'AXI-stream, les interfaces maître et esclave implémentent quatres signaux principaux, t_ready, t_valid, t_data et t_last. Lorsque l'esclave est prêt à recevoir le stream de données, t_ready est envoyé au maître. Le maître envoie alors le stream de données dans t_data et fixe t_valid à 1 pour indiquer à l'esclave que le transfert est en cours, à la fin du transfert, t_last est envoyé. L'utilisation de t_last n'est pas obligatoire puisque l'interface esclave peut être configuré de sorte à attendre un nombre de données prédéfini.

Deux IPs associés à ces protocoles sont très importants dans Vivado.

Le premier est AXI-Interconnect, il permet d'interconnecter les interfaces AXI. Je l'ai notamment utilisé pour exposer les interfaces du FPGA au CPU, permettant alors la communication entre les deux.

Le deuxième est AXI-Direct-Memory-Access, je l'ai principalement utiliser pour faire fonctionner le protocol AXI-stream entre le CPU et le FPGA.

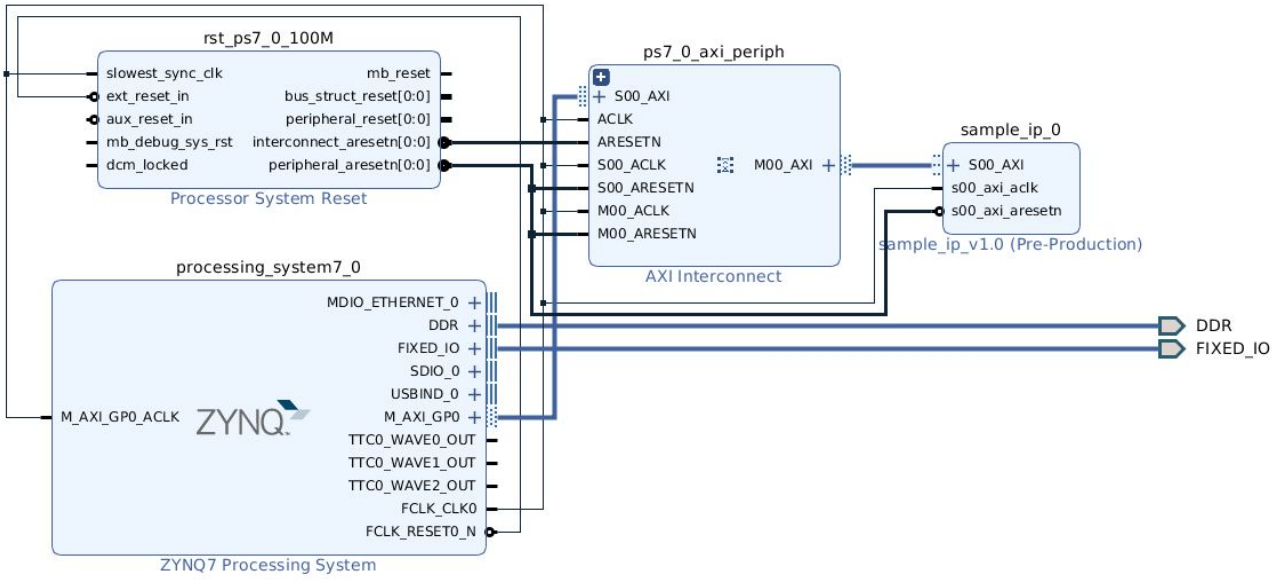
Vivado propose aussi la création d'IP "custom" avec des interfaces AXI et AXI-stream. Créer un IP est utile, puisque cet IP instancie une partie du code et permet donc de mieux organiser son projet et de mieux le visualiser sur les block designs.

c. Memory Mapping

Le memory mapping constitue la base des protocoles AXI-lite et AXI-full. En effet, chaque connection entre deux interfaces AXI réserve une plage mémoire qu'elle pourra utiliser pour échanger les données entre le maître et l'esclave.

Je l'ai utilisé uniquement pour communiquer entre le CPU et le FPGA, puisque pour les communications internes au FPGA j'ai beaucoup utilisé le protocol AXI-stream.

Pour comprendre le fonctionnement, j'ai commencé par réaliser un projet simple dans Vivado afin de pouvoir faire des tests de communications entre le CPU et le FPGA dont le block design est le suivant :



block design : exemple memory mapping

Dans la partie FPGA, un simple compteur a été mis en place avec deux registres de 32 bits à dispositions, il incrémente la valeur du deuxième registre à chaque front montant de l'horloge si la valeur du premier registre est égale à $FFFFFFFF_{hex}$. L'IP AXI-interconnect est aussi utilisé pour lier le CPU au module qu'il pilote.

Pour lancer le compteur, il faut donc écrire dans le premier registre depuis le CPU, la manière la plus directe est d'utiliser la RAM virtuelle en ouvrant le fichier /dev/mem, puis d'obtenir un pointeur vers la plage d'adresse réservé par le fpga (ici par l'IP AXI-Interconnect) avec la fonction mmap puis d'écrire à l'adresse voulue :

```
int fd = fopen("/dev/mem", O_RDWR);
void* ptr = mmap(NULL, 0x10000, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
ptr = *((unsigned *)(ptr + 0x43C00000)) = 0xFFFFFFFF;
unsigned int counter = *((unsigned *)(ptr + 0x43C00004));
```

0x43C00000 et 0x43C00004 étant les adresses des registres.

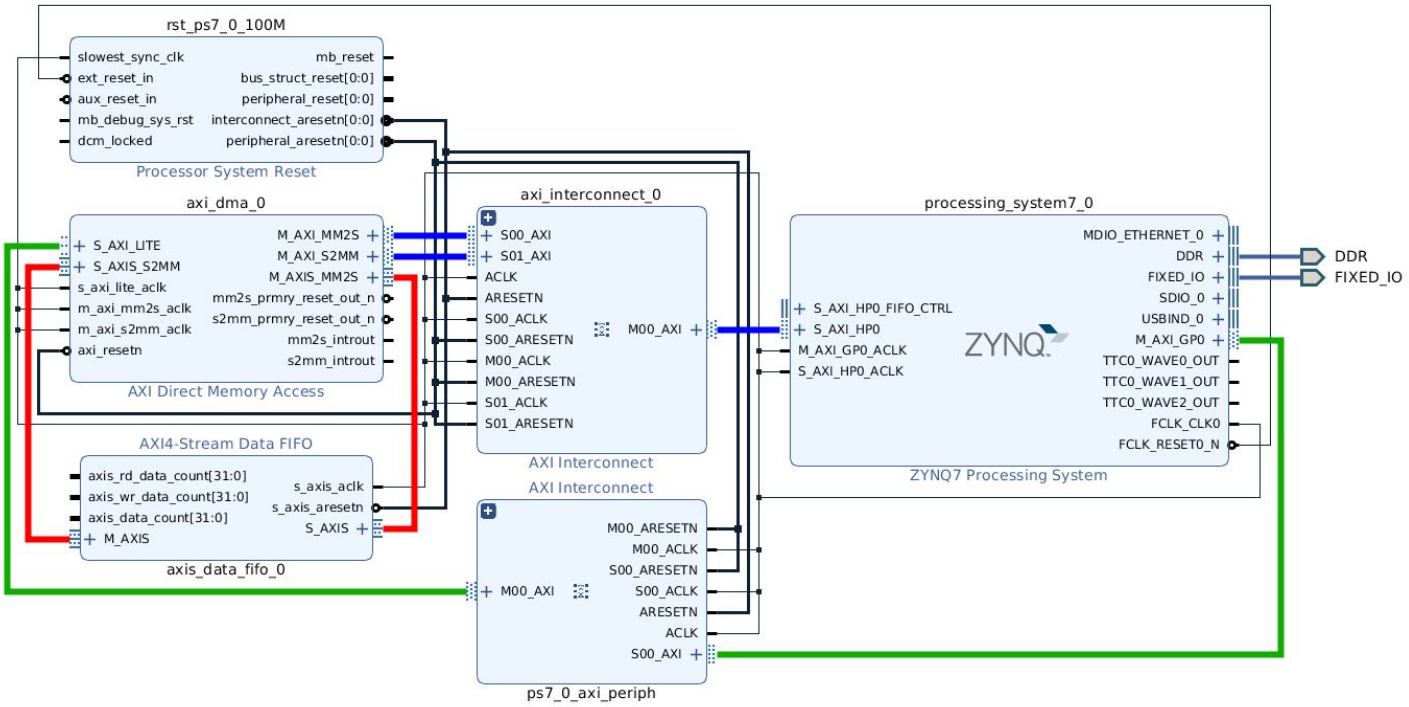
Après avoir écrit dans le premier registre, le deuxième s'incrémente comme prévu. J'ai donc pu vérifier que le FPGA recevait bien les ordres du CPU. Pour tester le code, j'ai créé une application avec le CLI Petalinux que j'ai ensuite exécutée depuis le shell de la Zybo.

Néanmoins, cette méthode exige d'avoir le privilège root pour pouvoir ouvrir le fichier de RAM virtuelle, ce n'est donc pas la plus recommandée.

d. Direct Memory Access (DMA)

Pour streamer des données entre le FPGA et le CPU, le memory mapping ne suffit pas. Il faut utiliser la méthode DMA, qui autorise un matériel (dans mon cas le FPGA) à utiliser directement la RAM indépendamment du CPU.

Pour ce faire j'ai utiliser l'IP AXI Direct Memory Access, évoqué plus haut. Grossièrement, le CPU va écrire de la donnée en RAM qui sera ensuite utilisée par l'IP pour l'envoyer en AXI-stream. Dans le but de tester cette méthode, j'ai créé un projet Vivado utilisant l'IP AXI-DMA, dont voici le block design :



block design : exemple direct access memory

Cette fois, il faut utiliser deux IPs AXI-Interconnect. L'un est utilisé par le CPU pour piloter l'IP AXI-DMA (lien vert). L'autre est utilisé par l'IP AXI-DMA afin d'utiliser la RAM (lien bleu) pour lire les données et les envoyer en stream, puis écrire les données reçues.

Pour cet exemple, j'ai connecté la sortie du stream directement à l'entrée (lien rouge), avec un buffer pour améliorer la stabilité.

Pour faire fonctionner l'AXI-DMA, j'ai aussi utilisé la RAM virtuelle avec mmap pour cette fois obtenir trois pointeurs, le premier pointe vers l'adresse de l'interface esclave de l'AXI-DMA, les deux autres pointes vers la donnée à envoyer et la donnée qui sera reçu. Grâce aux registres de contrôle de son interface esclave, j'indique à l'AXI-DMA les adresses de source et de destination. Ensuite je démarre le stream, puis la taille des données qui va transiter dans l'AXI-stream (puisque je n'utilise pas t_last). J'attend la fin du transfert en vérifiant les registres de status. Enfin je vérifie que la donnée reçue correspond à la donnée envoyée afin de confirmer le bon fonctionnement. Le détail du code peut être trouvé dans l'annexe n°1.

Il faut faire attention à la taille de transfert que l'on indique à l'AXI-DMA. Si le transfert est plus long que la taille indiquée, puisque que t_last n'est ici pas utilisé, l'IP reste bloqué en erreur. Le même phénomène est observé si la plage de mémoire de destination allouée n'est pas assez grande pour accueillir la donnée du stream.

Pour tester le code dans un OS Petalinux, j'ai configuré le kernel pour autoriser l'allocation de mémoire contiguë DMA, cela permet à l'AXI-DMA d'utiliser de la mémoire contiguë et d'être plus performant.

e. UIO drivers

Afin d'éviter d'utiliser le fichier de RAM virtuelle, il est possible de décrire le matériel par un UIO driver (Userspace I/O driver). Les registres du matériel seront alors accessibles depuis Linux par /dev/uio.

Pour tester un UIO driver, j'ai repris le projet utilisé pour l'exemple de memory mapping.

J'ai configurer le kernel de Petalinux pour y autoriser "Userspace I/O platform driver with generic IRQ handling" et "Userspace platform driver with generic irq and dynamic memory", ce qui permet d'accéder à la mémoire du matériel par /dev/uio ainsi que de gérer les interruptions émises.

Il faut aussi modifier le device tree en ajoutant dans le noeud correspondant au matériel l'option : compatible="generic-uio".

Dans le shell de la Zybo on peut alors retrouver le fichier /dev/uio0. Le numéro associé à l'UIO driver (ici 0) correspond à la place du noeud du matériel dans le device tree.

Pour utiliser l'UIO driver, je l'ouvre avec la fonction open, puis j'obtiens le pointeur vers la mémoire du matériel avec mmap :

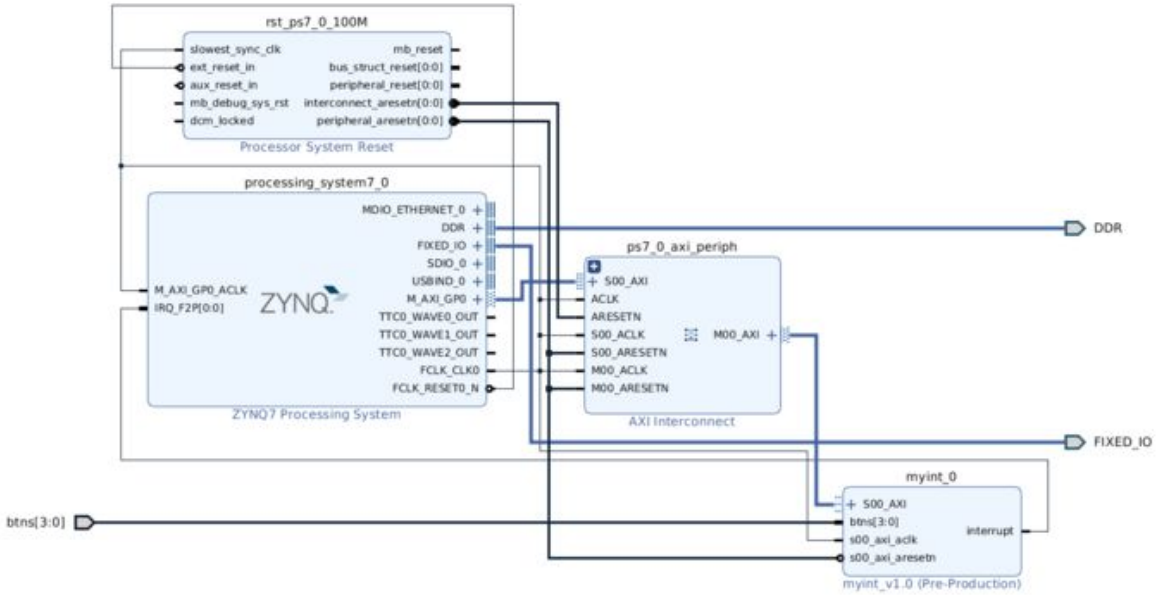
```
int uiofd = open("/dev/uio0", O_RDWR);
void* ptr = mmap(NULL, 0x10000, PROT_READ|PROT_WRITE, MAP_SHARED, uiofd, 0);
ptr = *((unsigned *)(ptr + 0x43C00000)) = 0xFFFFFFFF;
unsigned int counter = *((unsigned *)(ptr + 0x43C00004));
```

Encore une fois, le compteur s'incrémente après avoir écrit dans le premier registre, prouvant que l'envoi d'ordre au FPGA par l'intermédiaire d'un UIO driver fonctionne.

f. Interruptions

Les interruptions provenant du FPGA vers le CPU sont importantes dans ce projet, puisqu'elles vont permettre au FPGA de "prévenir" le CPU en cas d'anomalie.

