Université du Québec
à Rimouski

IMPLANTATION SUR CIRCUIT SOC-FPGA D'UN SYSTÈME DE CHIFFREMENT/DÉCHIFFREMENT AES-128 BITS EN UTILISANT DEUX APPROCHES DE DIFFÉRENTS NIVEAUX D'ABSTRACTION

MÉMOIRE PRÉSENTÉ

dans le cadre du programme de maîtrise en ingénierie

en vue de l'obtention du grade de maître ès sciences appliquées (M.Sc.A.)

PAR

© AYOUB CHANANE

Décembre 2022

Composition du jury :

Chan-Wang Park (Ph.D.), président du jury, Université du Québec à Rimouski

Mohammed Bahoura (Ph.D.), directeur de recherche, Université du Québec à Rimouski

Fethi Meghnefi (Ph.D.), examinateur externe, Université du Québec à Chicoutimi

Dépôt initial le 14 octobre 2022 Dépôt final le 15 décembre 2022

À ma mère et mon père

À mon frère

À mes sœurs

À mes proches

REMERCIEMENTS

Je voudrais dans un premier temps remercier, mon directeur de recherche Monsieur Mohammed Bahoura, professeur au Département de Mathématiques, d'Informatique et de Génie (DMIG) de l'Université du Québec à Rimouski (UQAR), pour l'accompagnement et l'encadrement qu'il m'a offert tout au long de ma maitrise. Je le remercie pour ses précieux conseils et ses critiques constructives qui ont permis d'améliorer la qualité de mon travail.

Je remercie également le professeur Chan-Wang Park de l'Université du Québec à Rimouski pour avoir accepté d'évaluer mon mémoire.

Je remercie aussi le professeur Fethi Meghnefi de l'Université du Québec à Chicoutimi pour avoir accepté d'évaluer mon mémoire.

Finalement, je remercie mes parents, mes frères et sœurs, ainsi que mes amis, qui m'ont soutenu et encouragé pendant toute la durée de mes études.

RÉSUMÉ

La sécurité des données est une priorité absolue dans le monde technologique. Pour garantir la sécurité et la confidentialité des données, l'usage des systèmes de chiffrement/déchiffrement devient une nécessité dans plusieurs domaines. Dans ce mémoire nous présentons une architecture simple de système de chiffrement avancé à 128 bits en mode compteur (AES-CTR-128 bit), implantée sur une carte PYNQ-Z2 pour chiffrer/déchiffrer des signaux d'électrocardiogramme ECG (ElectroCardioGram) de la base de données MIT-BIH.

Le système n'utilise que 13% des ressources matérielles du circuit Xilinx ZYNQ XC7Z020. Il consomme une puissance de 43 mW et opère à une fréquence maximale de 109.43 MHz, qui correspond à un débit maximal de 14 Gbps. Le temps d'exécution de chiffrement et de déchiffrement d'un fichier de valeurs séparée par des virgules CSV (Comma Separated Value) par rapport d'un fichier texte TXT (Text) est environ deux fois plus court dans les deux plateformes utilisant deux approches ayant des niveaux d'abstraction différents. La première utilise la programmation bas-niveaux via la plateforme Xilinx Vitis alors que la seconde utilise l'outil Jupyter/Python. L'architecture matérielle proposée est environ quatre fois plus rapide que l'implantation logicielle et il y a une légère différence au niveau du temps d'exécution pour l'implantation de notre architecture sur les deux plateformes présentées (Vivado/Vitis ou Jupyter/Python).

Nous avons aussi testé notre architecture matérielle avec d'autres types de données tels que les signaux audio et des images. Nous avons utilisé la plateforme Jupyter/Python pour sa simplicité de manipulation. Le chiffrement/déchiffrement d'un signal audio d'une durée de 7 secondes et d'une fréquence d'échantillonnage de 8 kHz est réduit respectivement à 4.6 ms et 4.87 ms, par rapport à 16.18 ms et 15.8 ms pour le chiffrement/déchiffrement d'un signal audio par l'implantation logicielle. De même pour l'image couleur et l'image en niveau de gris. Ainsi que le temps de chiffrement d'une image couleur prend entre trois à quatre fois le

temps de chiffrement d’une image en niveau de gris dans les deux implantations logicielle et matérielle. L'architecture matérielle présentée peut être utilisée dans un large éventail d'applications embarquées. Les résultats présentés ont montré que l'architecture proposée a surpassé toutes les autres implantations existantes sur FPGA.

