Rapport bibliographique

Projet 3A ; Filières IIS, SLR

2 décembre 2009

Implémentation FPGA d'algorithmes de surveillance de trafic

Projet n°11

Auteurs : Benoît FONTAINE, Tristan GROLÉAT, Franziska HUBERT

Table des matières

1. Introduction 3
2. NetFPGA 4
   2.1. Usages 5
   2.2. Le NetFPGA et notre projet 6
3. NetFlow 6
   3.1. Sonde NetFlow basée sur NetFPGA 6
   3.2. Qu'est-ce que NetFlow ? 6
   3.3. Architecture du système 8
   3.4. Firmware 8
   3.5. Parseur d'en-tête L3L4 9
   3.6. L'unité de Timestamp 10
   3.7. L'unité de hash 11
   3.8. L'unité FlowLookUp 12
   3.9. L'unité de traitement de flux 13
   3.10. L'unité Record Wrapper 14
   3.11. L'application logicielle 14
4. Les outils 15
   4.1. Xilinx 15
   4.2. ModelSim – Advanced Simulation and Debugging 16
   4.3. ChipScope 17
   4.4. SoCLib 18
   4.5. Verilog et VHDL 19
5. Les algorithmes 20
   5.1. Problématiques réseau 20
      5.1.1. Classification de services 20
      5.1.2. Détection d'attaques 20
      5.1.3. Attaques DoS et DDoS 21
      5.1.4. Syn Flooding 21
      5.1.5. Ping Flooding 21
      5.1.6. Attaques Teardrop 21
   5.2. Le Stream Mining 22
   5.3. L'algorithme CMS 22
      5.3.1. Le problème de l'inversion du hashage 24
   5.4. La mémoire Trie 24
   5.5. Notre position face aux implémentations existantes 25
6. Conclusion 26
7. Bibliographie 27

1. Introduction

Dans le cadre du projet de 3ème année, nous voulons implémenter sur un NetFPGA un algorithme d'analyse des flux de paquets IP sur un lien à très haut débit (100Gb/s) pour faire de la classification de services ou de la détection d'attaques sur un équipement de cœur de réseau (un routeur ou un pare-feu, c'est à dire un équipement dédié à la protection du réseau).

Le NetFPGA est une carte possédant un FPGA - c'est à dire une puce programmable - et qui gère 4 interfaces réseau Ethernet. Nous allons utiliser l'accélération matérielle dont il dispose pour permettre à notre système de fonctionner à 100Gb/s. L'étape suivante sera d'implémenter l'architecture programmée sur le FPGA en une puce dédiée, augmentant ainsi les capacités.

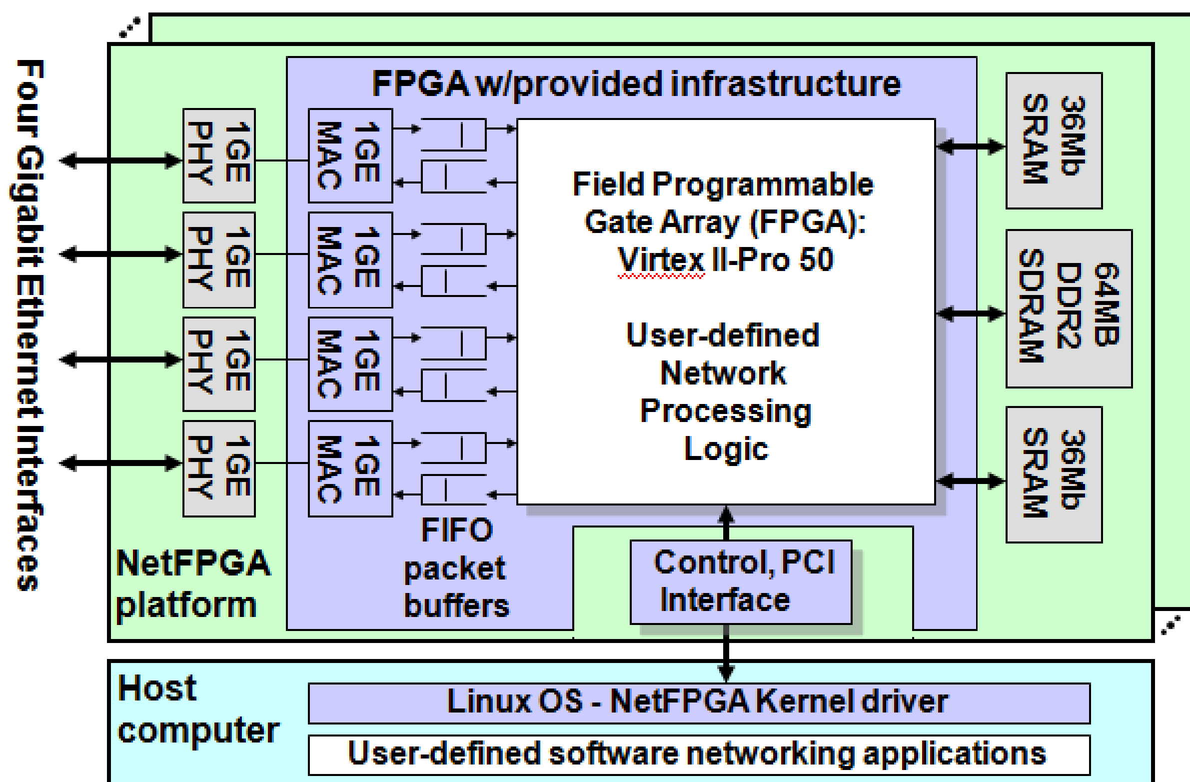
Le but du projet est de prendre en main le NetFPGA que TELECOM Bretagne vient d'acquérir, et de poser les bases pour implémenter des algorithmes de surveillance réseau avec accélération matérielle qui puissent fonctionner sur des réseaux à très haut débit.

Nous allons donc commencer par étudier ce qui existe déjà autour du NetFPGA, puis les outils de développement que nous pourrons utiliser. Enfin nous verrons quelles sont les problématiques réseau que nous pouvons résoudre et quels algorithmes existants peuvent nous y aider.

2. NetFPGA

Le NetFPGA est une carte PCI embarquant

- un large FPGA Xilinx Virtex2-Pro 50 embarquant la logique 'utilisateur' (horloge à 125MHz)
- un petit FPGA Xilinx Spartan II embarquant la logique permettant le contrôle de la carte depuis l'interface PCI
- 4 ports Gigabit Ethernet (Broadcom BCM5464SR)
- 4.5MB (2x18Mb) de SRAM (Static RAM)
- 64 MB de DDR2 DRAM (Double-Date Rate Dynamic RAM). (débit: 1,6 MB/s)
- 2 ports SATA (Serial ATA)

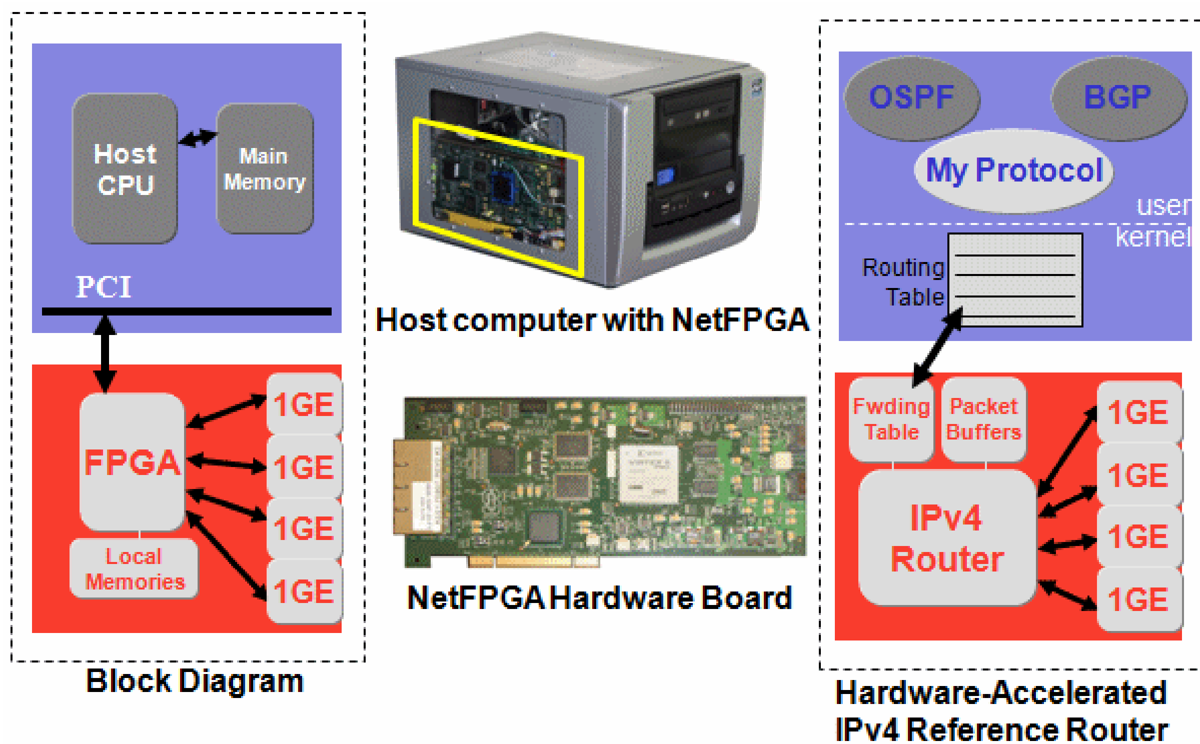


Graphique 1 : schéma du NetFPGA

Il a été conçu à l'univeristé de Stanford à Palo Alto, CA.