J'ai tester deux méthodes pour réaliser ce genre d'interruptions. Dans les deux cas, j'ai utiliser le même projet Vivado :



block design : exemple d'interruption

Le module FPGA myint envoie une interruption au CPU (port IRQ_F2), à chaque fois qu'un des boutons de la carte Zybo est pressé.

La première méthode utilise un module kernel (son code peut être trouvé en annexe n°2), qui réserve un port d'interruption est lui associe un handler.

Pour l'utiliser j'ai créer un module avec le CLI Petalinux pour qu'il soit accessible dans le système de fichier de la Zybo.

Il faut aussi modifier le device tree afin de décrire l'interruption dans le noeud du matériel : interrupts = <0 29 1>.

Le deuxième flag, 29, correspond au port d'interruption utilisé. En effet, l'entrée IRQ_FP2 (sur le block design) est un bus [15:0] mappé sur [91:84] et [68:61] dans le CPU. On peut voir sur le block design qu'on utilise IRQ_FP2[0:0], donc mappé sur 61 auquel il faut soustraire 32 (convention de syntax d'un device tree, il faut soustraire 16 si le premier flag est 1). Le troisième flag correspond au type d'interruption, ici c'est un front montant.

Cette fois ci, il ne faut surtout pas ajouter l'option compatible="generic-uio", cette dernière entrerait en conflit avec le module kernel.

Pour exécuter le module depuis le shell de la Zybo, j'ai utiliser la commande insmod. Si il n'y a pas d'erreur, le code renvoie une confirmation et l'interruption devrait visible dans /proc/interrupts. J'ai ensuite pu vérifier le bon traitement des interruptions en essayant d'appuyer sur les boutons de la carte, il y a bien le retour du module kernel, de plus le nombre d'interruptions indiqué dans /proc/interrupts s'incrémente.

La deuxième méthode utilise un UIO driver, il faut donc cette fois ajouter l'option compatible="generic-uio" dans le device tree. Les interruptions reçus par l'intermédiaire d'un UIO driver sont gérées par une boucle while :

```
uint32_t info = 1;
ssize_t flag;

int uiofd = open("/dev/uio0", O_RDWR);

while (CONTINU) {
   // enable interruption
   write(uiofd, &info, sizeof(info));
   // look for interrupt
   flag = read(uiofd, &info, sizeof(info));
   if (flag == (ssize_t)sizeof(info)) {
       printf("Interrupt: %u\n\r",info);
   }
}
```

Comme vu avec la première méthode, la ligne l'interruption est bien enregistrée dans /proc/interrupts (cette fois ci par le device tree et non par un module kernel), et une interruption est bien reçue à chaque fois qu'un bouton est pressé.

Cependant, cette deuxième méthode ne permet de gérer qu'un seul port d'interruption par UIO, même si je n'ai pas eu besoin de gérer plusieurs lignes d'interruptions provenant d'un même module FPGA, il est intéressant de le noter.

4. Modules FPGA

L'intérêt du projet et du type de CPU utilisé, est de pouvoir déléguer la majeure partie des calculs au FPGA. C'est pourquoi les modules de détection de comportements anormaux ont été réalisés dans la partie logique du Zynq.

Pour ce projet, j'ai réalisé trois modules.

Le premier module servira à récupérer des valeurs transmises par les objets IOT (par exemple son niveau de batterie ou la valeur d'un capteur). Si l'une donnée dépasse un certain seuil, le CPU est averti.

Le deuxième module réalisera une analyse spectrale par FFT pour détecter les signaux suspects.

Le dernier module est constitué d'un analyseur de trame IP, récupérant les informations importantes des paquets telles que le protocol, les adresses et les ports. Le but est de pouvoir déceler un trafic suspect.

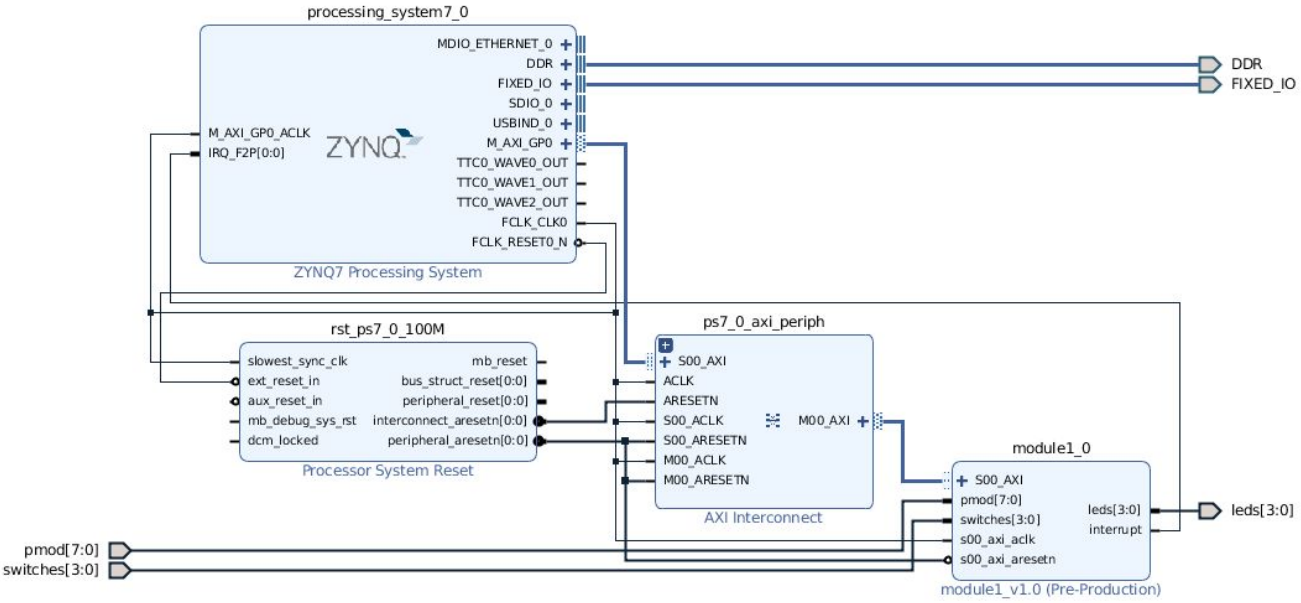
a. Détection de dépassement de seuil

La détection de dépassement de seuil peut être utilisée pour détecter les deux menaces identifiées sur la couche applicative : le blocage du mode veille et l'envoi de donnée corrompues.

Ce module FPGA est constitué d'un seul IP doté d'une interface AXI-lite esclave et de plusieurs registres de 32 bits. Pour chaque donnée à surveiller, deux registres sont alloués, un pour définir le seuil et l'autre pour stocker la valeur. En plus de ceux ci, un dernier registre permet d'indiquer au CPU quelle donnée a dépassé le seuil en cas d'interruption.

Pour les tests, J'ai ajouté un registre de plus qui définie la source des données (switches ou pmod). De plus les leds présentent sur la carte s'éclairent en fonction des quatres derniers bits du registres contenant la valeur à surveiller, me permettant de pouvoir contrôler la valeur présente dans le registre (sachant qu'il n'y a que quatres switches, donc je ne peux modifier que les quatres derniers bits).

Le block design est peu complexe :



block design : module de détection de dépassement de seuil

La logique instanciée (sans le détail de l'interface AXI) du module FPGA peut être trouvé en annexe n°3. Ici, elle est très simple puisque le module surveille une seule donnée. La valeur du seuil est fixée par le CPU qui écrit dans le registre nommé threshold_0, tandis que la valeur à l'instant t de la donnée est écrite dans le registre value_0 par le FPGA. Dans le cas où il y aurait plusieurs données à surveiller, il faudrait allouer plus de registres que nécessaire, et le CPU donnerait l'ordre au FPGA d'activer ou non certains canaux.

Pour contrôler le module, j'utilise une application C dans Petalinux (son code est en annexe n°4). L'application est destinée à tourner en background puisqu'elle utilise des boucles infinies. Au début du processus, le device driver est ouvert et un pointeur vers sa plage mémoire est obtenu. La source des données (ici les switches) et le seuil sont envoyés au FPGA, ensuite un fichier est ouvert qui servira à sauvegarder les données au format csv. Avant le début des boucles infinies, deux handler sont initialisées pour intercepter les signaux SIGINT et SIGTERM, des signaux d'interruptions de processus correspondant respectivement à Ctrl+C et kill (à ne pas confondre à kill -9, qui envoie SIGKILL). Ces handlers permettront au processus de se terminer gracieusement. Enfin le processus est divisé en deux processus (avec fork) contenant les boucles infinies, la première boucle intercepte les interruptions envoyées par le FPGA, tandis que la seconde récupère les valeurs captées par le FPGA toutes les 3 secondes et les écrit dans le fichier csv.

b. Analyse du spectre par FFT

L'analyse spectrale permet de détecter un brouillage ou des émissions suspectes.

Ce module utilise un IP de Xilinx afin de réaliser une analyse spectrale, il retourne au CPU la bande passante à -3 dB ainsi que la puissance maximale, cela permettra de détecter des signaux suspects dans le réseau.

L'IP effectuant la FFT est très complet et dispose de nombreuses configurations possibles. Dans mon cas, l'IP est configuré pour recevoir 1024 points de 64 bits en entrée contenant la partie réelle et la partie imaginaire du signal au format flottant ; en sortie, je récupère aussi 1024 points de 64 bits avec la partie réelle et la partie imaginaire du résultat de la FFT au format flottant. Il est possible de réaliser une FFT avec un signal réel pure (les parties imaginaires sont alors fixées à 0), le spectre obtenu est alors symétrique.

L'architecture algorithmique utilisé pour le calcul est Radix-4. elle réalise les calculs étape par étape contrairement à l'architecture Pipelined, qui comme son nom l'indique réalise les calculs en mode pipeline. L'architecture pipelined est plus performante, mais je n'ai pas pu l'utiliser car la Zybo ne dispose pas d'assez de BRAM.

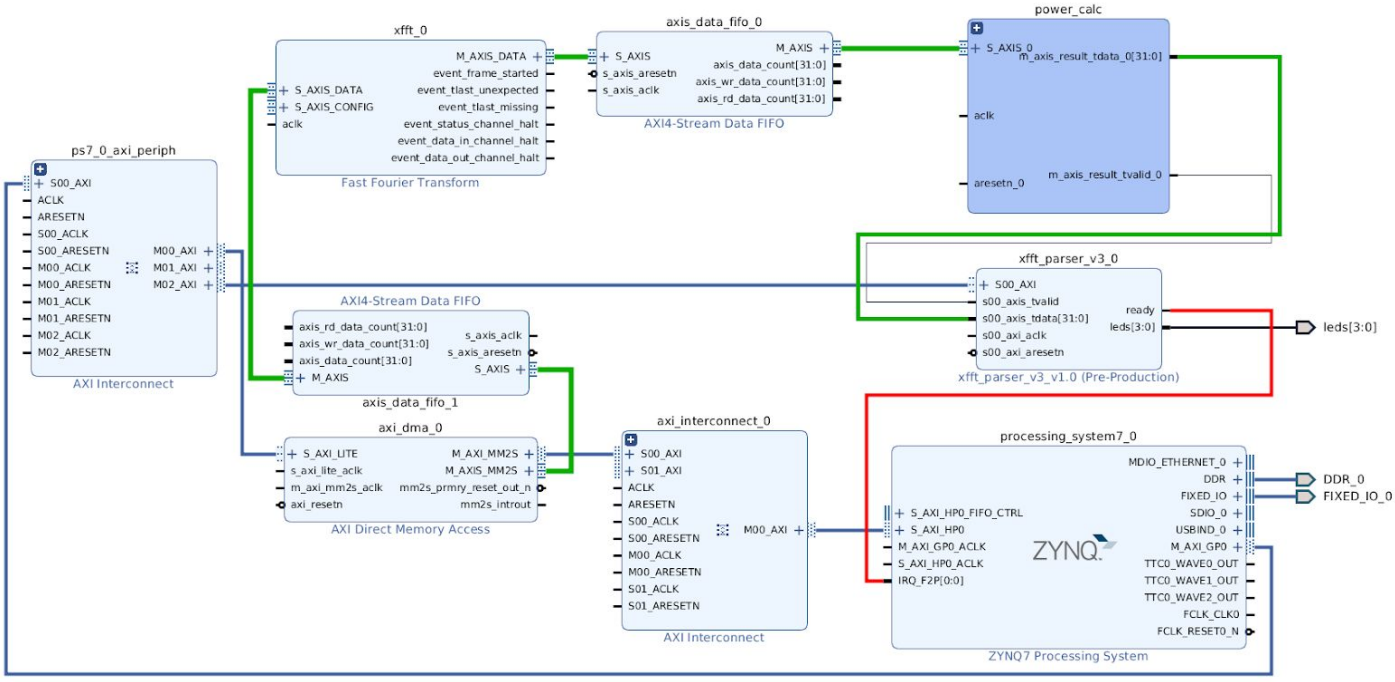
Pour tester le module, j'ai utilisé un code en C (en annexe n°5) permettant de générer les données complexes d'un sinus, qui seront par la suite envoyées à l'IP FFT. L'entrée et la sortie du bloc FFT sont des interfaces AXI-stream, pour le piloter depuis le CPU j'ai donc utilisé l'IP AXI-DMA. J'ai aussi ajouté (en entrée et en sortie de la FFT) des buffer car j'obtenais parfois des résultats incohérent, ces buffers permettent de rendre les bus AXI-stream plus stable.

Le block design et le code C utilisés pour ce test sont sensiblement les mêmes que ceux utilisés pour tester l'AXI-DMA. Les données créées à partir du générateur de sinus sont stockées dans une structure constituée de 1024 fois deux flottant de 32 bits avant d'être envoyées au bloc FFT en AXI-stream par l'intermédiaire de l'AXI-DMA. De la même manière, les données reçues sont enregistrées dans un fichier que j'ai ensuite pu copier depuis la Zybo sur la carte sd puis sur ma machine dans le but de les vérifier.

J'avais généré une sinusoïde de 443 Mhz avec un taux d'échantillonnage de 2 Ghz, le pic de puissance devrait donc se trouver à l'index $433.10^6 \times 1024 \div 2.10^9 \simeq 222$, ce qui est bien le cas.