Mots clés : Cryptographie, AES, Circuit FPGA, Signal ECG, Circuit ZYNQ, Chiffrement/Déchiffrement, Cryptage/Décryptage.

ABSTRACT

Data security is a top priority in the technological world. To ensure data security and privacy, the use of encryption/decryption systems becomes a necessity in several areas. In this dissertation, we present a simple architecture of advanced 128-bit counter mode encryption systems (AES-CTR-128 bit), implemented on a PYNQ-Z2 board to encrypt/decrypt electrocardiogram (ECG) signals from the MIT-BIH database.

The system uses only 13% of the hardware resources of the Xilinx ZYNQ XC7Z020 chip. It consumes 43 mW of power and operates at a maximum frequency of 109.43 MHz, which corresponds to a maximum through of 14 Gbps. The execution time of encryption and decryption of the comma-separated value (CSV) file compared to the text file (TXT) is about twice as short in both platforms using two approaches with different abstraction levels. The first use low-level programming via the Xilinx Vitis platform while the second uses the Jupyter/Python tool. The proposed hardware architecture is about four times faster than the software implementation and there is a slight difference in execution time for the implementation of our architecture on the two platforms presented (Vivado/Vitis or Jupyter/Python).

We also tested our hardware architecture with other types of data such as audio signals and images. We used the Jupyter/Python platform for its simplicity of handling. The encryption/decryption of an audio signal with a duration of 7 seconds and a sampling rate of 8 kHz are reduced to 4.6 ms and 4.87 ms, respectively, compared to 16.18 ms and 15.8 ms for the encryption/decryption of an audio signal by the software implementation. The same applies to the color and grayscale image. Thus, the encryption time of a color image takes between three and four times the encryption time of a grayscale image in both software and hardware implementations. The presented hardware architecture can be used in a wide range of embedded applications. The presented results showed that the proposed architecture

outperformed all other existing FPGA-based implementations.

Keywords: Cryptography, AES, FPGA circuit, ECG signal, ZYNQ circuit, Encryption/Decryption

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS, DES SIGLES ET DES ACRONYMES

AES	Advanced Encryption Standard Norme de chiffrement avancé
BRAM	Block Random Access Memory Bloc Mémoire à accès aléatoire
BSN	Body Sensor Network Réseau de capteurs corporels
BUFG	Buffer Global Mémoire tampon globale
CBC	Cipher Block Chaining Enchainement de blocs de chiffrement
CFB	Cipher FeedBack Chiffrement à rétroaction
CRC	Cyclic Redundancy Check Contrôle de redondance cyclique
CSV	Comma Separated Value Valeur séparée par une virgule
CTR	Counter Compteur
DDR	Double Data Rate Double débit de données
DES	Data Encryption System Système de chiffrement des données
DSP	Digital Signal Processing Traitement numérique du signal

ECB	Electronic CodeBook Dictionnaire de codes
ECG	ElectroCardioGram Électrocardiogramme
FF	Flip Flops Bascule à deux états
FIPS	Federal Information Processing Standards Normes fédérales de traitement de l'information
FPGA	Field Programmable Gate Arrays Réseau de portes logique programmables
HDL	Hardware Description Language Langage de description du matériel
HLS	High-level synthesis Synthèse de haut niveau
IDE	Integrated Design Environment Environnement de conception intégré
IDEA	International Data Encryption Algorithm Algorithme international de chiffrement des données
IP	Intellectual Property Propriété intellectuelle
IV	Initialization vector Vecteur d'initialisation
LUT	Look Up Table Tableau de correspondance
MIT-BIH	Massachusetts Institute of Technology-Bet Israël Hospital Institut de technologie du Massachusetts - Hôpital Bet Israël
NIST	National Institute of Standard and Technology Institut national des normes et de la technologie