Le NetFPGA permet aux étudiants et chercheurs de réaliser des prototypes de systèmes réseau à haute vitesse et bénéficiant d'une accélération matérielle. Les étudiants de Stanford ont déjà travaillé à réaliser un routeur IP à l'aide du NetFPGA. Une présentation de leurs travaux est disponible sur le site officiel [NET08].
La carte NetFPGA possède une interface PCI standard et peut donc être connecté à un ordinateur de bureau ou un serveur. Le NetFPGA permet de décharger le processeur de la machine hôte du traitement des données. Le CPU de la machine hôte a accès à la mémoire principale et peut effectuer des opérations DMA de lecture et d'écriture des registres et mémoires du NetFPGA. Contrairement aux autres projets open-source le NetFPGA offre une approche matériellement accélérée. Le NetFPGA fournit une interface matérielle directement

connectée aux quatre ports Gigabit et à de multiples banques de mémoire installées sur la carte.



Graphique 2 : diagrammes de système du NetFPGA

2.1. Usages

Des paquets NetFPGA (NFPs) sont disponibles et contiennent du code source (aussi bien pour la partie matérielle que logicielle) implémentant des fonctions réseau. Le "routeur de référence" est un NFP. En utilisant ce routeur comme référence, un développeur a alors trois façons d'utiliser ce NFP.

- Une première méthode est de configurer le FPGA de la carte en routeur et ensuite de modifier la partie logicielle (programme utilisateur, pilote Linux). Dans ce scénario la carte NetFPGA est programmée avec le code IPv4 et l'ordinateur hôte Linux utilise le logiciel 'Router kit' inclut dans le NFP. Router kit clone la table de routage ainsi que le cache ARP de l'ordinateur dans les tables matérielles de la carte. Le développeur peut donc modifier Linux afin d'implémenter de nouveaux protocoles.

- Une deuxième approche est d'utiliser le code FPGA du routeur et d'en étendre les fonctionnalités en développant un module utilisateur. On commence donc avec le code fourni dans le NFP officiel puis on le modifie en utilisant les modules disponibles dans la bibliothèque du NFP ou en écrivant directement du code Verilog. Enfin on compile le code source à l'aide d'outils industriels de conception. Le fichier 'bitfile' ainsi créé peut être téléchargé dans le FPGA. La nouvelle fonctionnalité peut être complétée par du logiciel supplémentaire ou en modifiant le logiciel existant. Un exemple serait d'implémenter une recherche de type **Trie LPM** (Longest Prefix Match) au lieu du **CAM LPM** utilisé actuellement. Un autre exemple serait de modifier le routeur pour ajouter la gestion du NAT ou ajouter des fonctions de pare-feu.

- Finalement il est aussi possible de créer un design FPGA totalement nouveau et d'implémenter sa propre logique et ses propres fonctions de traitement des données directement dans le NetFPGA. En réécrivant le cœur du FPGA on perd la possibilité (il faut le réécrire) d'accéder directement à la mémoire du NetFPGA (SRAM et DDR RAM) depuis Linux.

2.2. Le NetFPGA et notre projet

A quel niveau allons-nous implémenter le code de surveillance ?

Le code étant complexe (data mining) il serait intéressant d'en implémenter le plus possible en software. Mais en même temps il faut profiter de l'accélération matérielle que nous procure le NetFPGA.

Dans le cas du routeur IP, la table de routage est gérée en logiciel, mais ce qui est couteux en temps, le routage, est lui effectué par la carte. La situation, pour nous, n'est pas aussi facile: si on décide d'implémenter le code d'analyse en logiciel il faut alors faire remonter les données du NetFPGA vers l'ordinateur. Or c'est précisément ce que l'on cherche à éviter car c'est ce qui consomme le plus de ressources. Il faudrait qu'un traitement de gros soit réalisé au niveau du NetFPGA pour ne faire remonter à l'ordinateur qu'une petite quantité de données.

La fonction routeur IP déjà disponible avec le NetFPGA nous est utile pour nous insérer en coupure de réseau et transmettre les paquets tout en les analysant. Il paraît donc logique de réutiliser le code FPGA déjà écrit et d'en étendre les fonctionnalités. C'est donc la seconde approche que nous allons utiliser.

Il est intéressant d'étudier d'autres projets utilisant le NetFPGA afin d'analyser leurs approches. Une liste des projets basés sur NetFPGA est disponible sur le site officiel. Dans cette liste le projet NetFlowProbe possède un fonctionnement proche de celui que nous voudrions réaliser. Il effectue en particulier de la surveillance réseau et implémente les fonctions critiques (c'est à dire consommatrices de temps) directement en matériel dans le FPGA.

3. NetFlow

3.1. Sonde NetFlow basée sur NetFPGA

Afin de nous familiariser avec le développement d'applications de surveillance réseau basées sur NetFPGA, il est important d'analyser les projets de ce domaine. Ainsi le projet NetFlow probe est basé sur NetFPGA et possède un fonctionnement proche de celui que nous voudrions réaliser.

3.2. Qu'est-ce que NetFlow ?

Avec un volume sans cesse croissant de données qui sont transférées sur Internet, la nécessité d'une surveillance fiable devient plus urgente. Dispositifs de surveillance devraient être en mesure de fournir des informations actualisées, telles que les modes de circulation, statistiques, et anomalies diverses.

NetFlow est un protocole réseau développé par Cisco Systems pour ses équipements pourvus d'un Cisco IOS afin de collecter les informations trafic IP. C'est un protocole propriétaire mais supporté par d'autres plateformes qu'IOS telles que Linux, FreeBSD ou OpenBSD. Les routeurs Cisco implémentant NetFlow génèrent des enregistrements netflow; ceux-ci sont en suite exportés depuis le routeur dans des paquets UDP ou SCTP et collectés par un collecteur Netflow.

Un flux réseau, tel que le définit Cisco, est une clef 7-tuple, où le flux est représenté par une suite de paquets unidirectionnels partageant tous les sept valeurs suivantes:

1. Adresse IP source
2. Adresse IP de destination
3. Port source pour l'UDP et TCP, 0 pour les autres protocoles
4. Port de destination pour les ports UDP et TCP, type et code pour l'ICMP ou 0 pour les autres protocoles
5. Protocole IP
6. Interface entrante
7. Valeur du ToS (Type of Service)

Le routeur envoie l'enregistrement du flux quand il estime que le flux est terminé. Ceci est détectable si aucun nouveau trafic n'est aperçu pour le flux existant ou si un paquet TCP de terminaison est envoyé.

Un enregistrement NetFlow contient de nombreuses informations à propos du flux concerné. Ainsi dans la version 5 du protocole (la plus utilisée), un enregistrement contient:

- Numéro de version
- Numéro de séquence
- Interface d'entrée et de sortie
- Le temps en millisecondes depuis le dernier démarrage du début et de fin du flux (*timestamp*)
- Nombre d'octets et de paquets observés dans le flux
- Les en-têtes niveau 3:
    - IP source et destination
    - Port source et destination
    - Numéro de protocole IP
    - Valeur du ToS (Type of Service)
- Dans le cas d’un flux TCP, l'union de tous les drapeaux TCP observés durant la vie du flux
- Les informations de routages niveau 3:
    - Adresse IP du "next hop" immédiat dans la route vers la destination.
    - Masques IP source et destination (les longueurs de préfixe dans la notation CIDR)

Ce projet décrit l'implémentation d'un système de surveillance de flux réseau utilisant une plateforme matérielle dédiée (le NetFPGA) coopérant avec le PC hôte. Les fonctions critiques sont ainsi implémentées directement dans le matériel et le reste en logiciel. De cette façon on obtient une sonde NetFlow performante à faible coût.

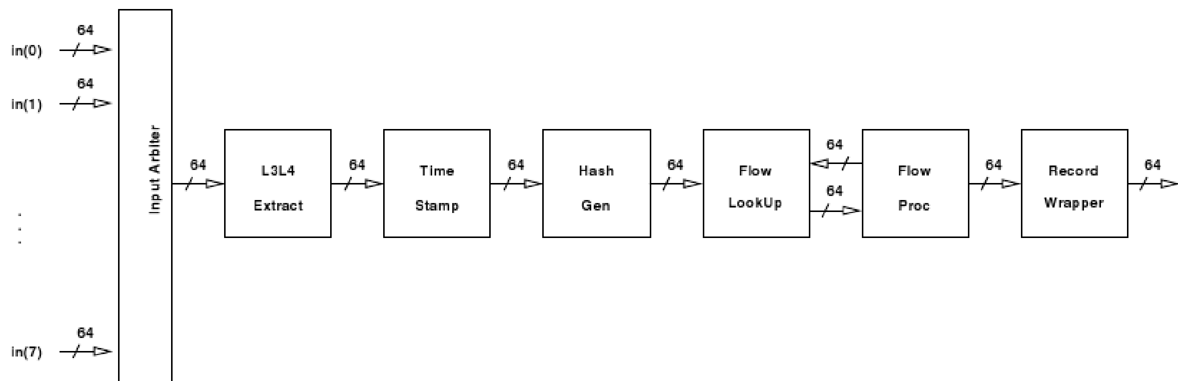
3.3. Architecture du système

La sonde NetFlow est contituée d'un ordinateur hôte et d'une carte NetFPGA. La mesure des flux est entièrement implémentée dans le NetFPGA tandis que le contrôle, la configuration et la collection sont implémentés dans un logiciel tournant sur l'ordinateur hôte. Les principaux paramètres de la sonde sont les suivants :

- Mesure en temps réel des flux des quatre interfaces réseau Gigabit
- Mémoire permettant d'accueillir 60000 flux différents
- Indexation des enregistrements de flux à l'aide d'un hash et de 8 recherches parallèles
- Export des enregistrements au format NetFlow v5

3.4. Firmware

Le firmware de la sonde NetFPGA est composé de différentes unités chaînées. Chaque unité du pipeline possède une tâche spécifique qui consiste la plupart du temps à traiter les données arrivantes et ajouter le résultat dans les données sortantes. L'interconnexion des unités est réalisée grâce au protocole d'interconnexion du NetFPGA, c'est à dire grâce au bus de données, au bus de contrôle et aux signaux.