J’avais aussi ajouté un bout de code permettant de configurer le bloc FFT. Cependant, dans mon cas, la seule configuration possible était le sens de la FFT (directe ou inverse) que je voulais fixer à directe, étant la configuration par défaut du bloc, je n’ai finalement pas utiliser ce code.

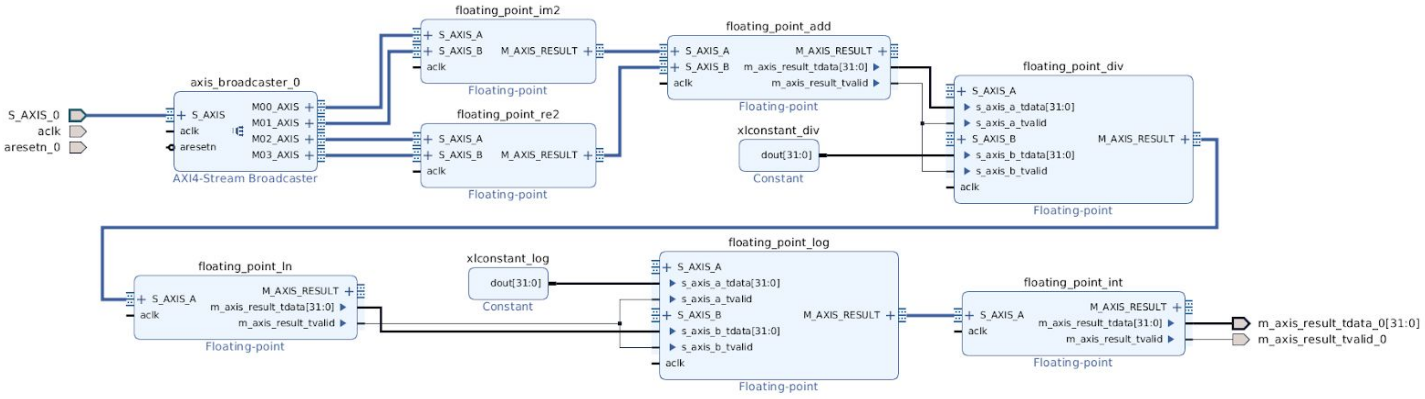
Après avoir vérifier le bon fonctionnement de l’IP FFT, j’ai ajouté un second module de calcul à la sortie de la FFT qui retourne au CPU la bande passante et la puissance maximale. Le block design est le suivant :



block design : analyse spectrale par fft

Le lien en vert représente le bus de données utilisées par l’IP FFT, ainsi que son résultat. Le lien en rouge correspond à l’interruption émise par le module qui analyse les résultats pour indiquer au CPU qu’ils sont prêts.

Le sous diagramme power_calc est utilisé pour calculer la puissance à partir des flottants retournés par le bloc FFT. Voici son détail :



block design : calcul de la puissance

Le stream de 64 bits est d'abord séparé en deux pour isoler la partie réelle et la partie imaginaire, ensuite chaque partie est élevée au carré puis les deux résultats sont additionnés. Cette addition est divisée par le carré du nombre de point de la FFT puis sont logarithme à base 10 est calculé. Enfin, le résultat final est converti en entier pour être exploité dans la logique du module qui analyse les résultats.

L'équation obtenue est : $P_{dB} = 1000 \times log((re^2 + im^2) \div points^2)$
J'ai ajouté un facteur 100 dans le but de conserver une meilleur précision lors d ela conversion en entier.

L'analyseur (nommé xfft_parser sur le block design et dont la logique peut être trouvée en annexe n°6), reçoit les 1024 valeurs de puissance est les enregistres dans un tableau, en parallèle il calcul la fréquence associé à chaque point avec l'équation : $F = index \times 2.10^9 \div 1024$.

Durant l'acquisition des 1024 point de puissance, l'index de la puissance maximale est enregistrée dans un registre. A la fin de l'acquisition, cet index est utilisé pour trouver la bande passante à -3 dB en vérifiant la valeur des puissances au dessus et en dessous de l'index de puissance maximale. La vérification est faire à -300 dB puisqu'il y a le facteur 100.

Lorsque le module a obtenu les index de la bande passante, il envoie une interruption au CPU et place les fréquences associées à la bande passante ainsi que la puissance maximale dans des registres.

La logique implémente aussi un système de reset permettant au CPU de piloter la FFT plusieurs fois de suite.

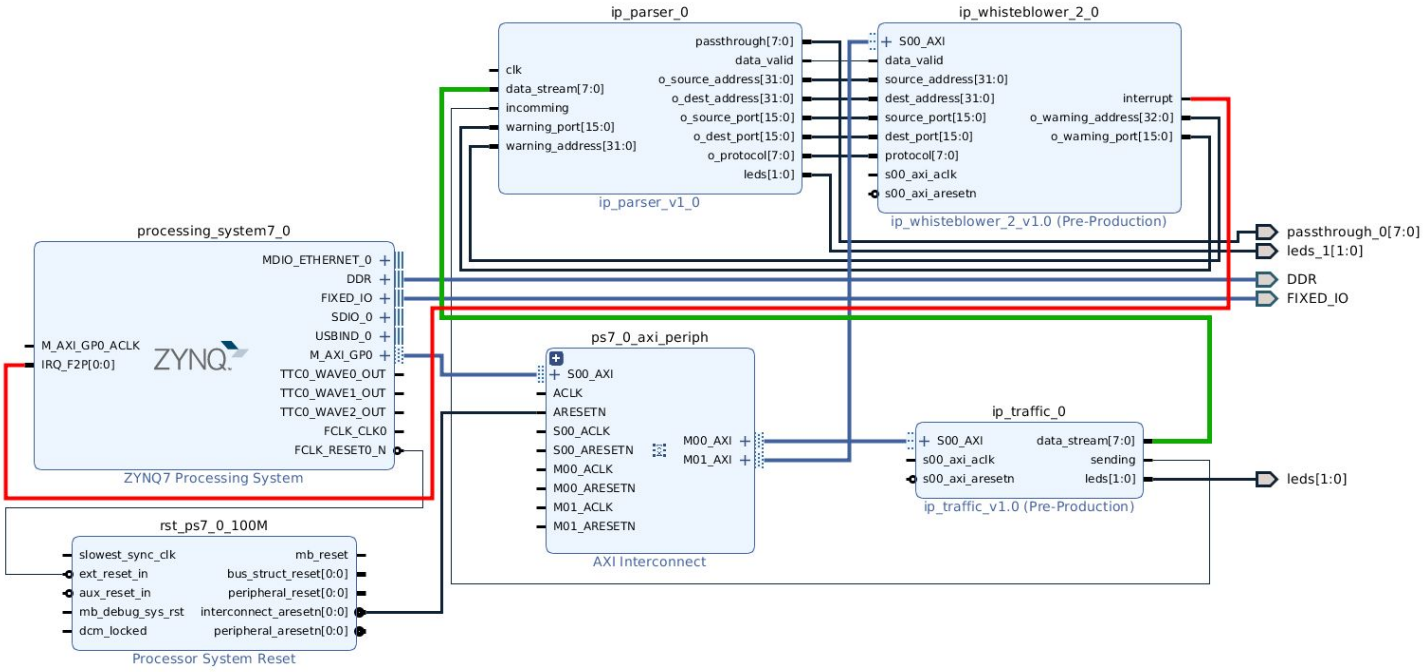
Pour piloter ce module j'utilise le code C qui se trouve en annexe n°7. Ce programme commence par créer un sous processus que boucle à l'infini et capte l'interruption du FPGA, ensuite il pilote la DMA comme vu précédemment, pour envoyer les données vers la FFT. Lorsque l'interruption est perçu, la boucle infinie se termine et les résultats (bande passante et puissance maximale) sont récupérés dans les registres du FPGA, puis affichés dans le terminal.

c. Analyse de trames IP

L'analyse de trame peut être utilisé en tant que protection de déni de service.

Il est possible avec ce module de protéger trois ports et trois adresses de destinations. Un seuil est défini par le CPU, si le nombre de paquets adressés à l'un des ports ou adresses protégés est plus grand que le seuil sur une fenêtre de dix secondes, une interruption est envoyée pour prévenir le CPU et les paquets ne sont plus transmis.

Ce troisième module FPGA est constitué de 3 IPs customs, le block design est le suivant :



block design : analyse de trame IP

Le premier IP génère du trafic ethernet IPv4 TCP ou UDP utilisé pour les tests. Pour générer le trafic j'utilise une sorte de machine à état avec un curseur qui boucle sur les différents headers ethernet, IP, TCP, ou UDP (annexe n°11). Le trafic est synchroniser entre le générateur et le receveur grâce au préambule du header ethernet, qui doit normalement être utilisé pour synchroniser les horloges. La plupart des headers ont une valeur par défaut, seulement les adresses IP, les ports et le protocol peuvent être choisi depuis le CPU grâce à une application C (annexe n°8) qui permet de choisir le type de trafic, par exemple :

```
$ ip-order -x -p TCP -s 192.168.10.51:22 -d 192.168.10.50:22
```

Permet de générer du trafic TCP, tandis que :

```
$ ip-order -k
```

Stop le trafic.

Le générateur est tout de même capable de préciser la taille des paquets avec les headers IPv4 internet header length (qui définit la taille du header IPv4) et total length (qui définit la taille du paquet IPv4), ainsi que le header TCP data offset (qui définit la taille du header TCP) ou le header UDP length (qui définit la taille du paquet UDP, sachant que le header fait toujours 8 octets).

Le second IP est le parser, il reçoit le trafic et l'analyse. Il enregistre notamment les adresses IP, les ports et le protocol, qui sont ensuite envoyés au troisième IP pour contrôler le nombres de requêtes. En parallèle, il transfère les paquets à un destinataire. Comme avec le premier IP, j'utilise un curseur pour parcourir les headers (annexe n°12). La taille des paquets est obtenus grâces aux headers pour ne pas faire d'erreur de parsing. Durant chaque transfert de trame, dès lors que l'IP connaît l'adresse IP de destination, il vérifie si cette adresse est désignée (par le troisième IP) comme subissant une attaque ; si c'est le cas, le transfert du paquet est stoppé et le curseur de la machine à état retourne à l'état initial. Il en est de même lorsque le port de destination est connu.

Le dernier IP est le whistleblower, il analyse les informations reçues par le parser et bloque le trafic en fonction de ces informations et des ordres du CPU. La vérification se fait sur une fenêtre de dix secondes grâce à un compteur. Chaque adresse IP et port protégé dispose de son propre compteur. A chaque fois qu'un paquet a été analysé, le whistleblower reçoit l'adresse et le port de destination du paquet de la part du parser. Il vérifie alors si l'un des deux est inclu dans ceux qu'il protège, si c'est le cas le compteur associé est incrémenté. Lorsqu'un compteur associé à une adresse ou un port protégé dépasse le seuil fixé par le CPU, une interruption est envoyé et le parser est informé. A l'expiration de la fenêtre de dix secondes, tous les compteurs sont réinitialisés. La logique associée à cet IP peut être trouvée en annexe n°13.

Pour configurer le whistleblower, j'utilise une application C (annexe n°9) qui permet de définir le seuil (en paquets par seconde sur une fenêtre de dix secondes) à partir duquel une adresse ou un port est bloqué, par exemple :

```
$ ip-config -t 10000
```

Fixe le seuil à 10 000 paquets par seconde.

La même application est utilisée pour configurer les adresses ou ports protégés :

```
$ ip-config -i 0 -a 192.168.10.50
$ ip-config -i 1 -p 22
```

Configure respectivement l'adresse 192.168.10.50 en tant que première adresse protégée et le port 22 en tant que deuxième port protégé.

Enfin, une dernière application (annexe n°10) permet d'intercepter les interruptions du whistleblower et d'afficher l'adresse ou le port bloqué dans le terminal.

5. Améliorations possibles

Des améliorations peuvent être faites sur les trois modules que ce soit en terme de fonctionnalités, ou même de la logique associée, en effet le FPGA de la Zybo ne dispose pas d'assez de LUTs pour accueillir les trois modules à la fois en l'état actuel.

Dans le module de détection de seuil, il serait intéressant d'allouer un grand nombre de registres, et le CPU pourrait fixer le seuil et la source des données à surveiller.

Avoir différents type de seuil (inférieur, supérieur, interne, externe) pourrait apporter une valeur ajoutée.

Une autre fonctionnalité intéressante serait d'avoir la possibilité de choisir d'autres type de vérification, comme par exemple l'évolution de la donnée (coefficient de la pente) .

Le module d'analyse spectrale est celui qui consomme le plus de LUTs, il faudrait réaliser les calculs sur des fixed points plutôt que sur des flottants afin de réduire la complexité de la logique.

Actuellement, les calculs de la bande passante et de la puissance maximale sont fait à partir d'un seul spectre. Superposer plusieurs résultats de la FFT permettrait de réduire le bruit.

Aussi, des plages de fréquences pourraient être définies à l'avance pour surveiller plusieures bandes d'émission à la fois.

Enfin, le module d'analyse de paquet pourrait être capable de parser les paquets de type ARP dans le but de détecter les attaques les utilisants.

Faire plus d'analyse sur les adresses IPs sources permettrait d'identifier des adresses suspectes et de les "blacklister". J'ai tenté de le faire, mais j'ai fait face à un problème de manque de LUTs.

Il serait aussi possible d'utiliser l'interface EMIO du Zynq pour transférer les paquets au CPU.

6. Difficultés rencontrées

La première difficulté que j'ai rencontré a été de prendre en main le logiciel Vivado et l'outil Petalinux. Ils ont tous deux de nombreuses fonctionnalités et cela demande beaucoup de temps et de recherches pour en faire le tour et comprendre comment les utiliser (par exemple la configuration du kernel de Petalinux, la configuration des entrées / sorties du zynq dans Vivado, l'utilisation d'un device tree ou d'un boot loader).

De plus, il faut faire attention à la compatibilité du logiciel Petalinux. En effet, il faut que la version de Petalinux soit la même que celle de Vivado, sinon la compilation du kernel avec un device tree généré à partir d'un block design ne fonctionnera pas. De la même manière, un block design réalisé sur windows ne peut pas être utilisé avec Petalinux.

La machine exécutant Petalinux doit aussi avoir certaines librairies installées sans lesquelles certaines commandes de Petalinux ne fonctionnent pas, le problème est que certaines de ces librairies ne sont pas listées dans la documentation de Petalinux, rendant le debuggage de l'installation compliqué.

La principale difficulté à laquelle j'ai dû faire face est l'utilisation d'un langage de description matériel, les concepts sont totalements différents que dans le cas des langage de programmation standards auxquelles je suis plus habitué. J'avais donc énormément de difficultés à debugger certain code.