NSRDB Normal Sinus Rythme DataBase
Base de données des rythmes sinusaux normaux

OFB Output FeedBack
Chiffrement à rétroaction de sortie

PIL Python Imaging Library
Bibliothèque d'imagerie Python

PL Programmable Logic
Logique programmable

PLA Programmable Logic Array
Réseau logique programmable

PN Pseudo-random Number
Nombre pseudo-aléatoire

PS Processing system
Système de traitement

PYNQ Python productivity on Zynq
Productivité de Python sur Zynq

Rcon Round constant
Constante ronde

RGB Red, Green, and Blue
Rouge, vert et bleu

ROM Read Only Memory
Mémoire à lecture seule

RSA Rivest Shamir Adleman

RTL Register Transfer Level
Niveau de transfert de registre

SoC System on Chip
Système sur puce

TDES Triple Data Encryption System
Système de triple chiffrement de données

THS Total hold Slack
Total des écarts du retard minimum

TNS Total Negative Slack
Total des écarts du retard maximum

TXT Text file
Fichier texte

TPWS Total Pulse Width Slack
Total des écarts de la largeur d'impulsion

URAM Ultra Random Access Memory
Mémoire à accès ultra aléatoire

VDMA Video Direct Memory Access
Accès direct à la mémoire vidéo

WBSN Wireless Body Sensor Network
Réseau de capteurs corporels sans fil

WHS Worst Hold Slack
Le pire des écarts de tous les chemins temporels

WNS Worst Negative Slack
Le pire des écarts de tous les chemins de synchronisation

WPWS Worst Pulse Width Slack
Le Pire des écarts de toutes les vérifications de temps

LISTE DES SYMBOLES

F	Frequency
GHz	GigaHertz
kHz	kiloHertz
MHz	MegaHertz
ms	Millisecond
ns	Nanosecond
T	Time

CHAPITRE 1
INTRODUCTION

Dans notre monde actuel, nous assistons à des avancées technologiques remarquables au niveau des réseaux informatiques et des télécommunications qui impliquent une hausse de plus en plus importante d'échange d'informations à travers le monde entier. Cette hausse d'échange a été accélérée grâce au déploiement et la popularité d'internet. Cependant, des questions sur la sécurité de nos informations et leurs utilisations se posent constamment.

La santé connectée est un domaine technologique qui permet à l'utilisateur de suivre et de gérer son propre bien-être et sa santé à l'aide d'un appareil connecté à internet. Elle consiste à accéder, collecter, gérer et analyser les données de santé à distance. Elle désigne l'ensemble des technologies permettant de connecter les dispositifs médicaux et les objets du quotidien au système de santé. C'est un domaine en pleine expansion et de nombreuses nouvelles applications sont développées chaque jour. Cette technologie va permettre aux médecins de suivre des patients en temps réel de leur domicile afin d'éviter l'engorgement des hôpitaux et d'améliorer les soins de santé à l'échelle mondiale. Il faut pouvoir protéger les données des patients pour éviter qu'elles soient altérées de façons intentionnelles qui pourraient conduire à des diagnostics erronés.

Un système offrant des services de santé connectée doit protéger et envoyer les informations générées à partir des données des patients de manière à assurer leur confidentialité et éviter qu'elles soient détournées. Pour assurer la confidentialité de données, il est important de mettre en place un système de sécurité performant qui empêche les tiers non autorisés d'accéder aux données. De plus, les données doivent être chiffrées lorsqu'elles sont transmises à travers un réseau public afin d'empêcher qu'elles soient interceptées et lues par des tiers.

Un système sur puce SoC (System on Chip), entièrement programmable, peut constituer une solution de choix pour une transmission de données sécurisée avec une faible consommation d'énergie et peu de ressources matérielles. Pour la sécurité des données, le système de chiffrement avancé AES (*Advanced Encryption Standard*) est une solution largement adoptée, car elle est considérée comme étant l'une des plus sûres. L'AES est un système de chiffrement avancé construit par Daemen et Rijmen(2000) et qui a été publié comme norme fédérale de traitement de l'information par l'Institut national des normes et de la technologie NIST (*National Institute of standard and Technology*) en 2001 (Dworkin *et al.*, 2001). Il permet un chiffrement plus rapide et sécurisé que les autres systèmes comme les standards de chiffrement de données DES (Data Encryption System) et TDES (Triple Data Encryption System) (Aleisa, 2015), ce qui le rend parfaitement adapté à de multiples applications.