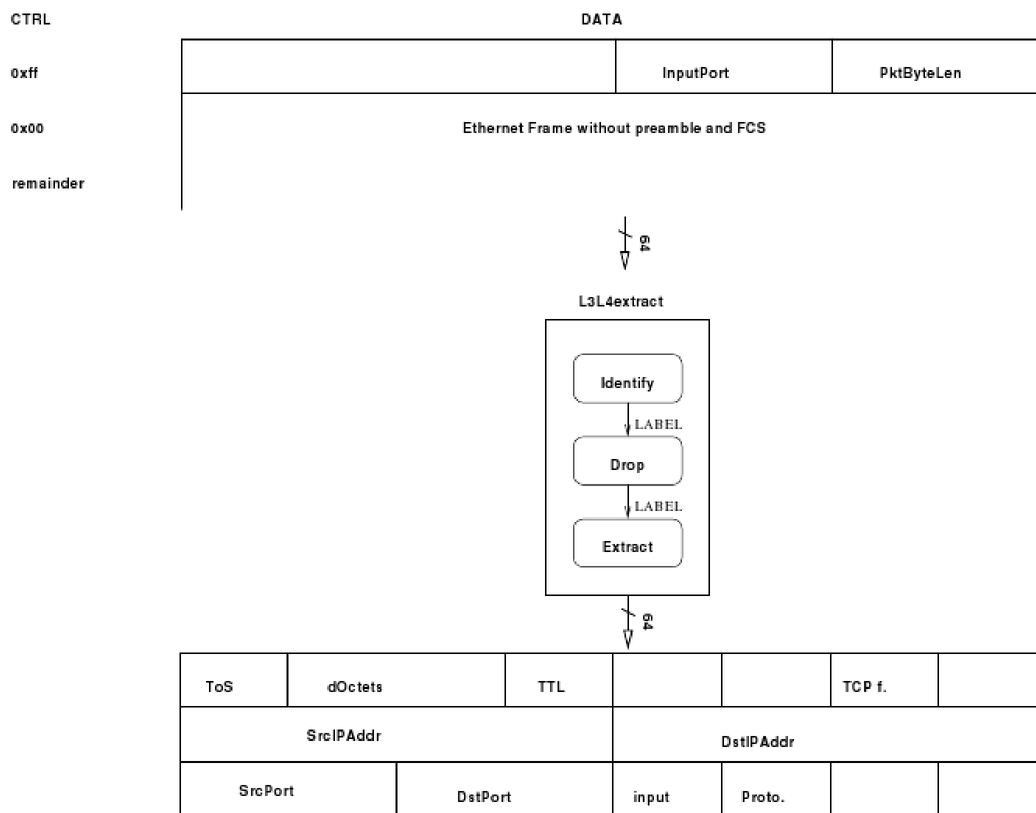
Graphique 3 : firmware pipeline

3.5. Parseur d'en-tête L3L4

Le parseur extrait les en-têtes du paquet ayant une utilité pour la surveillance. Ceci formera un Packet Record (qui sera désigné PR par la suite). Le reste du paquet est ignoré. L'entrée et la sortie de l'unité de parsage sont affichées sur la figure ci-dessous. Le parseur identifie le type de paquet encapsulé dans la frame et extrait les informations afin de créer un Packet Record. Les paquets supportés sont :

- TCP/IPv4
- UDP/IPv4
- ICMP

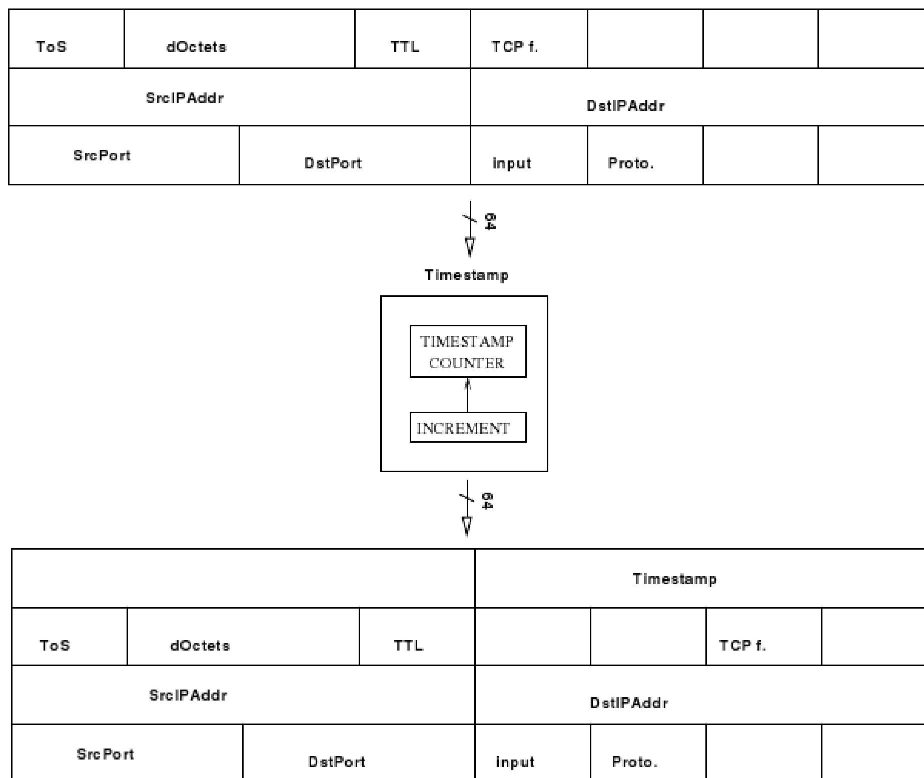
Les autres types de paquets sont ignorés et considérés comme inutiles. Cette unité de parsage est consitutée de deux processus indépendants : l'un parse les en-têtes du paquet, l'autre génère le flux de sortie.



Graphique 4 : l3,l4 parser

3.6. L'unité de Timestamp

Cette unité insère la valeur courante du timestamp (obtenu à partir d'un compteur dédié) dans le PR. Le timestamp représente le temps en millisecondes depuis lequel la sonde a commencé à fonctionner. Le compteur timestamp fait 32 bits et déborde après 49 jours et 17 heures. La vitesse du compteur est ajustée par incrémentation. La valeur d'incrémentation correspond au nombre de cycles d'horloge en une milliseconde, c'est à dire la durée d'une milliseconde. Cette valeur peut être changée de manière logicielle afin d'accélérer ou de ralentir le compteur permettant ainsi de synchroniser le temps du firmware avec le temps de l'ordinateur hôte.

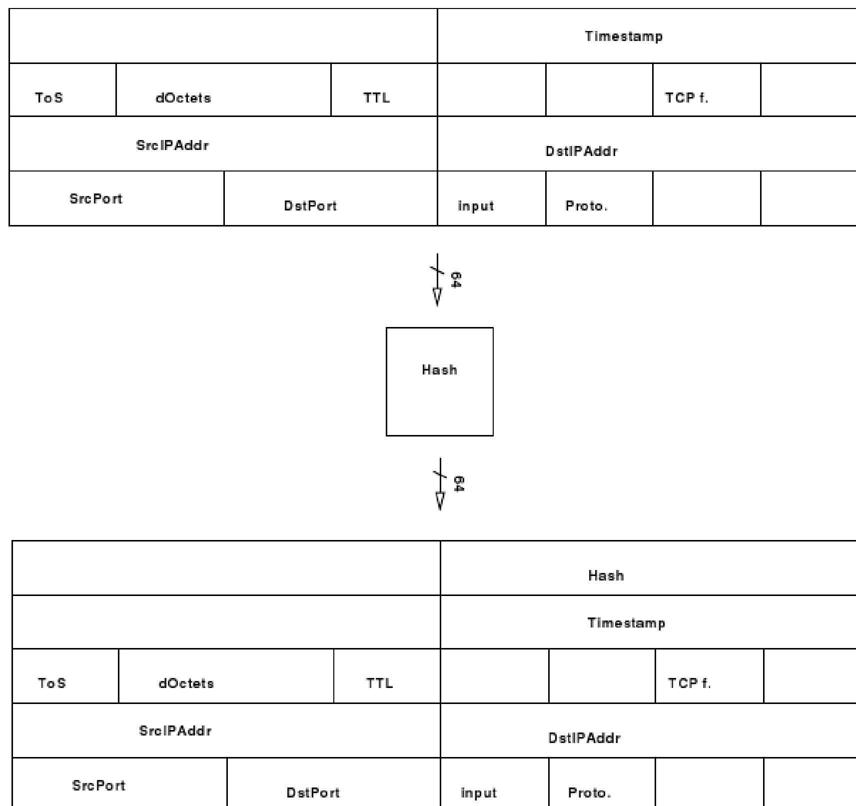


Graphique 5 : timestamp unit

3.7. *L'unité de hash*

L'unité de hash génère des valeurs de hash 64 bits (CRC-64) et l'insère dans le PR. Le hash est seulement calculé à partir des champs suivants :

