PR310
Projet Avancé
Rapport

Cryptographie sur cible software et hardware sur FPGA

Élèves :
Yohenn Perrot
Clément Couret
Ludovic Debonne
Yves-Guédi Farah

Enseignants :
Jérémie Crenne
Bertrand Legal

Année universitaire 2020-2021

Bordeaux INP
ENSEIRB
MATMECA

Table des matières

1 Introduction

La confidentialité, l'authenticité ou encore l'intégrité des données sont aujourd'hui des enjeux majeurs du monde numérique et sont regroupés sous le terme de cryptographie.

La cryptographie est une discipline ayant pour objectif de protéger les données. Depuis l'antiquité, elle est utilisée pour rendre n'importe quel message lisible uniquement par celui-ci détenant la clé pour le décrypter. De nos jours, de nombreux algorithmes cryptographiques veillent à protéger toute les informations sensibles dans le domaine bancaire, militaire ou encore privé...

Notre objectif lors de ce projet est de comparer les performances de plusieurs algorithmes de cryptographie en fonction de la cible, logicielles et/ou matérielles, sur laquelle ils sont implémentés. Pour cela nous avons à notre disposition des PCs équipés d'Intel Xeon, nos ordinateurs personnels équipés d'Intel Core i3, i5 et i7 et des cartes FPGA Nexys4 et Nexys A7.

Dans un premier temps, nous détaillerons notre démarche de recherche de code C existant d'algorithme de cryptographie, les tests logiciels sur PC de ces algorithmes et notre première approche du sujet. Puis nous décrirons, l'étape de modélisation haut niveau afin de passer nos algorithmes sur des cibles matérielles. Nous expliquerons ensuite l'implémentation sur Nexys A7 de l'algorithme RC4 et nous comparerons les performances logicielles et matérielles obtenues à des articles de recherche. Pour finir, nous analyserons notre travail et les possibilités futures.

2 Recherche bibliographique et développement logiciel

Dans un premier temps, nous avons décidé de chacun travailler sur un algorithme différent. Ainsi, selon les spécificité de chacun, nous pouvons comparer l'efficacité des accélérations que nous pouvons effectuer grâce à un passage de logiciel à matériel.

2.1 RC4

Le RC4 (Rivest Cipher 4) est un algorithme de chiffrement par flot. Cela signifie qu'il peut crypter des données de toute taille. Il a été utilisé dans des protocoles comme WEP, WPA et TLS.

Le fonctionnement est le suivant : la clef RC4 permet d'initialiser un tableau de 256 octets en répétant la clef autant de fois que nécessaire pour remplir le tableau. Par la suite, des opérations très simples sont effectuées : les octets sont déplacés dans le tableau, des additions sont effectuées, etc. Le but est de mélanger autant que possible le tableau. Finalement on obtient une suite de bits pseudo-aléatoires qui peuvent être utilisés pour chiffrer les données via un XOR.