De plus à chaque tentative de debuggage je devais recompiler le bitstream, ce qui me faisait perdre beaucoup de temps et qui est très frustrant. J'ai utilisé un simulateur pour éviter d'avoir à attendre le temps de compilation, mais dans certains cas, un code qui fonctionne sur simulateur ne fonctionne pas sur le matériel. Il faut bien différencier un code qui est synthétisable d'un code qui ne l'est, chose que je n'avais pas appréhender au début du projet.

Conclusion

Le système final propose des solutions pour surveiller les menaces sur les couches applicative, de transport et physique. Ces solutions sont encore au stage de preuves de conception, mais pourraient être utilisées pour une application réelle.

Durant ce projet j'ai dû me re-familiariser avec la programmation sur FPGA, et sur la programmation sur des couches basses en général dont je m'étais éloigné depuis le stage de fin d'étude.

J'ai pu me challenger sur des concepts et technologies que je ne connaissais pas ou peu (par exemple le device tree, les UIO driver, la DMA, la gestion des interruptions, ...), ce qui m'a permis d'en apprendre d'avantage sur le fonctionnement des systèmes électronique.

Remerciements

Je tiens à remercier M. Alexandre BOE et M. Thomas VANTROYS de m'avoir suivi durant ce projet et de m'avoir accueilli à l'IRCICA.

Annexes

Annexe n°1

```
#define MM2S_CONTROL_REGISTER 0x00
#define MM2S_STATUS_REGISTER 0x04
#define MM2S_START_ADDRESS 0x18
#define MM2S_LENGTH 0x28

#define S2MM_CONTROL_REGISTER 0x30
#define S2MM_STATUS_REGISTER 0x34
#define S2MM_DESTINATION_ADDRESS 0x48
#define S2MM_LENGTH 0x58

unsigned int dma_set(unsigned int* slv_ptr, int offset, unsigned int value) {
   slv_ptr[offset>>2] = value;
}

unsigned int dma_get(unsigned int* slv_ptr, int offset) {
   return slv_ptr[offset>>2];
}

int dma_mm2s_sync(unsigned int* slv_ptr) {
   unsigned int mm2s_status =  dma_get(slv_ptr, MM2S_STATUS_REGISTER);
   while(!(mm2s_status & 1<<12) || !(mm2s_status & 1<<1) ){
       mm2s_status =  dma_get(slv_ptr, MM2S_STATUS_REGISTER);
   }
}

int dma_s2mm_sync(unsigned int* slv_ptr) {
   unsigned int s2mm_status = dma_get(slv_ptr, S2MM_STATUS_REGISTER);
   while(!(s2mm_status & 1<<12) || !(s2mm_status & 1<<1)){
       s2mm_status = dma_get(slv_ptr, S2MM_STATUS_REGISTER);
   }
}

void memdump(void* ptr, int byte_count) {
   char *p = ptr;
   int offset;
   for (offset = 0; offset < byte_count; offset++) {
       printf("%02x", p[offset]);
       if (offset % 4 == 3) { printf(" "); }
   }
   printf("\n");
}


int main() {
   int fd = open("/dev/mem", O_RDWR | O_SYNC);
   // pointeur esclave AXI-DMA
   unsigned int* slv_ptr = mmap(NULL, 65535, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0x40400000);
   // pointeur source
   unsigned int* source_ptr  = mmap(NULL, 65535, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0x0e000000);
   // pointeur destination
   unsigned int* dest_ptr = mmap(NULL, 65535, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0x0f000000);
```

```
    // ecrit la donnee source
    memset(dest_ptr, 0xffffffff, 32);

    // reset les 2 cannaux
    dma_set(slv_ptr, S2MM_CONTROL_REGISTER, 4);
    dma_set(slv_ptr, MM2S_CONTROL_REGISTER, 4);
    // assure que les 2 cannaux soient stoppés
    dma_set(slv_ptr, S2MM_CONTROL_REGISTER, 0);
    dma_set(slv_ptr, MM2S_CONTROL_REGISTER, 0);
    // indique l'adresse de destination
    dma_set(slv_ptr, S2MM_DESTINATION_ADDRESS, 0x0f000000);
    // indique l'adresse source
    dma_set(slv_ptr, MM2S_START_ADDRESS, 0x0e000000)
    // demarre le canal d'entree
    dma_set(slv_ptr, S2MM_CONTROL_REGISTER, 0xf001);
    // demarre le canal de sortie
    dma_set(slv_ptr, MM2S_CONTROL_REGISTER, 0xf001);
    // indique la taille de l'entree
    dma_set(slv_ptr, S2MM_LENGTH, 32);
    // indique la taille de la sortie
    dma_set(slv_ptr, MM2S_LENGTH, 32);
    // attend la fin du transfert
    dma_mm2s_sync(slv_ptr);
    dma_s2mm_sync(slv_ptr);

    // print la donnee recu
    printf("Destination memory block: "); memdump(dest_ptr, 32);

    fclose(fd);
    munmap(slv_ptr, 65535);
    munmap(source_ptr, 65535);
    munmap(dest_ptr, 65535);

    return 0;
}
```

Annexe n°2

```c
#define IRQ_NUM   46

static irqreturn_t myint_handler(int irq, void *dev_id)
{
 printk("Interrupt Occurred : Button pressed !\n");
  return IRQ_HANDLED;
}

static int __init myint_init(void)
{

 //enregistre la ligne d'interruption
 int req_status = request_irq(IRQ_NUM, myint_handler, 0, "myint", NULL);
 if (req_status) {
   printk(KERN_ERR "myint_init: Cannot register IRQ %d\n", IRQ_NUM);
   return req_status;
 }
 else {
   printk(KERN_INFO "myint_init: Registered IRQ %d\n", IRQ_NUM);
 }

 return 0;
}

static void __exit myint_edit(void)
{
 //libère la ligne d'interruption
 free_irq(IRQ_NUM, NULL);
}

module_init(myint_init);
module_exit(myint_edit);

MODULE_LICENSE("GPL");
```

Annexe n°3

```
// Add user logic here
reg [32 : 0] data_out = 0;
reg[3 : 0] data_led = 0;
integer interrupt_flag = 0;

assign LEDS = data_led;
assign INTERRUPT = interrupt_flag;

always @( posedge S_AXI_ACLK )
begin
  case(source)
  32'h504d4f44: value_0 <= {data_out[31 : 8], PMOD[7 : 0]}; //source == PMOD
  32'h42544e53: value_0 <= {data_out[31 : 4], SWITCHES[3 : 0]}; //source = BTNS
  endcase
end

always @( posedge S_AXI_ACLK )
begin
  data_led[3 : 0] <= value_0[3 : 0];  // show value_0 register on leds

  if (value_0 > threshold_0)
       begin
      channel <= 32'h0; // channel 0
      interrupt_flag = 1; // assert an interrupt
    end
  else
       begin
      interrupt_flag = 0;
    end
end
// User logic ends
```

Annexe n°4

```c
#define SOURCE 0x42544e53 // BTNS
//#define SOURCE 0x504d4f44 // PMOD

#define THRESHOLD 12  // threshold value, if value_0 register exceeds it, an interruption is sent

#define MODULE1_UIOD "/dev/uio0" // module1 uio device driver location
#define MAP_SIZE 0x10000  // found in vivado address editor

// found in xilinx AXI documentation
#define SOURCE_OFFSET 0x00
#define THRESHOLD_0_OFFSET 0x04
#define VALUE_0_OFFSET 0x08
#define CHANNEL_OFFSET 0x0C

volatile sig_atomic_t CONTINU = 1;

void signal_handler(int signum, siginfo_t *info, void *ptr)
{
   CONTINU = 0;
}

void catch_signals()
{
   static struct sigaction _sigacterm;
   static struct sigaction _sigactint;

   memset(&_sigacterm, 0, sizeof(_sigacterm));
   _sigacterm.sa_sigaction = signal_handler;
   _sigacterm.sa_flags = SA_SIGINFO;

   memset(&_sigactint, 0, sizeof(_sigactint));
   _sigactint.sa_sigaction = signal_handler;
   _sigactint.sa_flags = SA_SIGINFO;

   sigaction(SIGINT, &_sigactint, NULL);
   sigaction(SIGTERM, &_sigacterm, NULL);
}

int main() {
   time_t t;
   struct tm tm;
   FILE* fd;
   int uiofd;
   unsigned int value, channel, threshold;
   uint32_t info;
   ssize_t flag;

   printf("Start of module 1 process\n\r");

    // open uio device driver
   uiofd = open(MODULE1_UIOD, O_RDWR);
   if(uiofd < 0) {
       printf("Error opening uio device\n\r");
       exit(EXIT_FAILURE);
   }
```

```
    // get a pointer to memory mapped device
    void* ptr = mmap(NULL, MAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, uiofd, 0);

    // open a file to store values return by the PL
    fd = fopen("./channel0.csv","w");
    if(fd < 0) {
        printf("Error opening csv file\n\r");
        exit(EXIT_FAILURE);
    }

    catch_signals();

    // tell the PL module to use BTNS as source
     *((unsigned *)(ptr + SOURCE_OFFSET)) = SOURCE;
    printf("Source setted\n\r");

    // tell the PL module the treshold
    *((unsigned *)(ptr + THRESHOLD_0_OFFSET)) = THRESHOLD;
    printf("threshold setted to %d\n", THRESHOLD);

    // init csv headers
    fprintf(fd,"\"timestamp\",\"value\"");

    if(fork() == 0) { // subprocess
        while (CONTINU) {
            info = 1;
            // enable interruption
            flag = write(uiofd, &info, sizeof(info));
            if (flag != (ssize_t)sizeof(info)) {
                printf("Error restting interruption mask\n\r");
                exit(EXIT_FAILURE);
            }
            // look for interrupt
            flag = read(uiofd, &info, sizeof(info));
            if (flag == (ssize_t)sizeof(info)) {
                channel = *((unsigned *)(ptr + CHANNEL_OFFSET));
                value = *((unsigned *)(ptr + VALUE_0_OFFSET));
                threshold = *((unsigned *)(ptr + THRESHOLD_0_OFFSET));

                fprintf(stderr, "Threshold exceeded on channel %d for %u times !\n\r", channel,info);
                fprintf(stderr, "Value is : %d\n\r", value);
                fprintf(stderr, "Threshold is set to : %d!\n\r", threshold);
            }
        }
    } else { // main process
        // recurrently get values from the PL
        while(CONTINU) {
            value = *((unsigned *)(ptr + VALUE_0_OFFSET));
            t = time(NULL);
            tm = *localtime(&t);
            fprintf(fd, "\n\"%d-%d-%d %d:%d:%d\",\"%d\"", tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
tm.tm_hour, tm.tm_min, tm.tm_sec, value);

            sleep(3);
        }
    }
```

```
    fclose(fd);
    munmap(ptr, MAP_SIZE);
    close(uiofd);

    printf("End of module 1 process\n\r");

    exit(EXIT_SUCCESS);
}
```

Annexe n°5

```
#define FREQUENCY 433000000 //433MHz
#define PHASE 0
#define AMPLITUDE 2.3
#define SAMPLE_RATE 2000000000 //2GHz
#define MAX_POINT 1024

void main() {
   double i;
   double t,re,im,x;
   double w = 2*M_PI*FREQUENCY;
   FILE* fd;

   fd = fopen("./sine_data","w");

   for(i = 0 ; i < MAX_POINT ; i++) {
       t = (i / (SAMPLE_RATE));
       x = (w*t) + PHASE;
       re = AMPLITUDE*cos(x);
       im = AMPLITUDE*sin(x);
       fprintf(fd,"%f, %f,\n", (float)re,(float)im);
   }

   fclose(fd);

   exit(EXIT_SUCCESS);
}
```

Annexe n°6

```
 // Add user logic here
integer data_index = 0;
integer index_max = 0;
integer bw_index = 0;
integer bw_high = 0;
integer bw_low = 0;
reg data_ready = 0;
reg bw_high_ready = 0;
reg bw_low_ready = 0;
reg [3 : 0] leds_data = 0;

assign READY = data_ready && bw_high_ready && bw_low_ready;
assign LEDS = leds_data;

integer frequency [0 : FFT_MAX_NUM_PTS-1];
integer power [0 : FFT_MAX_NUM_PTS-1];

always @( posedge S_AXI_ACLK)
begin
  if(slv_reg0 == 32'hffffffff)
  begin
    data_ready <= 1'b0;
    data_index = 0;
  end
  else begin
    if(data_index > FFT_MAX_NUM_PTS-1)
    begin
      data_ready <= 1'b1;
    end
    else begin
      if(S_AXIS_TVALID)
      begin
        power[data_index] <= S_AXIS_TDATA;
        frequency[data_index] = data_index * ratio;
        data_index = data_index + 1;
      end
    end
  end
end

always @( posedge S_AXI_ACLK)
begin
  if(slv_reg0 == 32'hffffffff)
  begin
    index_max = 0;
  end
  else begin
    if (!data_ready)
    begin
      if(power[data_index] > power[index_max])
      begin
        index_max = data_index;
```

```
      end
    end
  end
end

always @(posedge S_AXI_ACLK)
begin
  if(slv_reg0 == 32'hffffffff)
  begin
    leds_data[3:2] <= 2'b0;
    bw_index = 0;
  end
  else begin
    if (data_ready && (!bw_high_ready || !bw_low_ready))
    begin
      leds_data[3:2] <= 2'b11;
      bw_index = bw_index + 1;
    end
  end
end

always @( posedge S_AXI_ACLK)
begin
  if(slv_reg0 == 32'hffffffff)
  begin
    leds_data[1:1] <= 1'b0;
    bw_high_ready <= 1'b0;
    bw_high = 0;
  end
  else begin
    if(data_ready && !bw_high_ready)
    begin
       if(power[index_max+bw_index] < power[index_max]-300)
      begin
         leds_data[1:1] <= 1'b1;
        bw_high_ready <= 1'b1;
        bw_high = index_max + bw_index;
      end
    end
  end
end

always @( posedge S_AXI_ACLK)
begin
  if(slv_reg0 == 32'hffffffff)
  begin
    leds_data[0:0] <= 1'b0;
    bw_low_ready <= 1'b0;
    bw_low = 0;
  end
  else begin
    if(data_ready && !bw_low_ready)
    begin
```

```
        if(power[index_max-bw_index] < power[index_max]-300)
       begin
          leds_data[0:0] <= 1'b1;
         bw_low_ready <= 1'b1;
         bw_low = index_max - bw_index;
       end
     end
  end
end

always @( posedge S_AXI_ACLK )
begin
  if(slv_reg0 == 32'hffffffff)
  begin
    slv_reg1 <= 0;
    slv_reg2 <= 0;
    slv_reg3 <= 0;
  end
  else begin
    if(READY)
    begin
      slv_reg1 <= power[index_max];
      slv_reg2 <= frequency[bw_high];
      slv_reg3 <= frequency[bw_low];
    end
  end
end

// User logic ends
```