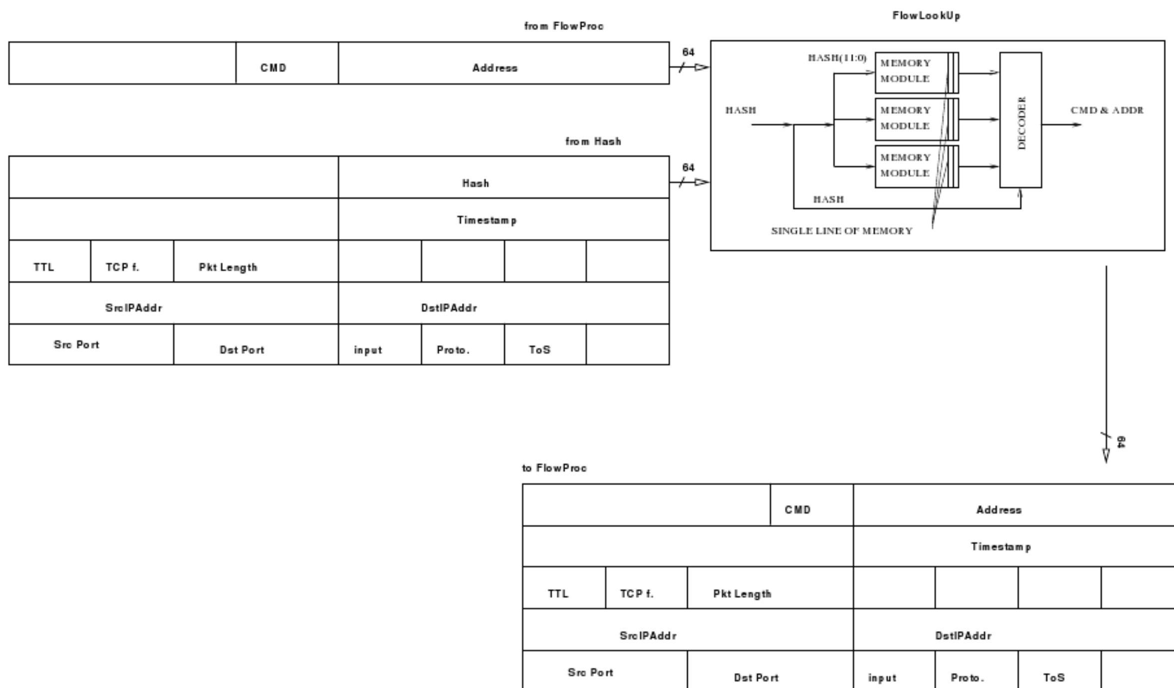
- IP source
- IP destination
- Port source
- Port destination
- Interface d'arrivée du paquet
- Protocole



Graphique 6 : hashing

3.8. *L'unité FlowLookUp*

Cette unité divise le hash en deux parties. La première partie est utilisée pour adresser une ligne contenant 8 valeurs de hash de 8 enregistrements de flux différents (8 empreintes). Ces empreintes sont comparées à la seconde partie du hash. S'il y a une correspondance avec l'une des valeurs de hash dans la ligne alors l'enregistrement de flux est déjà dans la mémoire de flux. On peut obtenir son adresse en joignant la première partie du hash et le rang de l'empreinte qui lui correspond. S'il n'y a pas de correspondance et que le nombre d'enregistrements de flux est inférieur à 8 alors l'espace disponible est utilisé pour ajouter la nouvelle empreinte. S'il n'y a ni correspondance ni espace libre alors un enregistrement de flux, arbitrairement choisi, est défini comme expiré et est remplacé par le nouveau.



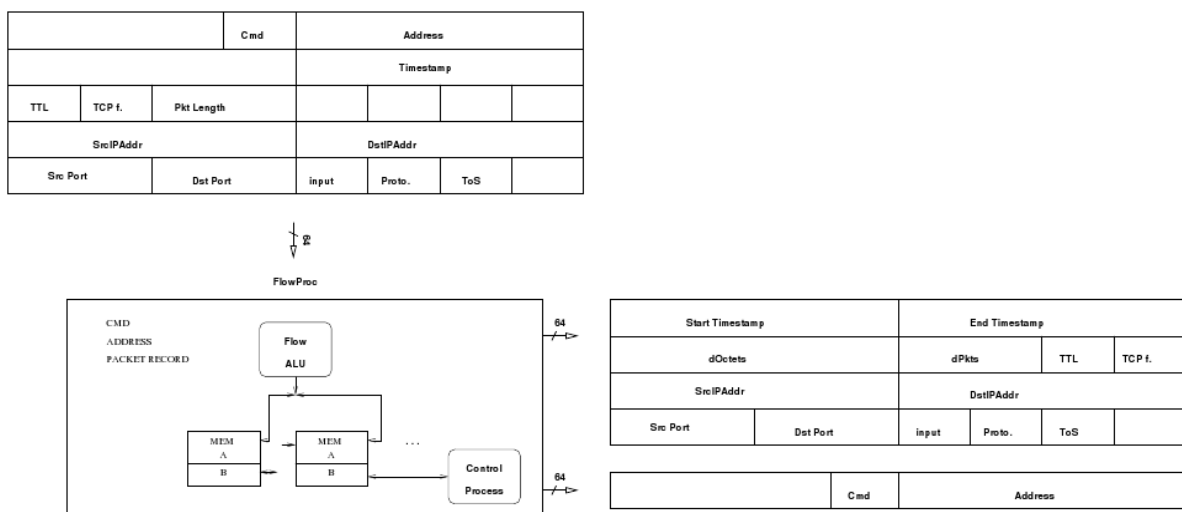
Graphique 7 : Flow look up

Quand un enregistrement de flux est sur le point d'être expiré, l'unité FlowLookUp reçoit une commande de la part de l'unité FlowProc pour qu'il supprime l'empreinte associée de la table de hash. Immédiatement après cette commande, il est envoyé à l'unité FlowProc.

3.9. *L'unité de traitement de flux*

L'unité de traitement de flux contrôle la création de nouveaux flux, la mise à jour des flux existants et l'expiration des flux inactifs. Le processus d'expiration tourne en parallèle des processus création/mise à jour. L'exclusion mutuelle n'est pas nécessaire car elle est implicitement assurée par le protocole suivant :

1. Quand le processus d'expiration identifie un flux inactif alors il envoie une commande de suppression à l'unité FlowLookUp.
2. L'unité FlowLookUp supprime l'index de l'enregistrement du flux et renvoie en retour la commande de suppression à l'unité de traitement de flux.
3. L'unité de traitement de flux rapatrie et efface l'enregistrement de flux correspondant et l'envoie à sa sortie.



Graphique 8 : flow processing

3.10. L'unité Record Wrapper

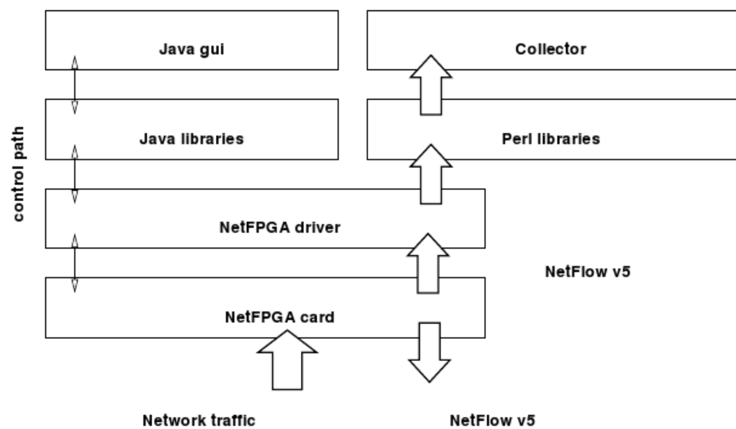
Les enregistrements de flux libérés sont temporairement stockés dans un module Record Wrapper. Dès que les conditions suivantes sont remplies les enregistrements stockés sont reformatés au format NetFlow v5 et envoyés à la file d'attente de sortie du NetFPGA :

- 15 enregistrements sont présents dans le tampon.
- Le premier enregistrement du tampon est plus vieux que 20 ms.

Le datagramme NetFlow v5 est envoyé à l'interface de sortie spécifiée dans un registre logiciel. Il pourrait y avoir plus d'une interface de sortie.

3.11. L'application logicielle

L'application logicielle de la sonde NetFlow consiste en des interfaces de configuration et de statistiques ainsi qu'en un simple collecteur (NetFlow).



Graphique 9 : software

Conclusion

L'étude de la sonde NetFPGA NetFlow nous montre le potentiel des cartes réseau FPGA pour collecter des informations sur les flux réseau. Elle nous éclaire aussi sur la façon de diviser les tâches et de programmer pour un NetFPGA. Cette implémentation va donc pouvoir nous servir de brique de base, de modèle aussi bien au niveau du découpage des tâches que des fonctions qui sont implémentées.