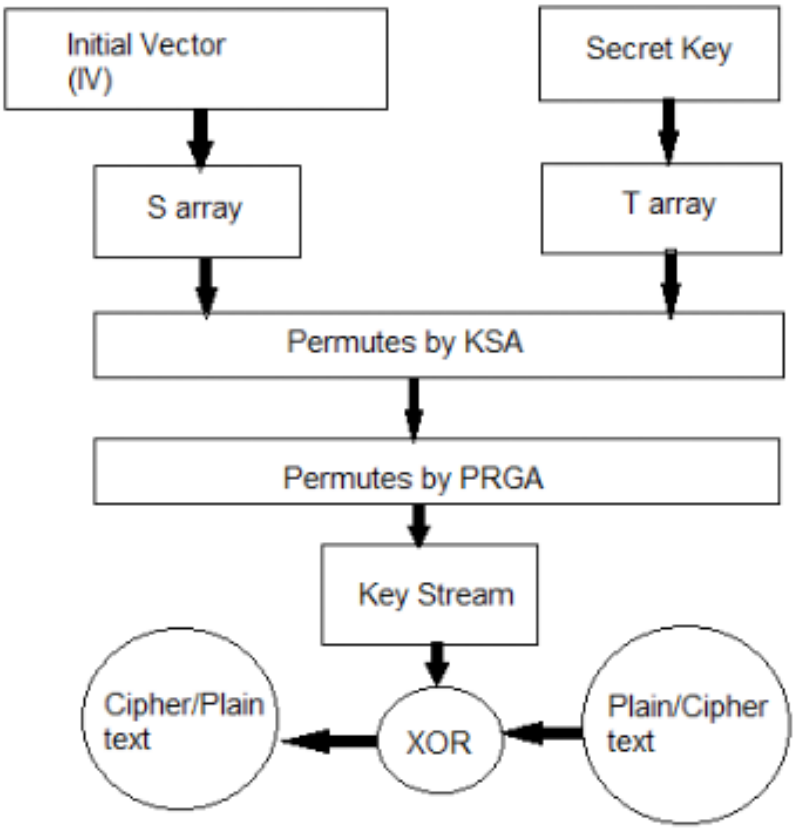

FIGURE 1 – RC4

2.2 AES

AES (Advanced Encryption Standard) est le successeur du DES (Data Encryption Standard) qui était utilisé par les organisations gouvernementales américaines jusque dans les années 90. C'est un algorithme symétrique qui a remporté un concours en 2000 pour devenir la nouvelle méthode d'encryptage pour les Etats-Unis.

L'AES est un algorithme très robuste car il n'a pas été cassé (même de manière théorique) à ce jour. Son fonctionnement repose sur le principe du chiffrement par bloc. Chaque donnée n'est pas traitée indépendemment mais un bloc de données est créé et crypté. Plutôt que de faire un copier-coller-reformuler je vous conseille d'aller voir le fonctionnement détaillé sur Wikipédia car il est très bien expliqué. La figure ci-dessous résume également très bien le fonctionnement général de l'algorithme.

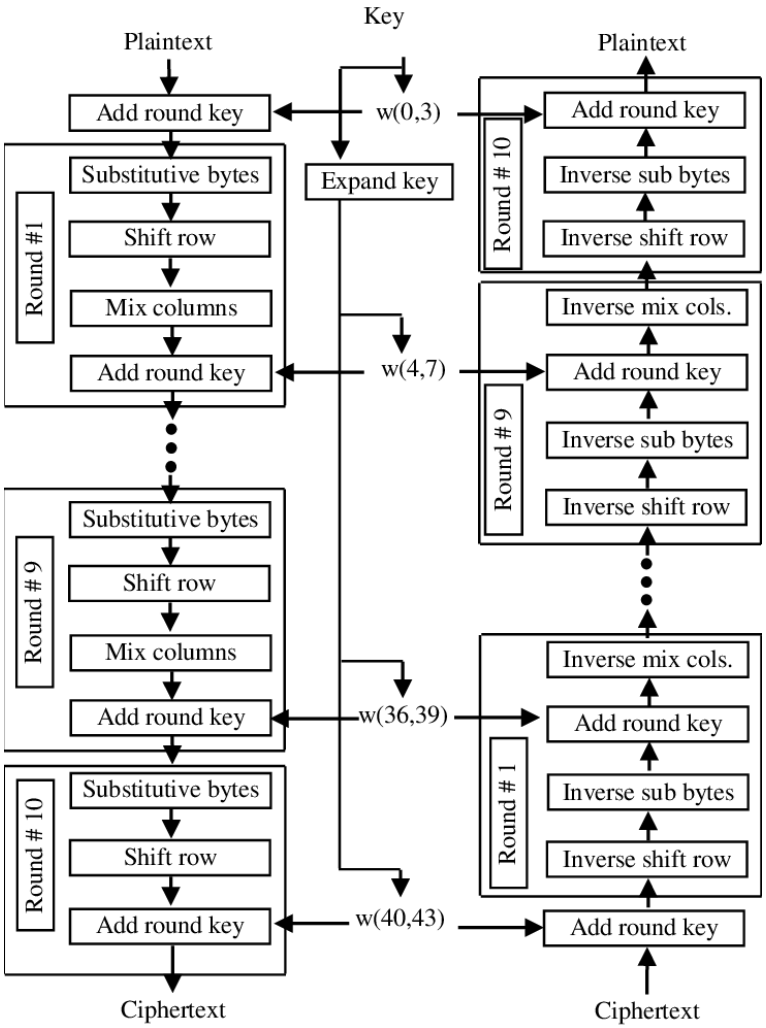

FIGURE 2 – AES

La NSA (Agence de Sécurité National américaine) considère aujourd'hui que AES est suffisament fiable pour être utiliser sur des informations classées secrète. En revanche, elle recommande pour les informations top secrète de ne pas utiliser une clé 128 bits mais 192 ou 256 bits qui sont les trois tailles de clés possibles pour AES.