Annexe n°7

```
#include "csine_data.h"

#define XFFT_PARSER_UIOD "/dev/uio1"
#define XFFT_PARSER_ADDR 0x43C00000
#define XFFT_PARSER_MAP_SIZE 0x10000
#define XFFT_PARSER_RESET_OFFSET 0x00
#define XFFT_PARSER_POWER_OFFSET 0x04
#define XFFT_PARSER_BWH_OFFSET 0x08
#define XFFT_PARSER_BWL_OFFSET 0x0C

#define AXI_SLAVE_ADDR 0x40400000 //found in device tree or adress editor
#define MM2S_BASE_ADDR 0x0e000000

#define MEM_SIZE 0x10000

#define MM2S_CONTROL_REGISTER 0x00 //found in device tree
#define MM2S_STATUS_REGISTER 0x04
#define MM2S_START_ADDRESS 0x18
#define MM2S_LENGTH 0x28

typedef struct data_in_t
{
    float data_re;
    float data_im;
} data_in_t;

extern float sig_csine_waves[FFT_MAX_NUM_PTS * 2];

unsigned int dma_set(unsigned int* dma_slv_ptr, int offset, unsigned int value) {
   dma_slv_ptr[offset>>2] = value;
}

unsigned int dma_get(unsigned int* dma_slv_ptr, int offset) {
   return dma_slv_ptr[offset>>2];
}

void dma_mm2s_status(unsigned int* dma_slv_ptr) {
   unsigned int status = dma_get(dma_slv_ptr, MM2S_STATUS_REGISTER);
   printf("Memory-mapped to stream status (0x%08x@0x%02x):", status, MM2S_STATUS_REGISTER);
   if (status & 0x00000001) printf(" halted"); else printf(" running");
   if (status & 0x00000002) printf(" idle");
   if (status & 0x00000008) printf(" SGIncld");
   if (status & 0x00000010) printf(" DMAIntErr");
   if (status & 0x00000020) printf(" DMASlvErr");
   if (status & 0x00000040) printf(" DMADecErr");
   if (status & 0x00000100) printf(" SGIntErr");
   if (status & 0x00000200) printf(" SGSlvErr");
   if (status & 0x00000400) printf(" SGDecErr");
   if (status & 0x00001000) printf(" IOC_Irq");
   if (status & 0x00002000) printf(" Dly_Irq");
   if (status & 0x00004000) printf(" Err_Irq");
   printf("\n");
```

```
}

int dma_mm2s_sync(unsigned int* dma_slv_ptr) {
   unsigned int mm2s_status =  dma_get(dma_slv_ptr, MM2S_STATUS_REGISTER);
   while(!(mm2s_status & 1<<12) || !(mm2s_status & 1<<1) ){
       dma_mm2s_status(dma_slv_ptr);

       mm2s_status =  dma_get(dma_slv_ptr, MM2S_STATUS_REGISTER);
   }
}

void main() {
   data_in_t input_buffer[FFT_MAX_NUM_PTS];
   int continu = 1;
   int uiofd;
   int bw_high, bw_low;
   float power_max;

   // Reset results
   // open uio device driver
   uiofd = open(XFFT_PARSER_UIOD, O_RDWR);
   if(uiofd < 0) {
       printf("Error opening uio device\n\r");
       exit(EXIT_FAILURE);
   }

   // get a pointer to memory mapped device
   void* ptr = mmap(NULL, XFFT_PARSER_MAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, uiofd, 0);

   *((unsigned *)(ptr + XFFT_PARSER_RESET_OFFSET)) = 0xffffffff;

   usleep(1000);

   *((unsigned *)(ptr + XFFT_PARSER_RESET_OFFSET)) = 0x0;

   printf("FFT results reset\n\r");

   if(fork() == 0) {
       int j = 0;

       int fd = open("/dev/mem", O_RDWR | O_SYNC);

       unsigned int* slv_ptr = mmap(NULL, MEM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
AXI_SLAVE_ADDR);
       unsigned int* source_ptr  = mmap(NULL, sizeof(input_buffer), PROT_READ | PROT_WRITE,
MAP_SHARED, fd, MM2S_BASE_ADDR);

       for(int i=0 ; i < FFT_MAX_NUM_PTS ; i++) {
           input_buffer[i].data_re = sig_csine_waves[j];
           j++;
           input_buffer[i].data_im = sig_csine_waves[j];
           j++;
       }
```

```
        memcpy(source_ptr, input_buffer, sizeof(input_buffer));

        // reset DMA
        dma_set(slv_ptr, MM2S_CONTROL_REGISTER, 4);
        // halt DMA
        dma_set(slv_ptr, MM2S_CONTROL_REGISTER, 0);
        // assert source address
        dma_set(slv_ptr, MM2S_START_ADDRESS, MM2S_BASE_ADDR);
        // start channel
        dma_set(slv_ptr, MM2S_CONTROL_REGISTER, 0xf001);
        // assert transfert length
        dma_set(slv_ptr, MM2S_LENGTH, sizeof(input_buffer));
        // wait for transfert to finish
        dma_mm2s_sync(slv_ptr);

        munmap(slv_ptr,MEM_SIZE);
        munmap(source_ptr,sizeof(data_in_t)*FFT_MAX_NUM_PTS);
        close(fd);

    } else {
        uint32_t info;
        ssize_t flag;

        while (continu) {
            info = 1;
            // eflagale interrupt
            flag = write(uiofd, &info, sizeof(info));
            if (flag != (ssize_t)sizeof(info)) {
                    printf("Error restting interruption mask\n\r");
                    exit(EXIT_FAILURE);
            }
            // look for interrupt
            flag = read(uiofd, &info, sizeof(info));
            if (flag == (ssize_t)sizeof(info)) {
                printf("fft results are ready \n\r");
                continu = 0;
            }
        }
    }

    power_max = *((unsigned *)(ptr + XFFT_PARSER_POWER_OFFSET));
    bw_high = *((unsigned *)(ptr + XFFT_PARSER_BWH_OFFSET));
    bw_low = *((unsigned *)(ptr + XFFT_PARSER_BWL_OFFSET));

    power_max = power_max / 100;

    printf("BW: %d - %d, Pmax = %f\n\r", bw_low,bw_high,power_max);

    munmap(ptr, XFFT_PARSER_MAP_SIZE);
    close(uiofd);
    exit(EXIT_SUCCESS);
}
```

Annexe n°8

```
 #define UIOD "/dev/uio0"
#define MAP_SIZE 0x10000
#define SEND_OFFSET 0x00
#define SOURCE_IP_OFFSET 0x04
#define DEST_IP_OFFSET 0x08
#define SOURCE_PORT_OFFSET 0x0C
#define DEST_PORT_OFFSET 0x10
#define PROTOCOL_OFFSET 0x14
#define LENGTH_OFFSET 0x18

void main(int argc, char *argv[])
{
   int opt;
   int cmd = -1;
   char *source_ip,*dest_ip;
   unsigned int protocol, source_port, dest_port;

   while((opt = getopt(argc, argv, "xkp:s:d:")) != -1)
   {
       switch(opt)
       {
           case 'x':
               cmd = 1;
               break;
           case 'k':
               cmd = 0;
               break;
           case 'p':
               if(strcmp(optarg,"TCP") == 0) {
                   protocol = 0x6;
               } else if (strcmp(optarg,"UDP") == 0) {
                   protocol = 0x11;
               } else {
                   printf("Wrong protocol \n\r");
                   exit(EXIT_FAILURE);
               }
               break;
           case 's':
               source_ip = strsep(&optarg, ":");
               source_port = atoi(strsep(&optarg, ":"));
               break;
           case 'd':
               dest_ip = strsep(&optarg, ":");
               dest_port = atoi(strsep(&optarg, ":"));
               break;
       }
   }

    int uiofd = open(UIOD, O_RDWR);
    if (uiofd < 0) {
        printf("Error opening UIO device\n\r");
        exit(EXIT_FAILURE);
```

```
    }

    void *ptr = mmap(NULL, MAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, uiofd, 0);

   if(cmd == 1){
        *((unsigned *)(ptr + SOURCE_IP_OFFSET)) = inet_addr(source_ip);
        *((unsigned *)(ptr + DEST_IP_OFFSET)) = inet_addr(dest_ip);
        *((unsigned *)(ptr + SOURCE_PORT_OFFSET)) = source_port;
        *((unsigned *)(ptr + DEST_PORT_OFFSET)) = dest_port;
        *((unsigned *)(ptr + PROTOCOL_OFFSET)) = protocol;
        *((unsigned *)(ptr + LENGTH_OFFSET)) = 0x0;
        *((unsigned *)(ptr + SEND_OFFSET)) = 0xffffffff; // start traffic
        printf("Traffic started :\n\r");
        printf("protocol = %d\n\r",protocol);
        printf("source ip = %s\n\r",source_ip);
        printf("source port = %d\n\r",source_port);
        printf("dest ip = %s\n\r",dest_ip);
        printf("dest port = %d\n\r",dest_port);
   } else if (cmd == 0) {
        *((unsigned *)(ptr + SEND_OFFSET)) = 0x0; // stop traffic
        printf("Traffic stopped \n\r");
   } else {
        printf("Error: no command specified (use -x or -k)\n\r");
        munmap(ptr, MAP_SIZE);
        close(uiofd);
        exit(EXIT_FAILURE);
   }

    munmap(ptr, MAP_SIZE);
   close(uiofd);

    exit(EXIT_SUCCESS);
}
```

Annexe n°9

```
#define UIOD "/dev/uio1"
#define MAP_SIZE 0x10000
#define THRESHOLD_OFFSET 0x00
#define PORT0_OFFSET 0x04
#define PORT1_OFFSET 0x08
#define PORT2_OFFSET 0x0C
#define IP0_OFFSET 0x10
#define IP1_OFFSET 0x14
#define IP2_OFFSET 0x18

void main(int argc, char *argv[])
{
   int opt;
   int cmd = -1;
   int index = -1;
   int threshold;
   char *ip;
   unsigned int port;

   while((opt = getopt(argc, argv, "t:p:a:i:")) != -1)
   {
       switch(opt)
       {
           case 't':
               cmd = 0;
               threshold = atoi(optarg);
               break;
           case 'i':
               index = atoi(optarg);
               break;
           case 'p':
               cmd = 1;
               port = atoi(optarg);
               break;
           case 'a':
               cmd = 2;
               ip = optarg;
               break;
       }
   }

    int uiofd = open(UIOD, O_RDWR);
    if (uiofd < 0) {
        printf("Error opening UIO device\n\r");
        exit(EXIT_FAILURE);
    }

    void *ptr = mmap(NULL, MAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, uiofd, 0);

   if(cmd == 0){
       *((unsigned *)(ptr + THRESHOLD_OFFSET)) = threshold * 10;
       printf("Threshold set to %d per second (10 sec window)\n\r", threshold);
```

```
    } else if (cmd == 1) {
        switch(index) {
            case 0:
                *((unsigned *)(ptr + PORT0_OFFSET)) = port;
                printf("Protected port 0 set to %d\n\r", port);
                break;
            case 1:
                *((unsigned *)(ptr + PORT1_OFFSET)) = port;
                printf("Protected port 1 set to %d\n\r", port);
                break;
            case 2:
                *((unsigned *)(ptr + PORT2_OFFSET)) = port;
                printf("Protected port 2 set to %d\n\r", port);
                break;
        }
    } else if (cmd == 2) {
        switch(index) {
            case 0:
                *((unsigned *)(ptr + IP0_OFFSET)) = inet_addr(ip);
                printf("Protected address 0 set to %s\n\r", ip);
                break;
            case 1:
                *((unsigned *)(ptr + IP1_OFFSET)) = inet_addr(ip);
                printf("Protected address 1 set to %s\n\r", ip);
                break;
            case 2:
                *((unsigned *)(ptr + IP2_OFFSET)) = inet_addr(ip);
                printf("Protected address 2 set to %s\n\r", ip);
                break;
        }
    } else {
        printf("Error\n\r");
        munmap(ptr, MAP_SIZE);
        close(uiofd);
        exit(EXIT_FAILURE);
    }

     munmap(ptr, MAP_SIZE);
    close(uiofd);

     exit(EXIT_SUCCESS);
}
```

Annexe n°10

```
 #define UIOD "/dev/uio1"
#define MAP_SIZE 0x10000
#define PORT_OFFSET 0x1C
#define ADDRESS_OFFSET 0x20
#define TYPE_OFFSET 0x24

volatile sig_atomic_t CONTINU = 1;

void signal_handler(int signum, siginfo_t *info, void *ptr)
{
       CONTINU = 0;
}

void catch_signals()
{
       static struct sigaction _sigacterm;
       static struct sigaction _sigactint;

       memset(&_sigacterm, 0, sizeof(_sigacterm));
       _sigacterm.sa_sigaction = signal_handler;
       _sigacterm.sa_flags = SA_SIGINFO;

       memset(&_sigactint, 0, sizeof(_sigactint));
       _sigactint.sa_sigaction = signal_handler;
       _sigactint.sa_flags = SA_SIGINFO;

       sigaction(SIGINT, &_sigactint, NULL);
       sigaction(SIGTERM, &_sigacterm, NULL);
}

int main() {
   int uiofd;
   unsigned int type, port;
   uint32_t info;
   ssize_t flag;
   struct in_addr ip_addr;

   // open uio device driver
   uiofd = open(UIOD, O_RDWR);
   if(uiofd < 0) {
       printf("Error opening uio device\n\r");
       exit(EXIT_FAILURE);
   }

   // get a pointer to memory mapped device
   void* ptr = mmap(NULL, MAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, uiofd, 0);

   catch_signals();

   while (CONTINU) {
       info = 1;
```

```
        // enable interruption
        flag = write(uiofd, &info, sizeof(info));
        if (flag != (ssize_t)sizeof(info)) {
                printf("Error restting interruption mask\n\r");
                exit(EXIT_FAILURE);
        }
        // look for interrupt
        flag = read(uiofd, &info, sizeof(info));
        if (flag == (ssize_t)sizeof(info)) {
            type = *((unsigned *)(ptr + TYPE_OFFSET));
            if(type == 0x706f7274){
                port = *((unsigned *)(ptr + PORT_OFFSET));
                printf("warning on port %d\n\r", port);
            } else if(type == 0x61646472) {
                ip_addr.s_addr = *((unsigned *)(ptr + ADDRESS_OFFSET));
                printf("warning on address %s\n\r", inet_ntoa(ip_addr));
            }
        }
    }

    munmap(ptr, MAP_SIZE);
    close(uiofd);

    exit(EXIT_SUCCESS);
}
```