4. Les outils

4.1. *Xilinx*

Disposant d'un FPGA de Xilinx, le NetFPGA nous propose de travailler avec les outils de conception fournis par Xilinx (Xilinx ISE Version: 10.1 SP3).
Xilinx ISE (Integrated Software Environment) est un outil qui permet de contrôler tout aspect du flux de conception. Au travers du navigateur du projet on peut accéder aux designs et aux outils de conception ainsi qu'à tous les documents et fichiers associés au projet. Xilinx ISE nous permet ainsi de commencer par la conception de l'architecture le « HDL Design Flow », une fois bien défini il nous permet de le simuler, de l'exécuter, de réaliser des simulations adaptées temporellement et de le configurer et l'intégrer sur le FPGA.

Les étapes proposées par ISE :

→ *ISE Design Entry Flow*

- « ISE primary user interface », permet d'accéder à tout élément nécessaire et agit comme navigateur
- « HDL-based design », conception basée sur HDL, éditeur de texte, création de noyau
- « Schematic-Based Design », conception de flow à base de schémas

→ *ISE Simulation Flow (Modelsim), ISE Implementation Flow*

- « Behavioral Simulation », simulation des designs, vérification de la logique
- « Design Implementation », traduction, mapping, routage et génération du Bit file
- « Timing Simulation », simulation temporisée, analyse des pires des cas

Graphique 10 : Xilinx ISE 10, le navigateur de projets

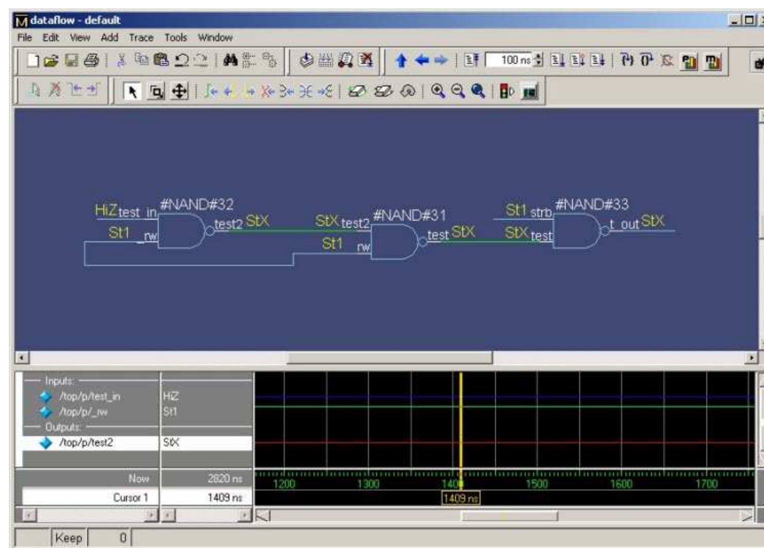
4.2. ModelSim – Advanced Simulation and Debugging

ModelSim, développé par Mentor Graphics, est un environnement de simulation pour des circuits, ou de HDL (Hardware Description Language).

Il propose un environnement de simulation avec des possibilités de débogage pour Verilog, VHDL et SystemC et est ainsi un choix possible en tant que simulateur pour des ASICs et FPGAs. Les circuits peuvent être analysés grâce à des modèles numériques ainsi que analogiques et des courbes de simulation.

ModelSim permet une simulation synchrone et exacte en termes de temps, des éléments logiques, tant que du calcul et l'affichage des valeurs analogiques.

Exemple de processus de simulation d'une description VHDL

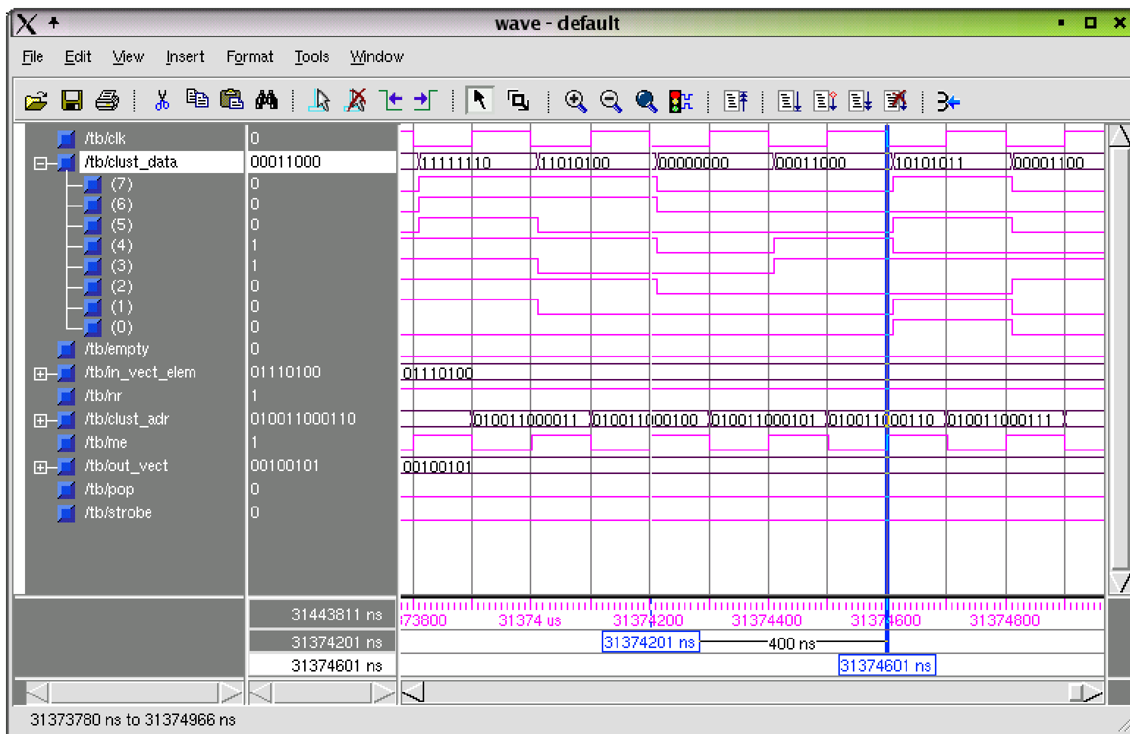


Graphique 11 : circuit logique représenté sous ModelSim

Afin de fournir un format compréhensible au simulateur, un compilateur VHDL transforme la description VHDL dans un format binaire en vérifiant en même temps que la syntaxe est correcte.

Le compilateur cherche les bibliothèques nécessaires et y place les données simulées. Ces données sont chargées par ModelSim et la simulation est lancée après avoir défini les signaux concernés. Les résultats de la simulation sont enregistrés dans un ficher wave.

La simulation peut être lancée pas à pas, avec ou sans une limitation temporelle. Pour la simulation pas à pas, avec chaque pas, une seule ligne du code source est exécutée. Après avoir arrêté la simulation avec un Break-Point, manuellement ou après le temps de simulation défini, les signaux et variables choisis vont être représentés graphiquement dans la fenêtre « wave ».



Graphique 12 : la fenêtre „wave“, représentant l’évolution des signaux et variables choisis

La version que nous allons utiliser est la version SE 6.2G, qui correspond à la version haute performance avec des capacités de débogage de blocs plus larges.

4.3. ChipScope

Chipscope est un utilitaire qui fait partie de la suite Xilinx SE [XIL09]. Il sert à ajouter à un système modélisé sur Xilinx la logique nécessaire pour analyser les signaux qui passent sur n'importe quel bus interne du système. La logique ainsi générée en surplus est optimisée pour utiliser le moins de performances et de connexions possibles.

Une fois le système ainsi généré transféré sur le FPGA, Chipscope permet d'analyser certains signaux choisis pendant un temps donnée. Il affiche alors des graphiques montrant la valeur des signaux binaires surveillés en fonction du temps. Il est ensuite possible de manipuler ces signaux binaires (par exemple en les regroupant pour former un bus [LAS04]).

Cet outil ne sera utile qu'une fois que nous aurons un modèle synthétisable. Il ne fonctionne en effet pas à partir d'une simulation mais à partir d'une exécution réelle sur le FPGA. Il permet de trouver d'éventuels bugs une fois le système implémenté, et de les comprendre pour pouvoir les corriger, avec une vue interne du fonctionnement.

Chipscope fonctionne cependant normalement avec le FPGA connecté par JTAG à l'ordinateur. Il faudra donc probablement l'adapter si nous voulons le faire communiquer avec notre NetFPGA connecté par un port PCI. Mais d'autres ont déjà utilisé Chipscope avec un NetFPGA.