2.3 PRESENT

Le Present Cipher est un algorithme principalement utilisé pour l'IoT : l'internet des objets (Internet of Things). Par exemple, dans une maison connectée on peut retrouver des ampoules connectées à une base. Il y a donc des communications entre cette base et ces ampoules. Cet algorithme sert à crypter ces communications. Il est donc très efficace pour les communications entre éléments à faible distance et disposant de ressources limitées. De plus, il possède un bon équilibre entre vitesse et sécurité. De nos jours, cet algorithme est utilisé sur de nombreux objets connectés. Même s'il n'est pas très sécurisé, cela prendrait trop de temps de casser le cryptage au vu du gain réalisable car les communications contiennent majoritairement des informations presque inutiles pour l'espion qui souhaite les décrypter.

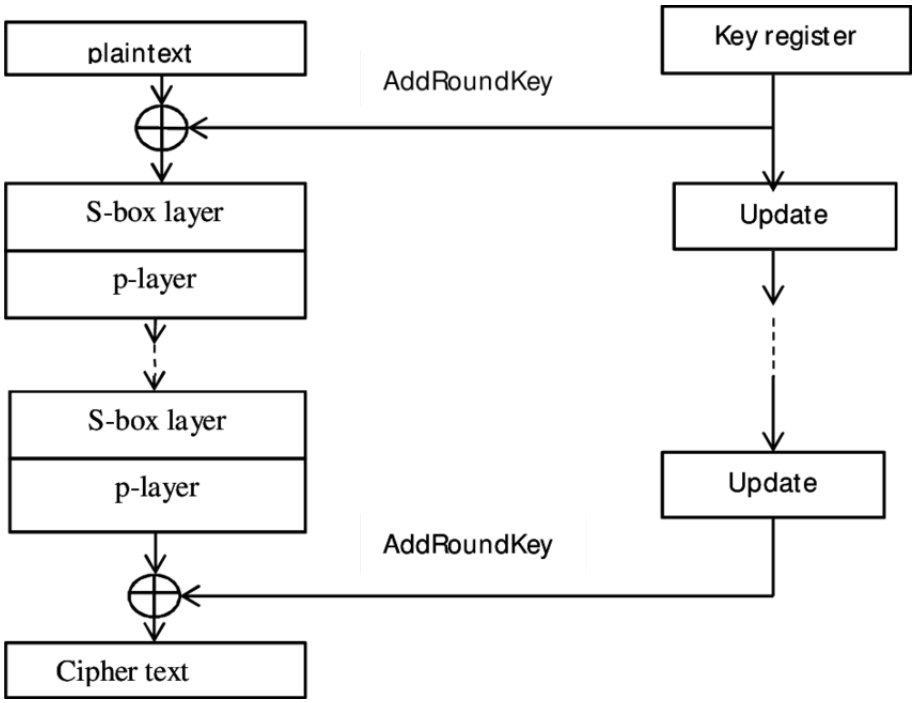

FIGURE 3 – Present Cipher

Comme le montre la figure ci-dessus, cet algorithme fonctionne de la façon suivante. L'émetteur et le récepteur possèdent une même clé de 80 bits ainsi qu'une table de permutation qui est publique. L'algorithme tournera un nombre de tours précis. A la fin de chaque tour, il y a une nouvelle clé et un nouveau message. Ainsi, si on augmente le nombre de tour on gagne en sécurité. Cependant, on perd naturellement en vitesse. Chaque tour est composé de trois actions. Pour commencer, les 64 premiers bits de la clé sont récupérés, puis un XOR est effectué entre la clé et le message. Ensuite, des permutations sont effectuées sur la clé et sur le message suivant la table de permutation pré-définie. Cela nous donne bien un nouveau message et une nouvelle clé. Puis on recommence le tour suivant. Une fois tous les tours effectués, récupère le message final en effectuant une dernière fois un XOR entre le message et les 64 premiers bits de la clé. La plupart du temps, 31 tours sont effectués. C'est suffisant pour obtenir une bonne sécurité et ce n'est pas trop long.

2.4 SERPENT

Il s'agit d'un algorithme de chiffrement par bloc conçu par Ross J. Anderson, Eli Biham et Lars Knudsen. Cet algorithme prends en entrée des blocs de taille 128 bits et supporte des clés de 128, 192 ou 256 bits. Le bloc considéré est subdivisé en 4 mots de 32 bits, qui subissent 32 tours de substitutions et de permutations. L'algorithme est doté de 8 matrices de 16*16 éléments appelées S-Boxes. Ce sont ces S-Boxes qui sont utilisée lors des permutations et substitutions. Les 4 mots de 32 bits subissent par la suite une série d'opérations linéaires afin de générer le bloc pour le tour suivant. Aujourd'hui, Serpent est considéré comme l'un des algorithme de chiffrement les plus sûrs existant, mais n'est pas le plus rapide lors de son exécution.