Annexe n°11

```
//ETH headers
localparam [7 : 0] syncword = 8'b10101010;
localparam [7 : 0] sfd = 8'b10101011;
localparam [47 : 0] source_mac = 48'h004096acf8dc;
localparam [47 : 0] dest_mac = 48'h0040964e5150;
localparam [31 : 0] tpid = 32'h81008100;
localparam [15 : 0] ether_type = 16'h0800; //IPv4
localparam [31 : 0] fcs = 32'hc704dd7b;
//IPv4 headers
localparam [3 : 0] version = 4'h4;
localparam [3 : 0] ihl = 4'd5; // no options
localparam [5 : 0] dscp = 5'b0;
localparam [1 : 0] ecn = 2'b0;
localparam [15 : 0] identification = 15'h0;
localparam [2 : 0] ipv4_flags = 3'b0;
localparam [12 : 0] fragment_offset = 12'h0;
localparam [7 : 0] ttl = 8'h0;
localparam [15 : 0] header_checksum = 16'h0;
//TCP headers
localparam [31 : 0] sequence_number = 32'b0;
localparam [31 : 0] acknowledgment_number = 32'b0;
localparam [3 : 0] data_offset = 4'd5; // no options
localparam [3 : 0] reserved = 3'b0;
localparam [8 : 0] tcp_flags = 9'b000000000; // NS, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN
localparam [15 : 0] window_size = 16'h0;
localparam [15 : 0] urgent_pointer = 16'h0;
//TCP & UDP headers
localparam [15 : 0] checksum = 16'h0;

// 1 byte states
localparam [7 : 0] START = 8'd0;
//ETH
localparam [7 : 0] PSEUDO_PREAMBLE0 = 8'd1;
localparam [7 : 0] PSEUDO_PREAMBLE1 = 8'd2;
localparam [7 : 0] DEST_MAC0 = 8'd3;
localparam [7 : 0] DEST_MAC1 = 8'd4;
localparam [7 : 0] DEST_MAC2 = 8'd5;
localparam [7 : 0] DEST_MAC3 = 8'd6;
localparam [7 : 0] DEST_MAC4 = 8'd7;
localparam [7 : 0] DEST_MAC5 = 8'd8;
localparam [7 : 0] SOURCE_MAC0 = 8'd9;
localparam [7 : 0] SOURCE_MAC1 = 8'd10;
localparam [7 : 0] SOURCE_MAC2 = 8'd11;
localparam [7 : 0] SOURCE_MAC3 = 8'd12;
localparam [7 : 0] SOURCE_MAC4 = 8'd13;
localparam [7 : 0] SOURCE_MAC5 = 8'd14;
localparam [7 : 0] ETH_TAG0 = 8'd15;
localparam [7 : 0] ETH_TAG1 = 8'd16;
localparam [7 : 0] ETH_TAG2 = 8'd17;
localparam [7 : 0] ETH_TAG3 = 8'd18;
localparam [7 : 0] ETH_TYPE0 = 8'd19;
localparam [7 : 0] ETH_TYPE1 = 8'd20;
localparam [7 : 0] FCS0 = 8'd21;
localparam [7 : 0] FCS1 = 8'd22;
localparam [7 : 0] FCS2 = 8'd23;
```

```
localparam [7 : 0] FCS3 = 8'd24;
//IPv4
localparam [7 : 0] IP_VER_IHL = 8'd25;
localparam [7 : 0] IP_DSCP_ECN = 8'd26;
localparam [7 : 0] IP_LEN0 = 8'd27;
localparam [7 : 0] IP_LEN1 = 8'd28;
localparam [7 : 0] IP_ID0 = 8'd29;
localparam [7 : 0] IP_ID1 = 8'd30;
localparam [7 : 0] IP_FLAGS_FRAG0 = 8'd31;
localparam [7 : 0] IP_FRAG1 = 8'd32;
localparam [7 : 0] IP_TTL = 8'd33;
localparam [7 : 0] IP_PROTOCOL = 8'd34;
localparam [7 : 0] IP_CHECK0 = 8'd35;
localparam [7 : 0] IP_CHECK1 = 8'd36;
localparam [7 : 0] IP_SOURCE_ADDR0 = 8'd37;
localparam [7 : 0] IP_SOURCE_ADDR1 = 8'd38;
localparam [7 : 0] IP_SOURCE_ADDR2 = 8'd39;
localparam [7 : 0] IP_SOURCE_ADDR3 = 8'd40;
localparam [7 : 0] IP_DEST_ADDR0 = 8'd41;
localparam [7 : 0] IP_DEST_ADDR1 = 8'd42;
localparam [7 : 0] IP_DEST_ADDR2 = 8'd43;
localparam [7 : 0] IP_DEST_ADDR3 = 8'd44;
//TCP
localparam [7 : 0] TCP_SOURCE_PORT0 = 8'd45;
localparam [7 : 0] TCP_SOURCE_PORT1 = 8'd46;
localparam [7 : 0] TCP_DEST_PORT0 = 8'd47;
localparam [7 : 0] TCP_DEST_PORT1 = 8'd48;
localparam [7 : 0] TCP_SEQ0 = 8'd49;
localparam [7 : 0] TCP_SEQ1 = 8'd50;
localparam [7 : 0] TCP_SEQ2 = 8'd51;
localparam [7 : 0] TCP_SEQ3 = 8'd52;
localparam [7 : 0] TCP_ACK0 = 8'd53;
localparam [7 : 0] TCP_ACK1 = 8'd54;
localparam [7 : 0] TCP_ACK2 = 8'd55;
localparam [7 : 0] TCP_ACK3 = 8'd56;
localparam [7 : 0] TCP_OFFSET_FLAGS0 = 8'd57;
localparam [7 : 0] TCP_FLAGS1 = 8'd58;
localparam [7 : 0] TCP_WINDOW0 = 8'd59;
localparam [7 : 0] TCP_WINDOW1 = 8'd60;
localparam [7 : 0] TCP_CHECK0 = 8'd61;
localparam [7 : 0] TCP_CHECK1 = 8'd62;
localparam [7 : 0] TCP_URG0 = 8'd63;
localparam [7 : 0] TCP_URG1 = 8'd64;
//UDP
localparam [7 : 0] UDP_SOURCE_PORT0 = 8'd65;
localparam [7 : 0] UDP_SOURCE_PORT1 = 8'd66;
localparam [7 : 0] UDP_DEST_PORT0 = 8'd67;
localparam [7 : 0] UDP_DEST_PORT1 = 8'd68;
localparam [7 : 0] UDP_LEN0 = 8'd69;
localparam [7 : 0] UDP_LEN1 = 8'd70;
localparam [7 : 0] UDP_CHECK0 = 8'd71;
localparam [7 : 0] UDP_CHECK1 = 8'd72;
//Payload
localparam [7 : 0] _DATA = 8'd73;
//Protocol
localparam [7 : 0] _TCP = 8'd6;
localparam [7 : 0] _UDP = 8'd17;
```

```
reg [31 : 0] source_address;
reg [31 : 0] dest_address;
reg [15 : 0] source_port;
reg [15 : 0] dest_port;
reg [7 : 0] protocol;
reg [15 : 0] data_length;
reg [15 : 0] total_length;
reg [15 : 0] udp_length;

reg [7 : 0] data_reg;
reg sending_reg = 1'b0;
reg [1 : 0] leds_reg = 2'b00;
reg [7 : 0] cursor = 8'd0;

assign DATA_STREAM = data_reg;
assign SENDING = sending_reg;
assign LEDS = leds_reg;

//logic
always @ (posedge S_AXI_ACLK) begin
 if(SENDING) begin
   leds_reg <= 2'b11;
 end else begin
   leds_reg <= 2'b00;
 end
end

always @ (posedge S_AXI_ACLK) begin
 if(send_order == 32'hffffffff || SENDING) begin
   case (cursor)
     START: begin
         source_address <= source_address_order;
         dest_address <= dest_address_order;
         source_port <= source_port_order[15 : 0];
         dest_port <= dest_port_order[15 : 0];
         protocol <= protocol_order[7 : 0];
         data_length <= 16'b0;//data_length_order[15 : 0];
         sending_reg <= 1'b1;
         cursor <= PSEUDO_PREAMBLE0;
     end
      //ETH
      PSEUDO_PREAMBLE0: begin
        cursor <= PSEUDO_PREAMBLE1;
        data_reg <= syncword;
      end
      PSEUDO_PREAMBLE1: begin
        cursor <= DEST_MAC0;
        data_reg <= syncword;
      end
      DEST_MAC0: begin
        cursor <= DEST_MAC1;
        data_reg <= dest_mac[47 : 40];
      end
      DEST_MAC1: begin
        cursor <= DEST_MAC2;
        data_reg <= dest_mac[39 : 32];
```

```
    end
    DEST_MAC2: begin
      cursor <= DEST_MAC3;
      data_reg <= dest_mac[31 : 24];
    end
    DEST_MAC3: begin
      cursor <= DEST_MAC4;
      data_reg <= dest_mac[23 : 16];
    end
    DEST_MAC4: begin
      cursor <= DEST_MAC5;
      data_reg <= dest_mac[15 : 8];
    end
    DEST_MAC5: begin
      cursor <= SOURCE_MAC0;
      data_reg <= dest_mac[7 : 0];
    end
    SOURCE_MAC0: begin
      cursor <= SOURCE_MAC1;
      data_reg <= dest_mac[47 : 40];
    end
    SOURCE_MAC1: begin
      cursor <= SOURCE_MAC2;
      data_reg <= dest_mac[39 : 32];
    end
    SOURCE_MAC2: begin
      cursor <= SOURCE_MAC3;
      data_reg <= dest_mac[31 : 24];
    end
    SOURCE_MAC3: begin
      cursor <= SOURCE_MAC4;
      data_reg <= dest_mac[23 : 16];
    end
    SOURCE_MAC4: begin
      cursor <= SOURCE_MAC5;
      data_reg <= dest_mac[15 : 8];
    end
    SOURCE_MAC5: begin
      cursor <= ETH_TAG0;
      data_reg <= dest_mac[7 : 0];
    end
    ETH_TAG0: begin
      cursor <= ETH_TAG1;
      data_reg <= tpid[31 : 24];
    end
    ETH_TAG1: begin
      cursor <= ETH_TAG2;
      data_reg <= tpid[23 : 16];
    end
    ETH_TAG2: begin
      cursor <= ETH_TAG3;
      data_reg <= tpid[15 : 8];
    end
    ETH_TAG3: begin
      cursor <= ETH_TYPE0;
      data_reg <= tpid[7 : 0];
    end
```

```
    ETH_TYPE0: begin
      cursor <= ETH_TYPE1;
      data_reg <= ether_type[15 : 8];
    end
    ETH_TYPE1: begin
      cursor <= FCS0;
      data_reg <= ether_type[7 : 0];
    end
    FCS0: begin
      cursor <= FCS1;
      data_reg <= fcs[31 : 24];
    end
    FCS1: begin
      cursor <= FCS2;
      data_reg <= fcs[23 : 16];
    end
    FCS2: begin
      cursor <= FCS3;
      data_reg <= fcs[15 : 8];
    end
    FCS3: begin
      cursor <= IP_VER_IHL;
      data_reg <= fcs[7 : 0];
    end
   //IPv4
   IP_VER_IHL: begin
        cursor <= IP_DSCP_ECN;
        data_reg <= {version, ihl};
        case(protocol)
            _TCP: total_length <= (ihl * 4) + (data_offset * 4) + data_length;
            _UDP: total_length <= (ihl * 4) + 8 + data_length;
        endcase
   end
   IP_DSCP_ECN: begin
        cursor <= IP_LEN0;
        data_reg <= {dscp, ecn};
   end
   IP_LEN0: begin
        cursor <= IP_LEN1;
        data_reg <= total_length[15 : 8];
   end
   IP_LEN1: begin
        cursor <= IP_ID0;
        data_reg <= total_length[7 : 0];
   end
   IP_ID0: begin
        cursor <= IP_ID1;
        data_reg <= identification[15 : 8];
   end
   IP_ID1: begin
        cursor <= IP_FLAGS_FRAG0;
        data_reg <= identification[7 : 0];
   end
   IP_FLAGS_FRAG0: begin
        cursor <= IP_FRAG1;
        data_reg <= {ipv4_flags, fragment_offset[12 : 8]};
   end
```

```
IP_FRAG1: begin
    cursor <= IP_TTL;
    data_reg <= fragment_offset[7 : 0];
end
IP_TTL: begin
    cursor <= IP_PROTOCOL;
    data_reg <= ttl;
end
IP_PROTOCOL: begin
    cursor <= IP_CHECK0;
    data_reg <= protocol;
end
IP_CHECK0: begin
    cursor <= IP_CHECK1;
    data_reg <= header_checksum[15 : 8];
end
IP_CHECK1: begin
    cursor <= IP_SOURCE_ADDR0;
    data_reg <= header_checksum[7 : 0];
end
IP_SOURCE_ADDR0: begin
    cursor <= IP_SOURCE_ADDR1;
    data_reg <= source_address[31 : 24];
end
IP_SOURCE_ADDR1: begin
    cursor <= IP_SOURCE_ADDR2;
    data_reg <= source_address[23 : 16];
end
IP_SOURCE_ADDR2: begin
    cursor <= IP_SOURCE_ADDR3;
    data_reg <= source_address[15 : 8];
end
IP_SOURCE_ADDR3: begin
    cursor <= IP_DEST_ADDR0;
    data_reg <= source_address[7 : 0];
end
IP_DEST_ADDR0: begin
    cursor <= IP_DEST_ADDR1;
    data_reg <= dest_address[31 : 24];
end
IP_DEST_ADDR1: begin
    cursor <= IP_DEST_ADDR2;
    data_reg <= dest_address[23 : 16];
end
IP_DEST_ADDR2: begin
    cursor <= IP_DEST_ADDR3;
    data_reg <= dest_address[15 : 8];
end
IP_DEST_ADDR3: begin
    data_reg <= dest_address[7 : 0];
    case(protocol)
        _TCP: cursor <= TCP_SOURCE_PORT0;
        _UDP: cursor <= UDP_SOURCE_PORT0;
    endcase
end
//TCP
TCP_SOURCE_PORT0: begin
```

```
        cursor <= TCP_SOURCE_PORT1;
        data_reg <= source_port[15 : 8];
    end
    TCP_SOURCE_PORT1: begin
        cursor <= TCP_DEST_PORT0;
        data_reg <= source_port[7 : 0];
    end
    TCP_DEST_PORT0: begin
        cursor <= TCP_DEST_PORT1;
        data_reg <= dest_port[15 : 8];
    end
    TCP_DEST_PORT1: begin
        cursor <= TCP_SEQ0;
        data_reg <= dest_port[7 : 0];
    end
    TCP_SEQ0: begin
        cursor <= TCP_SEQ1;
        data_reg <= sequence_number[31 : 24];
    end
    TCP_SEQ1: begin
        cursor <= TCP_SEQ2;
        data_reg <= sequence_number[23 : 16];
    end
    TCP_SEQ2: begin
        cursor <= TCP_SEQ3;
        data_reg <= sequence_number[15 : 8];
    end
    TCP_SEQ3: begin
        cursor <= TCP_ACK0;
        data_reg <= sequence_number[7 : 0];
    end
    TCP_ACK0: begin
        cursor <= TCP_ACK1;
        data_reg <= acknowledgment_number[31 : 24];
    end
    TCP_ACK1: begin
        cursor <= TCP_ACK2;
        data_reg <= acknowledgment_number[23 : 16];
    end
    TCP_ACK2: begin
        cursor <= TCP_ACK3;
        data_reg <= acknowledgment_number[15 : 8];
    end
    TCP_ACK3: begin
        cursor <= TCP_OFFSET_FLAGS0;
        data_reg <= acknowledgment_number[7 : 0];
    end
    TCP_OFFSET_FLAGS0: begin
        cursor <= TCP_FLAGS1;
        data_reg <= {data_offset, reserved, tcp_flags[8 : 8]};
    end
    TCP_FLAGS1: begin
        cursor <= TCP_WINDOW0;
        data_reg <= tcp_flags[7 : 0];
    end
    TCP_WINDOW0: begin
        cursor <= TCP_WINDOW1;
```

```
        data_reg <= window_size[15 : 8];
    end
    TCP_WINDOW1: begin
        cursor <= TCP_CHECK0;
        data_reg <= window_size[7 : 0];
    end
    TCP_CHECK0: begin
        cursor <= TCP_CHECK1;
        data_reg <= checksum[15 : 8];
    end
    TCP_CHECK1: begin
        cursor <= TCP_URG0;
        data_reg <= checksum[7 : 0];
    end
    TCP_URG0: begin
        cursor <= TCP_URG1;
        data_reg <= urgent_pointer[15 : 8];
    end
    TCP_URG1: begin
        cursor <= _DATA;
        data_reg <= urgent_pointer[7 : 0];
    end
    //UDP
    UDP_SOURCE_PORT0: begin
        cursor <= UDP_SOURCE_PORT1;
        data_reg <=  source_port[15 : 8];
        udp_length <= data_length + 8;
    end
    UDP_SOURCE_PORT1: begin
        cursor <= UDP_DEST_PORT0;
        data_reg <= source_port[7 : 0];
    end
    UDP_DEST_PORT0: begin
        cursor <= UDP_DEST_PORT1;
        data_reg <= dest_port[15 : 8];
    end
    UDP_DEST_PORT1: begin
        cursor <= UDP_LEN0;
        data_reg <= dest_port[7 : 0];
    end
    UDP_LEN0: begin
        cursor <= UDP_LEN1;
        data_reg <= udp_length[15 : 8];
    end
    UDP_LEN1: begin
        cursor <= UDP_CHECK0;
        data_reg <= udp_length[7 : 0];
    end
    UDP_CHECK0: begin
        cursor <= UDP_CHECK1;
        data_reg <= checksum[15 : 8];
    end
    UDP_CHECK1: begin
        cursor <= _DATA;
        data_reg <= checksum[7 : 0];
    end
    //Payload
```

```
    _DATA: begin
        if(total_length == 0) begin // no more data
            cursor <= START;
            sending_reg <= 1'b0;
        end else begin
            total_length <= total_length - 1; // decrement data bytes count
            data_reg <= 8'hff;
        end
    end

  endcase
 end
end
```