4.4. SoCLib

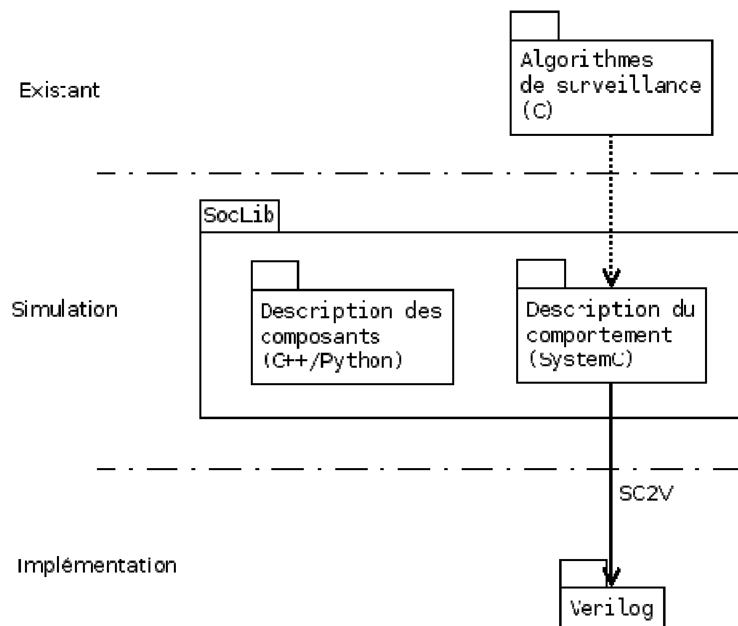
SoCLib est une plateforme ouverte permettant de réaliser le prototypage virtuel de système-sur-puce multiprocesseurs (MP-SoC). Elle s'adresse aux concepteurs d'applications embarquées en fournissant des outils d'exploration d'architecture.
Le projet SoCLib est financé par l'Agence Nationale pour la Recherche et implique six compagnies et dix laboratoires travaillant ensemble pour créer cette plateforme.

Les Objectifs

Le cœur de la plateforme est une bibliothèque de modèles de simulation SystemC pour composants virtuels (IP cores) qui est fournie sous forme libre. Le but est d'aider à la conception d'applications embarquées et de permettre une accélération de simulation en générant automatiquement des modèles de simulation. De cette façon on veut rendre possible une évaluation de performance plus tôt dans le processus de conception.

Après avoir défini l'architecture sous SoCLib, le code Verilog n'est pas mis à disposition librement. Il faut donc trouver une façon de traduire le code SystemC utilisé pour réaliser l'architecture sur FPGA.

Utilisation de SC2V
SC2V est un outil open source permettant de convertir du code SystemC en code Verilog. SoCLib étant écrit en SystemC, on peut espérer convertir le code SocLib en Verilog grâce à cet outil. L'idée est représentée dans l'image ci-dessous.



Graphique 13 : vue d'ensemble des outils

Cependant SC2V comporte d'importantes restrictions de conversion :

- Chaque module doit posséder un fichier .h avec les déclarations des ports, signaux et processus. Le code des processus doit se trouver dans un .cpp
- L'écriture dans un port ou signal doit se faire avec la méthode **.write()**

- Les **macros** ne sont pas entièrement supportées
- Seulement les types de données **bool**, **sc_int**, **sc_bigint**, **sc_uint** et **sc_biguint** sont supportées.
- Les **variables globales** ne sont pas supportées.

Nous avons pu tester avec succès SC2V sur les différents exemples SystemC fournis avec le programme. En revanche il reste toujours à le tester sur d'autres sources de codes.

Choix de méthode

Considérant les contraintes imposées par cette méthode il se pose ainsi la question de développer un convertisseur SoCLib vers Verilog ou d'écrire directement du code Verilog.
Utiliser un convertisseur implique une connaissance poussée de SocLib et SystemC ainsi que de Verilog. Le choix d'écrire directement en Verilog enlève un niveau d'abstraction mais rend possible un apprentissage progressif du Verilog et le code produit peut être utilisé immédiatement. Sachant que nous ne maitrisons que peu Verilog nous avons décidé de nous orienter vers le deuxième choix : écrire directement en Verilog. Les autres projets NetFPGA déjà réalisés et en réalisation n'utilisent pas SoCLib non plus et produisent le code Verilog directement, nous suivons donc une logique de développement qui a fait ses preuves. Par ailleurs la plateforme SoCLib est surtout intéressante pour des conceptions contenant plusieurs puces.

4.5. Verilog et VHDL

Verilog et VHDL (Very high speed integrated circuit Hardware Description Language) sont des langages de programmation utilisés pour décrire des systèmes électroniques. Ils permettent de décrire le matériel à un niveau d'abstraction plus élevé : Ils décrivent des circuits en définissant leurs modules, portes logiques et expressions. Ensuite c'est la combinaison des éléments décrits qui rend possible la synthèse du circuit et son test dans un outil de simulation. Les tests sont nécessaires pour valider le comportement des circuits ainsi que leurs performances.

5. Les algorithmes

Nous voulons implémenter sur le NetFPGA un algorithme d'analyse des flux de paquets IP sur un lien à très haut débit (100Gb/s) pour faire de la classification de services ou de la détection d'attaques.

Nous allons d'abord préciser les problématiques réseau qui nous intéressent, puis nous verrons plusieurs algorithmes et méthodes existants qui peuvent aider à implémenter des solutions sur un lien à très haut débit grâce à l'accélération matérielle du NetFPGA.

5.1. Problématiques réseau

Nous nous intéressons à deux types de problèmes dans les réseaux : la classification de services, et la détection d'attaques.

5.1.1. Classification de services

La classification de services par les routeurs d'un réseau est un problème important : cette classification permet d'affecter des qualités de service différentes aux services (par exemple une priorité pour la voix sur IP, et une priorité plus faible pour les applications de téléchargement peer-to-peer). Elle peut aussi servir à détecter certaines attaques qui se mettent à utiliser intensivement des services inhabituels.

Cette classification se base sur les en-têtes des paquets : en-tête IP (adresse source et destination) mais aussi en-tête UDP ou TCP (numéros de port source et destination). Mais les ports utilisés par une application sont facilement modifiables. Pour être plus fiable, la classification se base donc sur des analyses plus fines comme la taille des paquets envoyés, le degré d'asymétrie de la connexion (les paquets sont presque tous envoyés de A vers B, et seuls des acquittements sont envoyés de B vers A, ou au contraire A et B envoient la même quantité de données)...[TAN97] L'analyse n'utilise pas directement le contenu des paquets UDP ou TCP car il peut être chiffré, il n'est pas toujours bien défini, et cela nécessiterait de plus des ressources importantes (beaucoup de protocoles à ce niveau là sont définis en texte, et non plus de manière binaire).

Ceci pose un problème important : la reconstitution du flux. Que ce soit :

- en TCP où un flux est clairement défini par le protocole : il commence par un paquet SYN et se termine par un paquet FIN ;
- ou en UDP où un flux se crée de fait : il y a une communication entre deux terminaux : il faut récupérer tous les paquets entre les deux terminaux sur les mêmes ports, en maintenant un timer pour décider quand le flux s'arrête.

Cette reconstitution demande de grosses ressources en mémoire et en processeur. Il faut en effet retenir les premiers paquets tant que l'on n'a pas assez d'informations pour prendre une décision sur le flux, puis garder ce flux en mémoire.

5.1.2. Détection d'attaques

Actuellement énormément de machines et de services sont accessibles par Internet. Ce qui facilite grandement les échanges et le travail collaboratif facilite aussi les attaques de tous types. Pour se protéger de ces attaques, les antivirus et pare-feux sur les terminaux ne suffisent pas. C'est pourquoi nous nous intéressons à l'implémentation de la détection d'attaques à l'intérieur même du réseau, sur des liens à très haut débit, pour arrêter les attaques aussi tôt que possible.

Contrairement à la classification de services, la détection d'attaques peut dans certains cas se faire efficacement sans reconstituer les flux

5.1.3. Attaques DoS et DDoS

Les attaques DoS (Denial of Service : attaques par déni de service) sont les plus courantes : elles consistent à rendre des services inaccessibles. Elles ne servent pas à accéder à des données protégées ou à les modifier, mais simplement à nuire à la réputation d'un fournisseur de services en provoquant un arrêt de certains services pendant le plus longtemps possible.

Ces attaques peuvent être effectuées en saturant un serveur de requêtes, ou en exploitant les failles d'un protocole ou d'un logiciel (souvent les attaques utilisent les deux méthodes à la fois).

Les attaques DDoS (Distributed Denial of Service : déni de service distribué) sont des attaques par déni de service effectuées depuis plusieurs machines sur le réseau. La personne malveillante prend au préalable le contrôle de plusieurs machines dites zombies (en utilisant un cheval de Troie par exemple), puis utilise ces zombies pour augmenter sa puissance d'attaque.

Détecter ces attaques au cœur du réseau évite de devoir les détecter sur la machine cible qui risque de toute manière d'être surchargée rien que par la détection des attaques.

5.1.4. Syn Flooding

Les attaques par Syn Flooding font partie des attaques par déni de service. Elles saturent le serveur cible en utilisant une faille du protocole TCP. Elles peuvent être détectées sans reconstituer de flux.

Le Syn Flooding consiste à envoyer un grand nombre de requêtes TCP de type SYN à un serveur. Ces requêtes ont pour but d'ouvrir une connexion TCP. Les serveur répond alors par des paquets de type SYN-ACK, et le client est sensé répondre par un paquet ACK, ce qu'il ne fera pas. Ceci ouvre à moitié de nombreuses requêtes, ce qui consomme des ressources sur le serveur, qui finit par saturer.

En effet le protocole TCP n'oblige pas la machine qui initie la connexion à créer un contexte de connexion pour envoyer la première requête, alors que le serveur doit créer un contexte avant de répondre. Il consomme donc plus de mémoire et de ressource que l'attaquant, ce qui rend l'attaque efficace.

Pour détecter ces attaques, il faut compter le nombre de paquets TCP SYN envoyés à un même hôte. S'il y en a un nombre anormalement élevé, c'est une attaque par Syn Flooding.