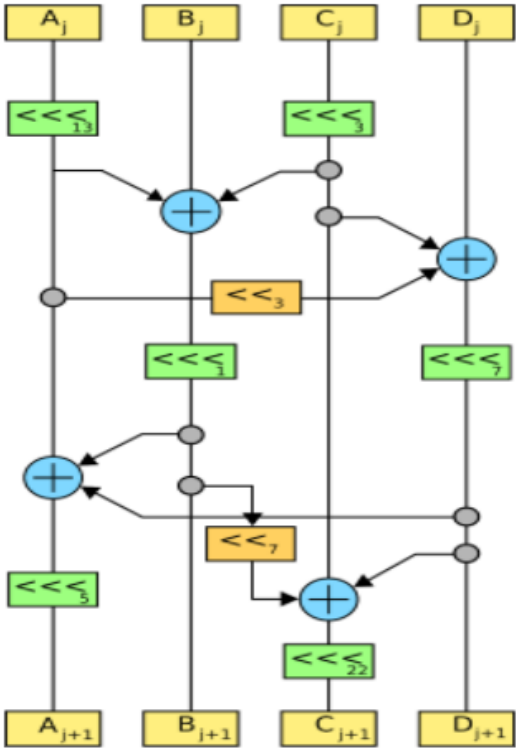

FIGURE 4 – Serpent

3 Modélisation haut niveau pour décrire le matériel

Une fois la digestion de l'algorithme ainsi que la vérification de son bon fonctionnement logiciel, notre objectif est d'effectuer une implémentation sur cible matérielle, ici une carte FPGA. Nous avons mis les codes C dans vivado HLS puis nous avons voulu générer une architecture matérielle. Les codes n'étant pas adapté nous avons dus les modifiés. Le code RC4 étant un algorithme assez simple, la modification fût rapide. Les autres algorithmes demandaient quand à eux un plus gros travail du à leur complexité. Nous sommes resté bloqué sur ce problème durant les premières séances puis nous avons décidé de travailler à plusieurs sur le RC4 qui était la meilleure piste que nous avions.

Nous avons alors généré une première architecture avec Vivado HLS pour l'algorithme RC4 qui était compliquée à comprendre et à utiliser. Nous avions pour fonction principale **void rc4 (char key[256], char plaintext[256], char ciphertext[256])** . Cette fonction demandait 3 tableaux déjà existants qui signifie en matérielle des blocks RAM. Les blocks RAM afin d'être correctement utilisés demandent une synchronisation et beaucoup de signaux à connecter.

Cette solution étant assez compliquée et une perte de temps. De plus, si un changement était fait sur le code du RC4, l'architecture générée aurait été différente à chaque fois. Nous avons alors décidé de "forcer" l'architecture générée par Vivado HLS en utilisant la librairie SystemC.