1.1 PROBLÉMATIQUE

Selon les études de l'Organisation mondiale de la santé (OMS, 2020) les maladies cardiovasculaires font partie des causes de décès les plus fréquentes au monde, L'électrocardiogramme (ECG) est un signal qui représente l'activité électrique du cœur, qui permet donc de diagnostiquer la majorité des anomalies associées à celui-ci. De ce fait, le signal ECG est utilisé par les professionnels de la santé pour surveiller le bon fonctionnement du cœur. Les données ECG sont très critiques, car elles contiennent les minuscules détails sur une maladie cardiaque des patients. Par conséquent, l'utilisation de ce signal par plusieurs établissements de santé fait en sorte qu'il faut protéger les données à l'envoi de celles-ci. En plus il est illégal de transmettre des données et des rapports de santé non sécurisée, conformément au code 94.1 de déontologie des médecins (MTESS, 2022).

Par la suite, la pandémie COVID-19 que nous vivons actuellement nous mène à accélérer ce processus de sécurisation de la transmission des données numériques. Cela permet à plusieurs patients de recevoir des soins à domicile afin de désengorger les

établissements de santé actuels. De plus, le vieillissement de la population qui s'accélérera au cours des années à venir nous pousse également à redoubler l'effort pour mettre en place ce système sécurisé des données.

Le problème principal avec les systèmes de santé existants est qu'ils sont conçus pour des soins médicaux en présentiel nécessitant le déplacement des patients aux différentes installations. Cela limite l'accès et allonge les listes d'attente pour recevoir un service. Ce type de service (consultation) en présentiel engendrons beaucoup de frais pour le système de santé et également pour le patient.

La santé connectée est une approche technologique qui permet de suivre les patients en temps réel à leurs domiciles afin de faciliter leurs accès aux services de santé et améliorer la qualité des soins reçus. Cependant, elle comporte des risques pour la vie privée et la sécurité des patients. La confidentialité des données est une préoccupation majeure, car les données peuvent être interceptées et lues par des tiers si elles ne sont pas convenablement protégées.

Notre problématique de recherche consiste donc à implanter un système de chiffrement/déchiffrement des signaux électrocardiogramme ECG (ElectroCardioGram) qui assure la confidentialité, l'intégrité et la disponibilité des données, tout en réduisant les ressources matérielles requises et la puissance consommée.

De façon générale, les chercheurs du génie biomédical accordent une grande attention aux circuits FPGA (Field Programmable Gate Arrays) à cause de leur puissance de calcul et leur flexibilité. Les récentes orientations de recherche dans les systèmes ECG-FPGA se concentrent sur des aspects importants des signaux ECG, telle que la sécurité de données. Le système AES a été l'objet de nombreuses recherches depuis sa construction. En raison de la complexité de ses calculs, différents moyens ont été élaborés pour l'amélioration de cet algorithme.

Pour l'implantation matérielle de l'algorithme AES, les circuits de réseaux de portes logiques programmables FPGA constituent un choix intéressant. Ces circuits sont largement utilisés dans l'industrie ainsi que dans le milieu académique à travers le monde pour implanter

des applications complexes, tels que la cryptographie des données. Ce type d'applications inclut les architectures matérielles pour l'accélération des algorithmes de chiffrement, les dispositifs de stockage de clés et les applications de sécurité réseau.