Annexe n°12

```
module ip_parser #
(
)
(
input wire clk,
//input wire sync
input wire [7 : 0] data_stream,
output wire [7 : 0] passthrough,
input wire incomming,
input wire [15 : 0] warning_port,
input wire [31 : 0] warning_address,

output wire data_valid,
output wire [31 : 0] o_source_address,
output wire [31 : 0] o_dest_address,
output wire [15 : 0] o_source_port,
output wire [15 : 0] o_dest_port,
output wire [7 : 0] o_protocol,
output wire [1 : 0] leds
);

// 1 byte states
localparam [7 : 0] IDLE = 8'd0;
//ETH
localparam [7 : 0] PREAMBLE = 8'd1;
localparam [7 : 0] DEST_MAC0 = 8'd2;
localparam [7 : 0] DEST_MAC1 = 8'd3;
localparam [7 : 0] DEST_MAC2 = 8'd4;
localparam [7 : 0] DEST_MAC3 = 8'd5;
localparam [7 : 0] DEST_MAC4 = 8'd6;
localparam [7 : 0] DEST_MAC5 = 8'd7;
localparam [7 : 0] SOURCE_MAC0 = 8'd8;
localparam [7 : 0] SOURCE_MAC1 = 8'd9;
localparam [7 : 0] SOURCE_MAC2 = 8'd10;
localparam [7 : 0] SOURCE_MAC3 = 8'd11;
localparam [7 : 0] SOURCE_MAC4 = 8'd12;
localparam [7 : 0] SOURCE_MAC5 = 8'd13;
localparam [7 : 0] ETH_TAG0 = 8'd14;
localparam [7 : 0] ETH_TAG1 = 8'd15;
localparam [7 : 0] ETH_TAG2 = 8'd16;
localparam [7 : 0] ETH_TAG3 = 8'd17;
localparam [7 : 0] ETH_TYPE0 = 8'd18;
localparam [7 : 0] ETH_TYPE1 = 8'd19;
localparam [7 : 0] FCS0 = 8'd20;
localparam [7 : 0] FCS1 = 8'd21;
localparam [7 : 0] FCS2 = 8'd22;
localparam [7 : 0] FCS3 = 8'd23;
//IPv4
localparam [7 : 0] IP_VER_IHL = 8'd24;
localparam [7 : 0] IP_DSCP_ECN = 8'd25;
localparam [7 : 0] IP_LEN0 = 8'd26;
localparam [7 : 0] IP_LEN1 = 8'd27;
localparam [7 : 0] IP_ID0 = 8'd28;
localparam [7 : 0] IP_ID1 = 8'd29;
localparam [7 : 0] IP_FLAGS_FRAG0 = 8'd30;
```

```
localparam [7 : 0] IP_FRAG1 = 8'd31;
localparam [7 : 0] IP_TTL = 8'd32;
localparam [7 : 0] IP_PROTOCOL = 8'd33;
localparam [7 : 0] IP_CHECK0 = 8'd34;
localparam [7 : 0] IP_CHECK1 = 8'd35;
localparam [7 : 0] IP_SOURCE_ADDR0 = 8'd36;
localparam [7 : 0] IP_SOURCE_ADDR1 = 8'd37;
localparam [7 : 0] IP_SOURCE_ADDR2 = 8'd38;
localparam [7 : 0] IP_SOURCE_ADDR3 = 8'd39;
localparam [7 : 0] IP_DEST_ADDR0 = 8'd40;
localparam [7 : 0] IP_DEST_ADDR1 = 8'd41;
localparam [7 : 0] IP_DEST_ADDR2 = 8'd42;
localparam [7 : 0] IP_DEST_ADDR3 = 8'd43;
localparam [7 : 0] IP_OPTIONS0 = 8'd44;
localparam [7 : 0] IP_OPTIONS1 = 8'd45;
localparam [7 : 0] IP_OPTIONS2 = 8'd46;
localparam [7 : 0] IP_OPTIONS3 = 8'd47;
//TCP
localparam [7 : 0] TCP_SOURCE_PORT0 = 8'd48;
localparam [7 : 0] TCP_SOURCE_PORT1 = 8'd49;
localparam [7 : 0] TCP_DEST_PORT0 = 8'd50;
localparam [7 : 0] TCP_DEST_PORT1 = 8'd51;
localparam [7 : 0] TCP_SEQ0 = 8'd52;
localparam [7 : 0] TCP_SEQ1 = 8'd53;
localparam [7 : 0] TCP_SEQ2 = 8'd54;
localparam [7 : 0] TCP_SEQ3 = 8'd55;
localparam [7 : 0] TCP_ACK0 = 8'd56;
localparam [7 : 0] TCP_ACK1 = 8'd57;
localparam [7 : 0] TCP_ACK2 = 8'd58;
localparam [7 : 0] TCP_ACK3 = 8'd59;
localparam [7 : 0] TCP_OFFSET_FLAGS0 = 8'd60;
localparam [7 : 0] TCP_FLAGS1 = 8'd61;
localparam [7 : 0] TCP_WINDOW0 = 8'd62;
localparam [7 : 0] TCP_WINDOW1 = 8'd63;
localparam [7 : 0] TCP_CHECK0 = 8'd64;
localparam [7 : 0] TCP_CHECK1 = 8'd65;
localparam [7 : 0] TCP_URG0 = 8'd66;
localparam [7 : 0] TCP_URG1 = 8'd67;
localparam [7 : 0] TCP_OPTIONS0 = 8'd68;
localparam [7 : 0] TCP_OPTIONS1 = 8'd69;
localparam [7 : 0] TCP_OPTIONS2 = 8'd70;
localparam [7 : 0] TCP_OPTIONS3 = 8'd71;
//UDP
localparam [7 : 0] UDP_SOURCE_PORT0 = 8'd72;
localparam [7 : 0] UDP_SOURCE_PORT1 = 8'd73;
localparam [7 : 0] UDP_DEST_PORT0 = 8'd74;
localparam [7 : 0] UDP_DEST_PORT1 = 8'd75;
localparam [7 : 0] UDP_LEN0 = 8'd76;
localparam [7 : 0] UDP_LEN1 = 8'd77;
localparam [7 : 0] UDP_CHECK0 = 8'd78;
localparam [7 : 0] UDP_CHECK1 = 8'd79;
//Payload
localparam [7 : 0] _DATA = 8'd80;
//Protocol
localparam [7 : 0] _TCP = 8'd6;
localparam [7 : 0] _UDP = 8'd17;
```

```
reg [31 : 0] source_address;
reg [31 : 0] dest_address;
reg [15 : 0] source_port;
reg [15 : 0] dest_port;

reg [31 : 0] o_source_address_reg;
reg [31 : 0] o_dest_address_reg;
reg [15 : 0] o_source_port_reg;
reg [15 : 0] o_dest_port_reg;
reg [7 : 0] o_protocol_reg;
reg data_valid_reg = 1'b0;
reg [1 : 0] leds_reg = 2'b00;

assign o_source_address = o_source_address_reg;
assign o_dest_address = o_dest_address_reg;
assign o_source_port = o_source_port_reg;
assign o_dest_port = o_dest_port_reg;
assign o_protocol = o_protocol_reg;
assign data_valid = data_valid_reg;
assign leds = leds_reg;

reg [3 : 0] ihl;
reg [15 : 0] total_length;
reg [3 : 0] tcp_headers;
reg [15 : 0] udp_length;
reg [7 : 0] protocol;

reg [7 : 0] cursor = 8'd0;
reg acquiring = 1'b0;
reg [7 : 0] passthrough_reg = 8'b0;
reg [15 : 0] warning_port_reg = 15'b0;
reg [31 : 0] warning_address_reg = 31'b0;

assign passthrough = passthrough_reg;

always @ (posedge clk) begin
   if(incomming) begin
       leds_reg[1 : 1] <= 1'b1;
   end else begin
       leds_reg[1 : 1] <= 1'b0;
   end
end

always @ (posedge clk) begin
   if(cursor == IDLE) begin
       leds_reg[0 : 0] <= 1'b1;
   end else begin
       leds_reg[0 : 0] <= 1'b0;
   end
end

always @ (posedge clk) begin
   if(warning_port_reg != warning_port) begin
       warning_port_reg <= warning_port;
   end
   if(warning_address_reg != warning_address) begin
       warning_address_reg <= warning_address;
```

```
   end
end

always @ (posedge clk) begin
   if(incomming || acquiring) begin
       case (cursor)
           IDLE: begin
               cursor <= (incomming) ? PREAMBLE : IDLE; //initial state, waiting for a packet
               data_valid_reg <= 1'b0;
               ihl <= 4'b0;
               total_length <= 16'b0;
               tcp_headers <= 4'b0;
               udp_length <= 16'b0;
               protocol <= 8'b0;
               source_address <= 32'b0;
               dest_address <= 32'b0;
               source_port <= 32'b0;
               dest_port <= 32'b0;
           end
           //ETH
            PREAMBLE: begin
               if(data_stream == 8'b10101010) begin
                     cursor <= PREAMBLE;
               end else begin
                   cursor <= DEST_MAC0;
               end
                passthrough_reg <= data_stream;
            end
            DEST_MAC0: begin
                cursor <= DEST_MAC1;
                passthrough_reg <= data_stream;
            end
            DEST_MAC1: begin
                cursor <= DEST_MAC2;
                passthrough_reg <= data_stream;
            end
            DEST_MAC2: begin
                cursor <= DEST_MAC3;
                passthrough_reg <= data_stream;
            end
            DEST_MAC3: begin
                cursor <= DEST_MAC4;
                passthrough_reg <= data_stream;
            end
            DEST_MAC4: begin
                cursor <= DEST_MAC5;
                passthrough_reg <= data_stream;
            end
            DEST_MAC5: begin
                cursor <= SOURCE_MAC0;
                passthrough_reg <= data_stream;
            end
            SOURCE_MAC0: begin
                cursor <= SOURCE_MAC1;
                passthrough_reg <= data_stream;
            end
            SOURCE_MAC1: begin
```

```
            cursor <= SOURCE_MAC2;
            passthrough_reg <= data_stream;
        end
        SOURCE_MAC2: begin
            cursor <= SOURCE_MAC3;
            passthrough_reg <= data_stream;
        end
        SOURCE_MAC3: begin
            cursor <= SOURCE_MAC4;
            passthrough_reg <= data_stream;
        end
        SOURCE_MAC4: begin
            cursor <= SOURCE_MAC5;
            passthrough_reg <= data_stream;
        end
        SOURCE_MAC5: begin
            cursor <= ETH_TAG0;
            passthrough_reg <= data_stream;
        end
        ETH_TAG0: begin
            cursor <= ETH_TAG1;
            passthrough_reg <= data_stream;
        end
        ETH_TAG1: begin
            cursor <= ETH_TAG2;
            passthrough_reg <= data_stream;
        end
        ETH_TAG2: begin
            cursor <= ETH_TAG3;
            passthrough_reg <= data_stream;
        end
        ETH_TAG3: begin
            cursor <= ETH_TYPE0;
            passthrough_reg <= data_stream;
        end
        ETH_TYPE0: begin
            cursor <= ETH_TYPE1;
            passthrough_reg <= data_stream;
        end
        ETH_TYPE1: begin
            cursor <= FCS0;
            passthrough_reg <= data_stream;
        end
        FCS0: begin
            cursor <= FCS1;
            passthrough_reg <= data_stream;
        end
        FCS1: begin
            cursor <= FCS2;
            passthrough_reg <= data_stream;
        end
        FCS2: begin
            cursor <= FCS3;
            passthrough_reg <= data_stream;
        end
        FCS3: begin
            cursor <= IP_VER_IHL;
```

```
            passthrough_reg <= data_stream;
        end
        //IPv4
        IP_VER_IHL: begin
            cursor <= IP_DSCP_ECN;
            ihl <= data_stream[3 : 0];
            passthrough_reg <= data_stream;
        end
        IP_DSCP_ECN: begin
            cursor <= IP_LEN0;
            passthrough_reg <= data_stream;
        end
        IP_LEN0: begin
            cursor <= IP_LEN1;
            total_length[15 : 8] <= data_stream;
            passthrough_reg <= data_stream;
        end
        IP_LEN1: begin
            cursor <= IP_ID0;
            total_length[7 : 0] <= data_stream;
            passthrough_reg <= data_stream;
        end
        IP_ID0: begin
            cursor <= IP_ID1;
            total_length <= total_length - (ihl * 4); // substract ip headers
            passthrough_reg <= data_stream;
        end
        IP_ID1: begin
            cursor <= IP_FLAGS_FRAG0;
            passthrough_reg <= data_stream;
        end
        IP_FLAGS_FRAG0: begin
            cursor <= IP_FRAG1;
            passthrough_reg <= data_stream;
        end
        IP_FRAG1: begin
            cursor <= IP_TTL;
            passthrough_reg <= data_stream;
        end
        IP_TTL: begin
            cursor <= IP_PROTOCOL;
            passthrough_reg <= data_stream;
        end
        IP_PROTOCOL: begin
            cursor <= IP_CHECK0;
            protocol <= data_stream;
            passthrough_reg <= data_stream;
        end
        IP_CHECK0: begin
            cursor <= IP_CHECK1;
            passthrough_reg <= data_stream;
        end
        IP_CHECK1: begin
            cursor <= IP_SOURCE_ADDR0;
            passthrough_reg <= data_stream;
        end
        IP_SOURCE_ADDR0: begin
```

```
            cursor <= IP_SOURCE_ADDR1;
            source_address[31 : 24] <= data_stream;
            passthrough_reg <= data_stream;
        end
        IP_SOURCE_ADDR1: begin
            cursor <= IP_SOURCE_ADDR2;
            source_address[23 : 16] <= data_stream;
            passthrough_reg <= data_stream;
        end
        IP_SOURCE_ADDR2: begin
            cursor <= IP_SOURCE_ADDR3;
            source_address[15 : 8] <= data_stream;
            passthrough_reg <= data_stream;
        end
        IP_SOURCE_ADDR3: begin
            cursor <= IP_DEST_ADDR0;
            source_address[7 : 0] <= data_stream;
            ihl <= ihl - 5; // get options length
            passthrough_reg <= data_stream;
        end
        IP_DEST_ADDR0: begin
            cursor <= IP_DEST_ADDR1;
            dest_address[31 : 24] <= data_stream;
            passthrough_reg <= data_stream;
        end
        IP_DEST_ADDR1: begin
            cursor <= IP_DEST_ADDR2;
            dest_address[23 : 16] <= data_stream;
            passthrough_reg <= data_stream;
        end
        IP_DEST_ADDR2: begin
            cursor <= IP_DEST_ADDR3;
            dest_address[15 : 8] <= data_stream;
            passthrough_reg <= data_stream;
        end
        IP_DEST_ADDR3: begin
            if({dest_address[31 : 8], data_stream} == warning_address_reg) begin
                cursor <= IDLE;
            end else begin
                dest_address[7 : 0] <= data_stream;
                passthrough_reg <= data_stream;
                if(ihl == 0) begin // no options
                    case(protocol)
                        _TCP: cursor <= TCP_SOURCE_PORT0;
                        _UDP: cursor <= UDP_SOURCE_PORT0;
                        default: cursor <= IDLE;
                    endcase
                end else begin
                    cursor <= IP_OPTIONS0;
                end
            end
        end
        IP_OPTIONS0: begin
            cursor <= IP_OPTIONS1;
            ihl <= ihl - 1;
            passthrough_reg <= data_stream;
        end
```

```
IP_OPTIONS1: begin
    cursor <= IP_OPTIONS2;
    passthrough_reg <= data_stream;
end
IP_OPTIONS2: begin
    cursor <= IP_OPTIONS3;
    passthrough_reg <= data_stream;
end
IP_OPTIONS3: begin
    if(ihl == 0) begin // no more options
        case(protocol)
            _TCP: cursor <= TCP_SOURCE_PORT0;
            _UDP: cursor <= UDP_SOURCE_PORT0;
            default: cursor <= IDLE;
        endcase
    end else begin
        cursor <= IP_OPTIONS0;
    end
    passthrough_reg <= data_stream;
end
//TCP
TCP_SOURCE_PORT0: begin
    cursor <= TCP_SOURCE_PORT1;
    source_port[15 : 8] <= data_stream;
    passthrough_reg <= data_stream;
end
TCP_SOURCE_PORT1: begin
    cursor <= TCP_DEST_PORT0;
    source_port[7 : 0] <= data_stream;
    passthrough_reg <= data_stream;
end
TCP_DEST_PORT0: begin
    cursor <= TCP_DEST_PORT1;
    dest_port[15 : 8] <= data_stream;
    passthrough_reg <= data_stream;
end
TCP_DEST_PORT1: begin
    if({dest_port[15 : 8],data_stream} == warning_address_reg) begin
        cursor <= IDLE;
    end else begin
        cursor <= TCP_SEQ0;
        dest_port[7 : 0] <= data_stream;
        passthrough_reg <= data_stream;
    end
end
TCP_SEQ0: begin
    cursor <= TCP_SEQ1;
    passthrough_reg <= data_stream;
end
TCP_SEQ1: begin
    cursor <= TCP_SEQ2;
    passthrough_reg <= data_stream;
end
TCP_SEQ2: begin
    cursor <= TCP_SEQ3;
    passthrough_reg <= data_stream;
end
```

```
        TCP_SEQ3: begin
            cursor <= TCP_ACK0;
            passthrough_reg <= data_stream;
        end
        TCP_ACK0: begin
            cursor <= TCP_ACK1;
            passthrough_reg <= data_stream;
        end
        TCP_ACK1: begin
            cursor <= TCP_ACK2;
            passthrough_reg <= data_stream;
        end
        TCP_ACK2: begin
            cursor <= TCP_ACK3;
            passthrough_reg <= data_stream;
        end
        TCP_ACK3: begin
            cursor <= TCP_OFFSET_FLAGS0;
            passthrough_reg <= data_stream;
        end
        TCP_OFFSET_FLAGS0: begin
            cursor <= TCP_FLAGS1;
            tcp_headers <= data_stream[7 : 4];
            passthrough_reg <= data_stream;
        end
        TCP_FLAGS1: begin
            cursor <= TCP_WINDOW0;
            total_length <= total_length - (tcp_headers * 4); // get data length
            passthrough_reg <= data_stream;
        end
        TCP_WINDOW0: begin
            cursor <= TCP_WINDOW1;
            passthrough_reg <= data_stream;
        end
        TCP_WINDOW1: begin
            cursor <= TCP_CHECK0;
            passthrough_reg <= data_stream;
        end
        TCP_CHECK0: begin
            cursor <= TCP_CHECK1;
            passthrough_reg <= data_stream;
        end
        TCP_CHECK1: begin
            cursor <= TCP_URG0;
            tcp_headers <= tcp_headers - 5; // get options length
            passthrough_reg <= data_stream;
        end
        TCP_URG0: begin
            cursor <= TCP_URG1;
            passthrough_reg <= data_stream;
        end
        TCP_URG1: begin
            if(tcp_headers == 0) begin // no options
                cursor <= _DATA;
            end else begin
                cursor <= TCP_OPTIONS0;
            end
```

```
            passthrough_reg <= data_stream;
        end
        TCP_URG1: begin
            cursor <= _DATA;
            passthrough_reg <= data_stream;
        end
        TCP_OPTIONS0: begin
            cursor <= TCP_OPTIONS1;
            tcp_headers <= tcp_headers - 1;
            passthrough_reg <= data_stream;
        end
        TCP_OPTIONS1: begin
            cursor <= TCP_OPTIONS2;
            passthrough_reg <= data_stream;
        end
        TCP_OPTIONS2: begin
            cursor <= TCP_OPTIONS3;
            passthrough_reg <= data_stream;
        end
        TCP_OPTIONS3: begin
            passthrough_reg <= data_stream;
            if(tcp_headers == 0) begin // no more options
                cursor <= _DATA;
            end else begin
                cursor <= TCP_OPTIONS0;
            end
        end
        //UDP
        UDP_SOURCE_PORT0: begin
            cursor <= UDP_SOURCE_PORT1;
            source_port[15 : 8] <= data_stream;
            passthrough_reg <= data_stream;
        end
        UDP_SOURCE_PORT1: begin
            cursor <= UDP_DEST_PORT0;
            source_port[7 : 0] <= data_stream;
            passthrough_reg <= data_stream;
        end
        UDP_DEST_PORT0: begin
            cursor <= UDP_DEST_PORT1;
            dest_port[15 : 8] <= data_stream;
            passthrough_reg <= data_stream;
        end
        UDP_DEST_PORT1: begin
            if({dest_port[15 : 8],data_stream} == warning_address_reg) begin
                cursor <= IDLE;
            end else begin
                cursor <= UDP_LEN0;
                dest_port[7 : 0] <= data_stream;
                passthrough_reg <= data_stream;
            end
        end
        UDP_LEN0: begin
            cursor <= UDP_LEN1;
            udp_length[15 : 8] <= data_stream;
            passthrough_reg <= data_stream;
        end
```

```
            UDP_LEN1: begin
                cursor <= UDP_CHECK0;
                udp_length[7 : 0] <= data_stream;
                passthrough_reg <= data_stream;
            end
            UDP_CHECK0: begin
                cursor <= UDP_CHECK1;
                total_length <= udp_length - 8; // get data length
                passthrough_reg <= data_stream;
            end
            UDP_CHECK1: begin
                cursor <= _DATA;
                passthrough_reg <= data_stream;
            end
            //Payload
            _DATA: begin
                if(total_length == 0) begin
                    cursor <= IDLE;
                    o_source_address_reg <= source_address;
                    o_dest_address_reg <= dest_address;
                    o_source_port_reg <= source_port;
                    o_dest_port_reg <= dest_port;
                    o_protocol_reg <= protocol;
                    data_valid_reg <= 1'b1;
                    acquiring <= 1'b0;
                end else begin
                    total_length <= total_length - 1;
                    passthrough_reg <= data_stream;
                end
            end
            default: cursor <= IDLE;
        endcase
    end else begin
        data_valid_reg <= 1'b0;
    end
end

endmodule
```