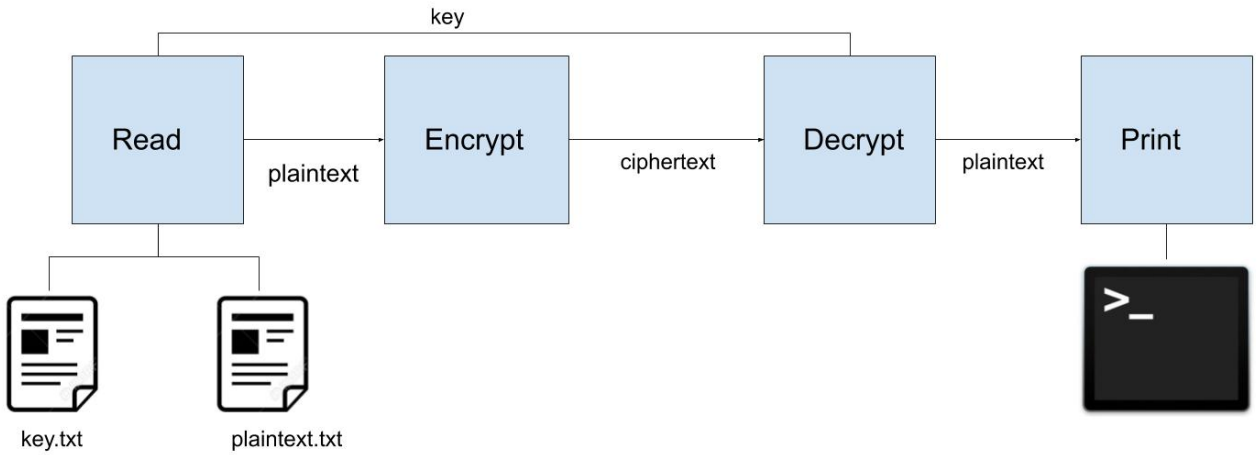

FIGURE 5 – Systemc

— Le bloc **read** récupère dans 2 fichiers la clé et le plaintext. Il transmet la clé puis le plaintext dans une FIFO.
— Le bloc **encrypt** récupere toutes les données puis génére le ciphertext.
— Le bloc **Decrypt** récupere toutes le ciphertext puis génére le plaintext.
— Le bloc **Print** affiche de nouveau plaintext dans le terminal, il nous permet de valider ou nous le fonctionnement du module.

Le module que nous allons implémenter sur la carte sera Encrypt. Il crée tout d'abord 3 tableaux qui stockeront clé, plaintext et ciphertext.

Listing 1 – Initialisation

```
char key[N];
char plaintext[N];
unsigned char ciphertext[N];
```

Il récupère 256 données provenant de la FIFO, ces données représente la Clé.

Listing 2 – Récupération de la clé

```
for( i=0;i<N;i++){
                key[i] = e.read();
        }
```

Il récupère ensuite 256 données provenant de la FIFO, ces données représente le plaintext.

Listing 3 – Récupération du plaintext

```
for(int j=0;j<N;j++){
                plaintext[j] = e.read();
```

```
}
```

A partir de la clé et du plaintext, il génére le ciphertext

Listing 4 – Encryption

```
KSA(key, S);
PRGA(S, plaintext, ciphertext);
```

Il écrit le ciphertext dans la FIFO de sortie.

Listing 5 – Écriture du ciphertext

```
for(k=0;k<N;k++){
s.write( sc_uint<8> (ciphertext[k]) );
}
```

4 Implémentation sur FPGA d'un algorithme RC4

Une fois le code SystemC fonctionnel, il est possible à l'aide de l'outil Vivado HLS de synthétiser du VHDL à partir du SystemC qui décrit le comportement attendu pour notre système. Ce bloc VHDL est généré sous forme d'IP (Intellectual Property). Un IP est un bloc possèdant des entrées et sorties connues et réalisant une tâche ou un comportement spécifique. Les IPs permettent de réduire de manière importante le temps de développement en réutilisant des blocs déjà conçus, testés et fonctionnels en les intégrant directement à son projet.

Dans notre cas, l'utilisation d'un IP permet d'interchanger rapidement l'algorithme que l'on souhaite étudié dans la chaine de test que nous avons développé. Il suffit de fixer pour toutes les IPs les mêmes entrées et sorties afin qu'il puisse se comporter comme des blocs "Plug and Play".

A partir du code vu ci-dessus, nous pouvons maintenant générer le bloc IP avec Vivado HLS. Le module sera intégré de la manière suivante :

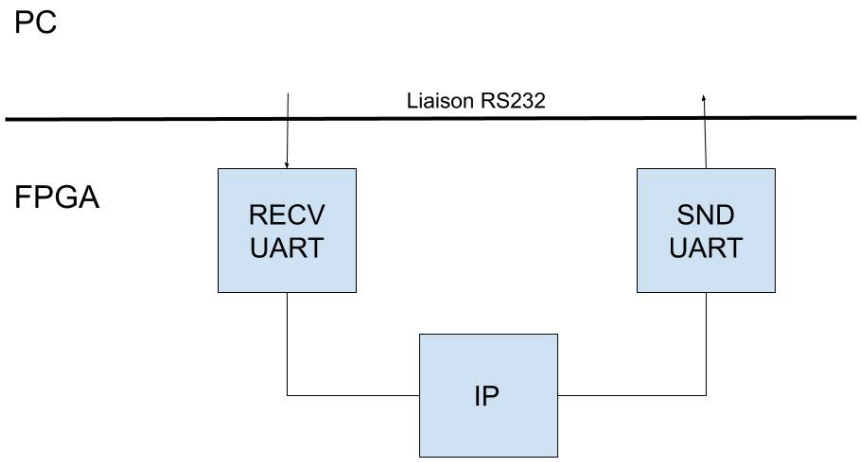

FIGURE 6 – Integration IP

On voit dans sur la figure ci-dessus que les blocs RECV UART et SND UART sont développés une seule et unique fois et que seul l'IP est remplacé pour tester un algorithme différent.