Le système de chiffrement avancé AES (Advanced Encryption System) a été utilisé dans plusieurs domaines de protection des données et des communications numériques. Le chiffrement AES-128 bits basé sur la carte FPGA Xilinx ZCU102, adaptée aux communications 5G (Visconti *et al*., 2021). Développés en utilisant une approche pipeline, il permet ainsi le traitement simultané des tours sur plusieurs paquets de données à chaque cycle d'horloge. Dans les réseaux de capteurs corporels sans fil WBSN (Wireless Body Sensor Network), le système de chiffrement avancé AES est utilisé avec des clés basées sur l'électrocardiographie (ECG) pour sécuriser les procédures d'échange de clés (Yang *et al*., 2016). Dans les médias numériques, le tatouage invisible et l'algorithme de chiffrement AES sont utilisés pour cacher des images binaires dans des supports d'images en couleur afin de fournir une double couche de protection (Mohanty, 2009).

Les applications récentes de systèmes de chiffrement avancés, proposées dans la littérature, sont multiples et la plupart d'entre elles sont basées sur des circuits FPGA (Field Programmable Gate Arrays). Un système sans fil reconfigurable pour la surveillance de l'électrocardiogramme (ECG) en temps réel est réalisé sur un circuit FPGA Xilinx Spartan3 (XC3S1500L) qui utilise l'algorithme AES pour la sécurité des données avec une meilleure performance (Amira *et al*., 2013). Une approche de protection de l'intégrité a été proposée pour les réseaux de capteurs corporels BSN (Body Sensor Network) afin de recueillir les signaux physiologiques en vue d'une analyse basée sur la biométrie (Miao *et al*., 2009). Elle concerne la conception de schémas de chiffrement à faible consommation basés sur des circuits FPGA. L'analyse dans le domaine de fréquence des signaux électrocardiogramme (ECG) a été utilisée pour générer les clés cryptographiques. En outre, un système sécurisé pour la transmission et l'identification des données ECG, basé sur le système de chiffrement avancé (AES) implanté sur une puce ZYNQ de la carte de prototypage Xilinx ZC702, a été proposé par Zhai *et al.* (2017). Les résultats d'implantation matérielle ont montré que le

système proposé répondait aux exigences temps réel. Le système proposé nécessite 30% des LUTs disponibles et consomme une puissance de 107 mW ce qui nécessite une énergie de 1.14 mJ pour traiter un signal ECG d'une durée de 1 min.

D'autres approches qui ont combiné la compression et la cryptographie ont été proposées pour avoir une protection de sécurité significative, qui se traduit par un gain considérable en temps de traitement (Hameed *et al*., 2020; Miaou *et al*., 2002).

L'optimisation d'une architecture matérielle revient à minimiser l'utilisation des ressources tout en atteignant une vitesse et un débit élevés avec une faible consommation d'énergie. Pour ce faire, plusieurs techniques d'optimisation ont été proposées. La première technique implique l'implantation parallèle du système de chiffrement avancé AES (Liu & Baas, 2013; Mondal et al., 2020; Priya et al., 2017; Rahimunnisa et al., 2014), la seconde technique concerne l'optimisation pipeline/sub-pipeline pour augmenter la fréquence d'opération et par conséquent le débit (Fan & Hwang, 2007; Good & Benaissa, 2007; Li & Li, 2005; Rahimunnisa *et al*., 2013; Soltani & Sharifian, 2015), la troisième technique utilise le déroulage de boucle (loop-unrolling) pour avoir un débit plus élevé et une réduction de la latence (Daoud *et al*., 2019).

La table de substitution (S-Box) effectue une transformation non linéaire sur les données en remplaçant chaque octet par un octet différent. Une nouvelle approche est proposée pour améliorer la qualité du système de chiffrement avancé (AES). Cela consiste à générer les valeurs de la table de substitution (S-Box) et la clé initiale requise pour le chiffrement et le déchiffrement, en utilisant un générateur de séquence Nombres Pseudo-aléatoires PN (Pseudo-random Numbers). Ce qui a pour but d'améliorer la qualité de chiffrement qui donne un effet d'avalanche de 60% par rapport au système de chiffrement avancé (AES) classique (Zodpe & Sapkal, 2020). Une architecture a été proposée pour accélérer le processus de génération des sous-clés à partir de la clé principale; elle consiste à diviser l'architecture entière en deux blocs et à les exécuter en parallèle. Les deux blocs peuvent générer des sous-clés en parallèle à partir de la clé principale (Manoj & Karthigaikumar, 2018). L'architecture proposée par Priya *et al.* (2017) basée sur une

technique d'accès parallèle à huit étapes, est utilisée dans la substitution d'octet (SubByte/S-Box) et un calcul parallèle à huit étapes est appliqué dans la transformation MixColumns. Lorsqu'elle est implantée dans le circuit FPGA Virtex 5 XC5VLX50T, elle permet d'économiser 50% du temps requis par l'architecture proposée en 2001 (Dworkin *et al*., 2001).