Annexe n°13

```
localparam [31 : 0] _ADDR = 32'h61646472;
localparam [31 : 0] _PORT = 32'h706f7274;

reg [31 : 0] threshold_value = 32'hffffffff;
reg [15 : 0] port0 = 16'h0;
reg [15 : 0] port1 = 16'h0;
reg [15 : 0] port2 = 16'h0;
reg [31 : 0] address0 = 32'h0;
reg [31 : 0] address1 = 32'h0;
reg [31 : 0] address2 = 32'h0;
reg [31 : 0] port0_counter = 32'h0;
reg [31 : 0] port1_counter = 32'h0;
reg [31 : 0] port2_counter = 32'h0;
reg [31 : 0] address0_counter = 32'h0;
reg [31 : 0] address1_counter = 32'h0;
reg [31 : 0] address2_counter = 32'h0;
reg interrupt = 1'b0;

assign INTERRUPT = interrupt;
assign O_WARNING_PORT = warning_port;
assign O_WARNING_ADRRESS = warning_address;

integer counter = 0;

always @( posedge S_AXI_ACLK ) begin
  if(threshold_value != threshold) begin
    threshold_value <= threshold;
  end
  if(port0 != protected_port0[15 : 0]) begin
    port0 <= protected_port0[15 : 0];
  end
  if(port1 != protected_port1[15 : 0]) begin
    port1 <= protected_port1[15 : 0];
  end
  if(port2 != protected_port2[15 : 0]) begin
    port2 <= protected_port2[15 : 0];
  end
  if(address0 != protected_address0) begin
    address0 <= protected_address0;
  end
  if(address1 != protected_address1) begin
    address1 <= protected_address1;
  end
  if(address2[31 : 0] != protected_address2) begin
    address2 <= protected_address2;
  end
end

always @( posedge S_AXI_ACLK ) begin
  if(counter < 1000000000) begin // 10 sec
    counter = counter + 1;
    if(DATA_VALID) begin
      case(DEST_PORT)
```

```
        port0: begin
          if(warning_port == port0) begin
            port0_counter <= 32'b0;
          end else begin
            port0_counter <= port0_counter + 1;
          end
        end
        port1: begin
          if(warning_port == port1) begin
            port1_counter <= 32'b0;
          end else begin
            port1_counter <= port1_counter + 1;
          end
        end
        port2: begin
          if(warning_port == port2) begin
            port2_counter <= 32'b0;
          end else begin
            port2_counter <= port2_counter + 1;
          end
        end
      endcase
      case(DEST_ADDRESS)
        address0: begin
          if(warning_address == address0) begin
            address0_counter <= 32'b0;
          end else begin
            address0_counter <= address0_counter + 1;
          end
        end
        address1: begin
          if(warning_address == address1) begin
            address1_counter <= 32'b0;
          end else begin
            address1_counter <= address1_counter + 1;
          end
        end
        address2: begin
          if(warning_address == address2) begin
            address2_counter <= 32'b0;
          end else begin
            address2_counter <= address2_counter + 1;
          end
        end
      endcase
    end
  end else begin
    counter = 0;
    port0_counter <= 32'b0;
    port1_counter <= 32'b0;
    port2_counter <= 32'b0;
    address0_counter <= 32'b0;
    address1_counter <= 32'b0;
```

```
    address2_counter <= 32'b0;
    warning_port <= 32'b0;
    warning_address <= 32'b0;
  end
end

always @( posedge S_AXI_ACLK ) begin
  if(port0_counter > threshold_value) begin
    interrupt <= 1'b1;
    warning_type <= _PORT;
    warning_port <= port0;
  end else if (port1_counter > threshold_value) begin
    interrupt <= 1'b1;
    warning_type <= _PORT;
    warning_port <= port1;
  end else if (port2_counter > threshold_value) begin
    interrupt <= 1'b1;
    warning_type <= _PORT;
    warning_port <= port2;
  end else if (address0_counter > threshold_value) begin
    interrupt <= 1'b1;
    warning_type <= _ADDR;
    warning_address <= address0;
  end else if (address1_counter > threshold_value) begin
    interrupt <= 1'b1;
    warning_type <= _ADDR;
    warning_address <= address1;
  end else if (address2_counter > threshold_value) begin
    interrupt <= 1'b1;
    warning_type <= _ADDR;
    warning_address <= address2;
  end else begin
    interrupt <= 1'b0;
    warning_port <= 32'b0;
    warning_address <= 32'b0;
  end
end
```