5 Optimisation

Nous avons maintenant un produit fonctionnel, nous allons chercher à optimiser celui-ci. Pour cela, nous avons effectué des recherches sur internet. Nous avons trouvé un groupe d'ingénieurs américains qui ont effectués la même implémentation mais sur une Virtex 2. Lorsque l'on compare les résultats, ils obtiennent de meilleures performances en utilisant moins de ressources que nous comme le montre le tableau suivant. Il y a donc des optimisations possibles à effectuer.

Cible	FPGA Artix 7	FPGA Virtex 2
Performances	18 MB/s	59 MB/s
LUTs	1278	298
FFs	528	205
BRAMs	6	2

Pour commencer, nous avons ajouté des directives sur Vivado HLS. Après plusieurs essai et de nombreuses comparaison, nous avons choisi d'utiliser l'architecture pipeline. Cela nous a permis de passer de 18 à 29 MB/s. Ensuite, nous avons modifié notre code afin de l'optimiser pour la carte cible. Par exemple, nous avons remplacé l'opération modulo 256 par un "et" logique avec un masque. Avec ces optimisations, nous obtenons 33 MB/s, ce qui est loin de ce qu'on obtenu les américains mais est presque le double de ce qu'on avait au départ. Par manque de temps, nous n'avons pas pu plus approfondir cette phase d'optimisation.

6 Comparaison du RC4 sur cible matériel et logiciel

Maintenant que nous avons optimisé l'implémentation du RC4, nous pouvons comparer les performances sur le CPU de l'ordinateur et la cible matérielle FPGA. Les résultats sont consignés en Figure 7.

Target	CPU - Intel Core I3 - 2.1Ghz	FPGA - Artix 7	USA - FPGA Virtex 2
Performance	7 MB/s	33 MB/s	59 MB/s
LUTs	---	1120	298
FFs	---	217	205
BRAMs	---	6	2

FIGURE 7 – Performances mesurées

Nous remarquons que le débit d'information est environ 5 fois supérieur sur le FPGA que sur le CPU. Cela peut s'expliquer par le fait que le CPU est généraliste, c'est à dire qu'il doit s'occuper de l'exécution de notre programme mais aussi de toutes les autres fonctions de l'ordinateur, alors que le FPGA est configuré pour ne réaliser qu'une tâche précise. Donc, si l'on souhaite chiffrer et déchiffrer un grand nombre d'information dans le minimum de temps, l'utilisation d'une cible matérielle FPGA dédiée à cette tâche est recommandé.

7 Possibilités futures

Tout d'abord, nous n'avons finalement réussi à réaliser le travail initialement prévu qu'avec le RC4. Dans un premier temps, nous pouvons donc travailler sur les trois autres algorithmes afin de les implémenter sur FPGA et ainsi pouvoir effectuer des comparaisons entre chaque algorithmes. Pour ce faire, nous pouvons également revoir notre méthode de travail. Durant ce projet, une fois les codes C++ récupéré, nous avons uniquement essayé de les retravailler nous-même. Une autre méthode est de récupérer des codes C++ sur internet qui sont déjà retravaillé pour fonctionner avec Vivado HLS. Ainsi, comme nous avons déjà créé la structure permettant de les accueillir et de calculer les temps de calcul, nous n'avons plus qu'à les comparer.

Enfin, l'autre possibilité est, au lieu de multiplier les algorithmes, de multiplier les cibles sur lesquelles on implémente l'algorithme. Ainsi, on travaille toujours avec le même algorithme mais on change de carte cible et de langage utilisé pour coder. Ainsi, on obtient aussi de nombreuses comparaison mais pour un seul algorithme.

8 Conclusion

Pour conclure, ce projet nous a permis d'implémenter un code C sur FPGA. Lors de ce projet nous avons été confronter à des problèmes lors de cette transition. Il faut adapter le code et réfléchir à son imlémentation matérielle derrière. Le problème le plus rencontré a été l'utilisation de pointeurs qui n'est pas implémentable directement par Vivado HLS. Il a fallu trouvé des alternatives et voir le programme d'une manière différente. Ce projet nous a de plus permis de d'aborder la cryptographie.