5.1.5. Ping Flooding

Les attaques par Ping Flooding sont aussi des attaques par déni de service qui peuvent être détectées sans reconstituer de flux. Ce sont des attaques très simples qui consistent simplement à envoyer un maximum de requêtes ICMP de ping à la cible. Il est aussi possibles d'envoyer des paquets ICMP avec un contenu plus gros que la taille maximum contenue dans un paquet IP. Cela oblige la cible à reconstituer les paquets à la sortie. Il faut cependant que l'attaquant ait d'énormes capacités pour surcharger un serveur de cette manière.

La détection du Ping Flooding est à peu près similaire à celle du Syn Flooding, mais ces attaques sont moins efficaces.

5.1.6. Attaques Teardrop

Les attaques Teardrop sont un exemple d'attaques par déni de service qui ne peuvent pas être détectées par un simple comptage comme les précédentes. Elles utilisent une faille dans le réassemblage des paquets IP fragmentés en insérant des données de décalage erronées dans

l'en-tête des paquets. Une machine recevant ces paquets IP risque de redémarrer si elle est sensible à cette attaque.

La meilleure méthode pour se protéger de ce genre d'attaques est de modifier les équipements terminaux pour corriger la faille. C'est un exemple d'attaques qui ne concerne par le cœur du réseau (la détection consommerait énormément de ressources pour vérifier les informations de fragmentation de tous les paquets).

5.2. Le Stream Mining

Il s'agit d'un ensemble de techniques visant à analyser un flux de données avec les contraintes suivantes :

- en temps réel ;
- sans stocker l'intégralité du flux dans la mémoire.

Dans notre cas, nous voulons analyser un flux de paquets IP sur un lien à très haut débit (100Gb/s) pour détecter des menaces ou classifier les services utilisés. Le temps réel est donc nécessaire pour pouvoir réagir immédiatement (bloquer certains paquets, en rendre d'autres prioritaires...). Étant donné le débit des données manipulées, nous ne pouvons pas stocker l'intégralité du flux : 100Gb/s stockés pendant 5 minutes nécessiteraient un disque de 30Tb/s, et l'accès au disque serait beaucoup trop lent.

Les techniques de Stream Mining sont particulièrement utilisées pour faire du comptage. Ici, en comptant certains types de requêtes, nous pourrions par exemple détecter des tentatives d'attaques par surcharge depuis une adresse IP donnée.

Comme nous ne stockons pas toutes les données, nous ne pouvons pas faire un comptage exact. Nous pouvons cependant donner une réponse approchée qui peut être :

- déterministe : on sait que l'estimation se trouve à une distance inférieure à ε de la valeur réelle
- probabiliste : on a une très faible probabilité δ que l'estimation se trouve à une distance supérieure à ε de la valeur réelle.

La conséquence de ε dans notre cas est nulle, car nous devrons fixer un seuil à partir duquel nous détectons une menace. Il suffit donc de relever ce seuil pour éviter les fausses alertes. En revanche δ est une probabilité d'erreur. C'est donc une probabilité de détecter une attaque alors qu'il n'y en avait pas. Il faudra donc le garder le plus petit possible, et étudier les conséquences d'une fausse alerte.

Le Stream Mining permet deux types de détection [BCC07] :

- la détection des « heavy hitters » : les sources qui génèrent beaucoup plus de trafic d'un certain type que les autres : c'est la méthode utilisée pour détecter le SYN flooding ;
- la détection des « heavy changes » : les sources qui génèrent à un moment donné beaucoup plus de trafic d'un certain type que quelques instants auparavant.

5.3. L'algorithme CMS

L'algorithme Count Min Sketch a été mis au point par Cormode et Muthukrishnan en 2004. Il est simple, ce qui rend possible son implémentation lors de la prise en main du FPGA. Il peut être utilisé pour la détection de heavy hitters, nous allons donc le présenter ci-dessous en l'appliquant à la détection d'attaques par SYN flooding.

Il prend en compte un flux d'éléments avec une marque donnée, et un identifiant donné, et donne une évaluation probabiliste de la somme des marques d'éléments reçus d'un

même identifiant. Dans le cas de la détection de Syn Flooding, les éléments du flux sont les paquets TCP de type SYN, et l'identifiant pourrait être simplement l'adresse IP destination. La marque serait toujours 1.

Pour faire fonctionner l'algorithme, on choisit une précision ε et une probabilité d'échec δ.
On prend alors $w = \frac{e}{\varepsilon}$, $d = \ln(\frac{1}{\delta})$.
Les identifiants sont dans l'ensemble {1,2,...,N}. On prends alors w fonctions de hashage aléatoires, uniformes au deuxième ordre et indépendantes qui vont de {1,2,...,N} dans {1,2,...,w}.
On utilise alors un tableau de taille d x w. C'est l'espace mémoire nécessaire pour faire fonctionner l'algorithme. Pour que l'utilisation de cet algorithme ait un sens, il faut que d x w << N. C'est ce qui contraint le choix de ε et δ.

A chaque paquet reçu, on applique chaque fonction de hashage à l'identifiant, et on ajoute dans la case désignée par le couple {numéro de la fonction de hashage, valeur obtenue par hashage de l'identifiant}la marque du paquet.
L'estimation de la somme des marques pour un identifiant donné à un moment donné est la valeur minimum trouvée dans les cases désignées par les couples {numéro de la fonction de hashage, valeur obtenue par hashage de l'identifiant} pour toutes les fonctions de hashage.

	φ_1	φ_2	φ_3	φ_4	φ_5
$1 = \varphi_3(IP_0)$	0	0	**1**	0	0
$2 = \varphi_1(IP_0)$	**1**	0	0	0	0
3	0	0	0	0	0
$4 = \varphi_2(IP_0) = \varphi_5(IP_0)$	0	**1**	0	0	**1**
5	0	0	0	0	0
$6 = \varphi_4(IP_0)$	0	0	0	**1**	0

Tableau 1: Exemple de remplissage du tableau lors de la réception du premier paquet (IP_0) de marque 1 avec $\varphi_1 ... \varphi_5$ fonctions de hashage dans 1...6

Les complexités en temps de cette algorithme, pour l'ajout d'un paquet ou pour l'estimation du poids d'un identifiant, sont en $O(\ln(\frac{1}{\delta}))$. La complexité en mémoire est de $O(\frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

Il est ensuite possible d'appliquer cet algorithme à la recherche d'objets massifs. Dans notre cas, les objets massifs sont les attaques probables. On considère alors tous les identifiants dans l'estimation du poids à l'instant t est supérieure à une constante multipliée par la somme des poids de tous les identifiants au même instant (la constante est strictement inférieure à 1, elle définit le seuil à partir duquel une alerte est lancée).

La complexité de la recherche d'objets massifs est comparable au simple algorithme CMS. On fait une évaluation du poids total de l'identifiant à la réception de chaque paquet, et on maintient une liste des paquets massifs et la somme totale des marques.

Cet algorithme (recherche d'objets massifs incluse) a été implémenté en C par S. Muthu Muthukrishnan en 743 lignes [MUT05]. Nous ne connaissons pas d'implémentation en Verilog/VHDL, qui sera donc à réaliser.

5.3.1. Le problème de l'inversion du hashage

L'algorithme CMP pose un problème soulevé dans les recherches : est-il possible d'inverser le hashage opéré sur les adresses IP : nous pouvons évaluer le poids d'une adresse IP donnée, mais pouvons-nous par exemple a posteriori chercher dans la table les poids les plus forts, et retrouver leurs adresses IP ?

La réponse est clairement non si nous choisissons les fonctions de hashage au hasard. Cependant des chercheurs dirigés par Robert Schweller [SLC09] ont implémenté sur FPGA un algorithme qui permet cette inversion, en choisissant intelligemment les fonctions de hashage et les identifiants (une fonction de l'IP au lieu de l'IP directement) utilisés.

Ceci permet des analyses a posteriori au lieu des analyses incrémentales comme la recherche d'objets massifs. On peut ainsi faire des détections plus poussées, et utiliser moins de mémoire (il n'est plus nécessaire de stocker la liste des objets massifs potentiels à chaque instant par exemple).Cela nous donne une piste d'amélioration de l'algorithme, même si ce n'est pas nécessaire pour une première implémentation de la détection d'attaques par SYN flooding.

5.4. La mémoire Trie

Lors de la présentation du projet, Sylvain Gombault nous a présenté le projet VTHD++ [GOM09], qui avait pour but de faire du contrôle d'accès réseau en minimisant l'impact sur la QoS. En particulier tous les paquets devaient être traités avec le même délai, et le nombre de règles définies ne doit pas modifier ce délai.

Pour mettre en œuvre le contrôle d'accès, un ensemble de règles est appliqué consécutivement à chaque paquet en fonction du quadruplet {adresse IP source, port source, adresse IP destination, port destination}.

Pour rendre le temps de traitement indépendant du nombre de règles et du type de paquet, le principe de la mémoire trie (de l'Anglais reTRIEve, récupérer [WIK09]) a été utilisé : un arbre orienté est généré à partir des règles de sécurité, et l'orientation dans cet arbre se fait en lisant un à un chaque octet du quadruplet {adresse IP source, port source, adresse IP destination, port destination}. A chaque descente d'un cran, on arrive soit sur un nœud et on lit la suite, soit sur une feuille qui donne une action à faire (transmettre, jeter, sauvegarder le paquet). Cet arbre orienté sert ensuite à générer une structure mémoire : il s'agit d'un tableau ayant en abscisses toutes les valeurs possibles d'un octet (les octets du quadruplet), et en ordonnées des numéros de règle. Ce tableau est en mémoire dans le FPGA qui le lit à chaque réception de paquet.