Les modifications structurelles sont basées sur les transformations de la table de substitution (S-Box) et la transformation MixColumns. Certaines méthodes de modification structurelle procurent un débit élevé (Priya *et al*., 2017), tandis que d'autres procurent une faible consommation d'énergie et la minimisation des ressources (Priya *et al.*, 2021). Une implantation pipeline à quatre niveaux de la table de substitution (S-Box) a été proposée par Arundhati *et al*. (2015) qui utilise une logique combinatoire basée sur l'arithmétique de champ composite de la table de substitution (S-Box) du système de chiffrement avancé AES pour une optimisation de la surface utilisée par le dispositif.

Des études comparatives ont été menées entre l'algorithme de chiffrement avancé AES et les autres normes de chiffrements tels que les standards de chiffrement de données DES (Data Encryption System) et TDES (Triple Data Encryption System) (Aleisa, 2015; Rakanovic & Struharik, 2017). Ainsi une comparaison entre les cinq modes d'opérations AES les plus courants (Blazhevski *et al*., 2013; Hameed *et al*., 2019). De plus, une comparaison de la complexité des architectures de cybersécurité pour évaluer les performances des algorithmes et leurs niveaux de sécurité (Alharam & El-Madany, 2017).

1.2 OBJECTIF

Dans ce contexte, nous nous intéressons à la manière dont nous allons nous y prendre pour permettre le traitement temps réel et sécurisé des données de santé à l'aide de l'algorithme de chiffrement avancé AES (Advanced Encryption System) implanté matériellement sur une puce électronique. Autrement dit, notre problématique porte sur la

conception d'une architecture matérielle efficace caractérisée par un faible coût (ressources) et une faible consommation d'énergie ainsi qu'un temps d'exécution réduit. Nos objectifs consistent donc à implanter efficacement l'algorithme de chiffrement/déchiffrement AES sur une puce électronique avec deux approches de différents niveaux d'abstraction, afin de chiffrer/déchiffrer des signaux électrocardiogramme (ECG), des sons audio et des images.

1.3 HYPOTHÈSES

Les circuits ZYNQ-7000 combinent les ressources matérielles d'un circuit FPGA avec un processeur ARM Cortex-A9 à doubles cœurs. Cela permet une grande flexibilité dans l'implantation des systèmes complexes.

Les cartes PYNQ peuvent offrir une meilleure solution pour une implantation matérielle sur un circuit ZYNQ en utilisant le langage de programmation Python. La programmation logicielle pour utiliser une librairie matérielle (Overlay) de la même façon qu'une librairie logicielle.

1.4 MÉTHODOLOGIE

Notre démarche consiste à utiliser deux approches à différents niveaux d'abstraction soient la plateforme Vivado/Vitis-C++ et la plateforme Vivado/Jupyter-Python. Jupyter Notebook est une application web utilisée comme interface avec le logiciel de programmation Python. Elle a été utilisée pour traiter les données électrocardiogramme ECG, audio et image en raison de son libre accès, sa simplicité et sa convivialité. Cependant, la plateforme Vitis IDE (Langage C), nécessitant une licence, est sélectionnée afin de servir de référence pour les résultats de traitement de données avec les résultats obtenus par le biais de la plateforme Jupyter (Python).