Valeur de l'octet	12	**22**	**172**	181	**200**	202	**203**
Règle 1			**1) → 4**				→ 6
Règle 2		→ 5					
Règle 3	forward				→ 8		
Règle 4		**2) → 7**				drop	
Règle 5	→ 2				store		
Règle 6		forward		→ 1			
Règle 7							**3) drop**
Règle 8	→ 7			store			

Tableau 2: Exemple d'application de la mémoire trie sur l'IP 172.22.203.200

Prenons l'exemple de la mémoire trie ci-dessus et d'un paquet provenant de l'IP 172.22.203.200. On ne prend en compte dans cet exemple que l'IP source, et pas le quadruplet.

La mémoire n'est pas complètement représentée pour ne pas surcharger le tableau :

- il commence à la règle 1 avec le premier octet 172 de l'adresse IP source, ce qui lui donne le numéro de règle suivant 4 ;
- il lit la règle 4 avec le second octet de l'adresse IP source 22, ce qui lui donne le numéro de règle suivant 7 ;
- il lit la règle 7 avec le troisième octet de l'adresse IP source 203, ce qui lui donne l'ordre « drop ». Il jette alors le paquet.

Le défaut de cette technique est l'espace mémoire nécessaire, qui est important. Lors du projet VTHD++, elle a cependant été implémentée sur un FPGA et pouvait fonctionner avec un débit allant jusqu'à 6,6Gb/s. Des améliorations sont possibles en traitant en parallèle plusieurs paquets par exemple.

La mémoire trie pourra nous être utile si nous cherchons à traiter les paquets par règles, car elle est très efficace à implémenter sur un FPGA et s'adapte bien à un filtrage par IP. Elle n'est pas nécessaire pour la simple détection d'attaques par SYN flooding, mais pourrait permettre de généraliser cette détection. Nous pourrions en effet créer un système générique de règles qui permettrait de compter certains types de paquets, puis de couper au-dessus d'un certain seuil. Une règle pourrait ainsi être écrite pour détecter le SYN flooding, et d'autres menaces pourraient être détectées de la même façon.

5.5. Notre position face aux implémentations existantes

Les problèmes de classification de services et de détection d'attaques sur les liens à haut débit sont actuellement très activement explorés. Il existe donc déjà de très nombreuses implémentations à ce sujet.

BoonPing Lim et Md. Safi Uddin [LUD05] proposent par exemple une implémentation de la détection d'attaques par SYN flooding sur un processeur réseau (donc pas directement une implémentation matérielle) en utilisant l'algorithme CUSUM, qui est une alternative plus complexe à l'algorithme CMS, avec les mêmes objectifs.

D'un autre coté, Robert Schwelle [SLC09] propose une implémentation sur FPGA qui utilise un algorithme de hashage réversible, permettant une gestion globale et a posteriori du réseau.

Nos objectifs sont d'abord de monter en compétences sur l'utilisation du NetFPGA, et d'utiliser l'accélération matérielle du NetFPGA dans un problème simple d'analyse te trafic en temps réel. C'est pourquoi nous allons nous intéresser à l'implémentation de l'algorithme CMS (dont nous avons déjà une implémentation en C) pour la détection d'attaques par SYN flooding sur un lien à très haut débit. Nous pourrons ensuite tenter de généraliser cette approche pour gérer d'autres attaques (par exemple le ping flooding, et toutes les attaques par flooding) en utilisant un système de règles implémenté grâce à une mémoire Trie.

6. Conclusion

Le NetFPGA permet d'apporter une accélération matérielle à des algorithmes de gestion du réseau dans des projets de recherche.

Nous nous baserons sur l'implémentation du routeur de référence pour développer notre prototype, et nous y ajouterons nos propres fonctions en les intégrant à cette architecture de référence. Ceci nous permettra de bénéficier des fonctions de base (gestion du protocole IP) tout en profitant de l'accélération matérielle.

Notre algorithme sera écrit directement en Verilog, ce qui évitera les problèmes de traduction depuis le SystemC. Nous utiliserons pour cela les outils classiques proposés pour le NetFPGA : la suite Xilinx ISE, ainsi que les utilitaires de commande spécifiques au NetFPGA.

Nous chercherons d'abord à réaliser un algorithme de détection d'attaques par SYN flooding, attaques courantes et relativement simples à détecter. Nous utiliserons pour cela l'algorithme CMS. L'architecture de la sonde NetFPGA NetFlow nous servira de modèle concernant la façon de diviser les tâches et de programmer sur NetFPGA.

Par la suite l'algorithme de mémoire Trie pourrait nous permettre d’adapter notre outil à la détection d'autres formes d'attaques.

7. Bibliographie

[BCC07] Tian Bu, Jin Cao, Aiyou Chen, Patrick P. C. Lee. *A Fast and Compact Method for Unveiling Significant Patterns in High Speed Networks*. IEEE. 2007.

[BFG09] Yannick BLEUWART, Éric FAGOT, Grégory GIEMZA, Yanick PIGNOT. *Séminaire de détection d'intrusions Dos, DDos. Attaque du Syn Flooding : exemple sur un serveur web Apache*. **[En ligne]** Disponible sur http://www.student.montefiore.ulg.ac.be/~bleuwart/ (consulté le 30 novembre 2009).

[CHE05] Pascal CHEUNG. *Une introduction aux techniques de stream mining*. France Telecom. **[En ligne]** Disponible sur http://trac.benoute.fr/netfpga/raw-attachment/wiki/CMS/t_12janvier05_c.pdf (consulté le 30 novembre 2009), 12 janvier 2005.

[FRE60] Edward Fredkin. *Trie Memory*. **[En ligne]** Disponible sur http://delivery.acm.org/10.1145/370000/367400/p490-fredkin.pdf?key1=367400&key2=4663059521&coll=GUIDE&dl=GUIDE&CFID=64097606&CFTOKEN=74272576 (consulté le 30 novembre 2009), 1960.

[GOM09] Sylvain Gombault. Octobre 2009. *Présentation du projet VTHD++*.

[LAS04] Anselmo Lastra. *Chipscope Tutorial*. **[En ligne]** Disponible sur http://www.cs.unc.edu/~lastra/comp190/Notes/13_Chipscope.pdf (consulté le 1 décembre 2009), Février 2004

[LUD05] BoonPing Lim, Md. Safi Uddin. *Statistical-based SYN-flooding Detection Using Programmable Network Processor*. IEEE. **[En ligne]** Disponible sur http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1489006&isnumber=32022 (consulté le 1er décembre 2009), juillet 2005.

[MSa] ModelSim. **[En ligne]** Disponible sur http://www.model.com/content/modelsim-se-high-performance-simulation-and-debug (consulté le 1 décembre 2009).

[MSb] ModelSim. **[En ligne]** Disponible sur http://www.mikrocontroller.net/articles/ModelSim (consulté le 1 décembre 2009).

[MUT05] S. Muthu Muthukrishnan. *MassDal public code bank*. **[En ligne]** Disponible sur http://www.cs.rutgers.edu/~muthu/massdal-code-index.html (consulté le 30 novembre 2009), 2005.

[NET08] John W. Lockwood, G. Adam Covington, Jan Kořenek et al. NetFPGA : Tutorial in Brno, Brno, Czech Republic, 5 Septembre 2008. Disponible sur http://netfpga.org/tutorials/Brno2008/ppt/NetFPGA_Brno_2008_09_04c.pdf

[NET09a] NetFPGA: Guide **[En ligne]**. Disponible sur http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/Guide (consulté le 1er décembre 2009).

[NET09b] NetFPGA: NetFPGA Video Demonstrations **[En ligne]**. Disponible sur http://www.netfpga.org/php/videos.php (consulté le 1er décembre 2009).

[SLC09] Robert Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A. Dinda, Ming-Yang Kao, Gokhan Memik. *Reversible Sketches: Enabling Monitoring and Analysis Over High-Speed Data Streams*. IEEE. 2009.

[SoCL08] SoCLib. **[En ligne]** Disponible sur https://www.soclib.fr/trac/dev/wiki (consulté le 30 novembre 2009).

[TAN97] K. M. C. Tan, B. S. Collie. *Detection and Classification of TCP/IP Network Services*.

[En ligne] IEEE. Disponible sur http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=646179&isnumber=14094 (consulté le 30 novembre 2009), 1997.

[VERa] Verilog. **[En ligne]** http://en.wikipedia.org/wiki/Verilog (consulté le 1 décembre 2009).

[VERb] Verilog. **[En ligne]** http://fr.wikipedia.org/wiki/Verilog (consulté le 1 décembre 2009).

[VHDL] VHDL. **[En ligne]** Disponible sur http://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language (consulté le 1 décembre 2009).

[WIK09] Wikipédia. *Trie.* **[En ligne]** Disponible sur http://en.wikipedia.org/wiki/Trie (consulté le 30 novembre 2009), 29 novembre 2009.

[XIL09] Xilinx, *ChipScope Pro and the Serial I/O Toolkit.* **[En ligne]** Disponible sur http://www.xilinx.com/tools/cspro.htm (consulté le 1er décembre 2009). 2009

[XIL-ISE] Xilinx ISE 11. **[En ligne]** Disponible sur http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise11tut.pdf (consulté le 30 novembre 2009).