Pour atteindre les objectifs de notre projet de recherche, notre démarche a été divisée en trois phases. Dans la première phase, nous avons effectué une recherche bibliographique sur le système de chiffrement avancé AES 128 bits en mode compteur (CTR). Ensuite, dans la deuxième phase, nous avons proposé une architecture efficace qui permet d'avoir une faible consommation d'énergie et une faible utilisation des ressources, pour implanter efficacement le système de chiffrement avancé AES-CTR-128. Finalement, dans la troisième phase, nous avons implanté l'algorithme AES sur la carte PYNQ-Z2 pour chiffrer/déchiffrer les signaux ECG, sons audio et des images.

1.5 CONTRIBUTION

Notre contribution consiste dans la mise en place d'un système de traitement rapide et sécurisé des données électrocardiogramme ECG à l'aide du système de chiffrement avancé AES-CTR-128. Pour cela, nous avons choisi de travailler avec deux approches de différents niveaux d'abstraction afin de comparer leurs performances, complexités et coûts.

Dans un premier temps, nous avons utilisé la plateforme Vivado/Vitis pour écrire le code C qui implante l'algorithme AES-CTR-128. Ce code est ensuite compilé et déployé sur la carte SoC PYNQ-Z2. Ensuite, nous avons utilisé la plateforme Jupyter Notebook pour écrire le code Python afin de faire le calcul sur l'Overlay implantant l'algorithme AES. Nous avons ensuite comparé les deux approches d'implantation en termes de complexité, de coût et de performances.

Finalement, nous avons implanté l'algorithme AES sur la carte FPGA PYNQ-Z2 uniquement en logiciel (Python) en utilisant la plateforme Jupyter Notebook pour chiffrer/déchiffrer des signaux audio et des images.

1.6 ENSEMBLE DE DONNÉES MIT-BIH

Les données électrocardiogramme (ECG) utilisé proviennent de la base de données *MIT-BIH Arrhythmia Database*, développée par le Massachusetts Institute of Technology (MIT) et le laboratoire du Beth Israël Hospital (BIH) de Boston. Cette base de données a été créée pour comparer les algorithmes de diagnostic d'arythmie cardiaque. Elle est disponible gratuitement et publiquement (Moody & Mark, 2001). L'ensemble de la base de données MIT-BIH est fourni sur la plateforme Physionet qui contient des données électrophysiologiques et des outils pour les lire. Cette plateforme est à accès libre, permettant ainsi de télécharger ou visualiser les données sous différents formats (Goldberger *et al*., 2000). Les signaux ECG sont sauvegardés dans des fichiers de format *.dat, *.hea ou *.txt, avec leurs fichiers d'annotations respectivement de format *.atr, *.hea, ou *.txt (PhysioNet, 2017). La plupart des enregistrements ECG contiennent deux ou plusieurs signaux ECG enregistrés simultanément, à l'aide d'électrodes placées à certains endroits spécifiques appelés "dérivations", pour avoir une vue complète de l'activité électrique cardiaque qui permette d'observer le complexe QRS (PhysioNet, 2017).

1.7 ORGANISATION DU MÉMOIRE

Ce mémoire est organisé comme suit : Ce premier chapitre présente la problématique de recherche, les objectifs de notre projet, les hypothèses ainsi que la méthodologie adoptée. Le second chapitre présente un bref historique de différents types de cryptographie et ces méthodes, ainsi que le principe de fonctionnement de système de chiffrement avancé (AES) et ses modes d'opération. Le troisième chapitre présente des généralités sur les circuits FPGA et les différentes étapes de l'implantation de l'algorithme de chiffrement AES. Le quatrième chapitre décrit l'expérimentation et les résultats obtenus de l'architecture proposée ainsi une comparaison avec d'autres travaux. Enfin, une conclusion qui permet de faire le bilan du travail accompli et des perspectives pour les travaux futurs.

CHAPITRE 2
CRYPTOGRAPHIE

La cryptographie est une science mathématique qui utilise des algorithmes pour protéger les informations contre tout accès non autorisé. Dans ce chapitre nous allons présenter l'historique de la cryptographie jusqu'à nos jours en passant par les différents types de chiffrements. Puis, nous verrons plus en détail le système de chiffrement avancé AES (*Advanced Encryption Standard*) qui est largement utilisé de nos jours.

2.1 INTRODUCTION À LA CRYPTOGRAPHIE

Nous allons d'abord commencer par définir le terme de la cryptographie, un mot composé de deux mots du grec ancien *kruptos* qui signifie "cacher" et *graphein* qui signifie "écrire", ce qui veut dire littéralement cacher l'écriture (Bergeron & Goupil, 2014).

La cryptographie est le processus de chiffrement des données qui permet de les rendre illisibles à quiconque ne possédant pas la clé de déchiffrement. Elle est utilisée à deux fins : la confidentialité et l'intégrité. La confidentialité fait référence à la protection des données contre une divulgation non autorisée, tandis que l'intégrité fait référence à la protection des données contre une modification non autorisée. Cette discipline concerne le développement et l'utilisation des méthodes qui permettent la transmission des messages chiffrés. En d'autres termes, cela signifie que seule la personne à qui est destiné le message peut décrypter l'information et ainsi voir les données réelles transmises.

2.2 HISTORIQUE DE LA CRYPTOGRAPHIE

Depuis plusieurs siècles, l'être humain est à la recherche de moyens pour améliorer la confidentialité et la sécurité de certaines informations spécialement celles transmises (messages) entre deux lieux. Comme illustré à la Figure 2.1, au 5^{e} siècle avant J-C, l'utilisation de la scytale, soit un bâton de bois que l'on enroulait d'une bande de cuire permettant de révéler le message. Cependant, si le bâton de bois n'était pas du bon diamètre, cela empêchait la lecture correcte de message. Ainsi, lire la bande sans l'enrouler sur le bâton de bois ne permettait pas de lire l'information (Dumas *et al.*, 2007; Thawte, 2013). Par la suite, au 1er siècle avant Jésus-Christ, une nouvelle méthode de cryptage a fait son apparition, soit celle qui était utilisée par Jules César. Ce chiffrement est basé sur la substitution des caractères du message par d'autres dans l'alphabet qui se trouve à une distance fixe (Bartz & Markey, 2012; Brisson & Théberge, 2000).

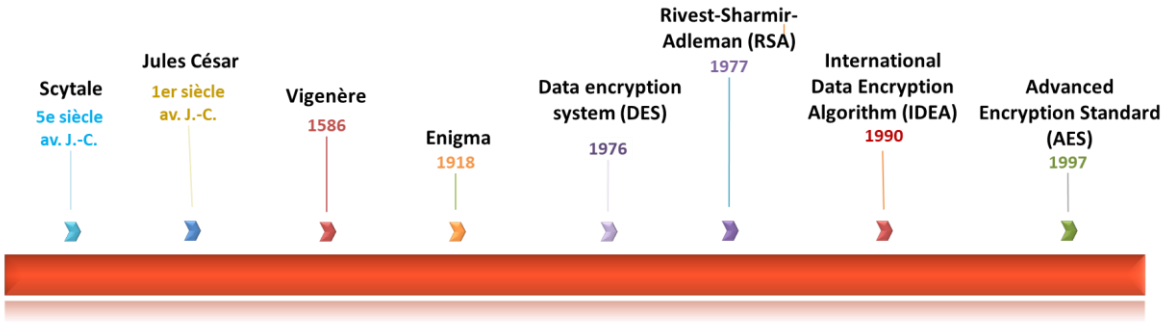

Figure 2.1 : Historique de la cryptographie

De plus, au 16e siècle, le chiffrement de Vigenère a fait son apparition. Ce processus est le même que celui de César, mais au lieu d'utiliser un décalage fixe, il se base sur une clé qui détermine le décalage pour chaque caractère. À partir du siècle dernier, la cryptographie a commencé à être de plus en plus utilisée, et par le fait même de plus en plus facile à déchiffrer (Brisson & Théberge, 2000). Ce qui a mené à la création de la machine allemande, Enigma, dont le principe de fonctionnement est simple. Il suffit seulement de presser une

lettre pour entrainer la fermeture d'un circuit électrique, ce qui va allumer une lampe qui indiquera la lettre codée. Ensuite, à chaque lettre frappée les rotors se mettent à tourner, ce qui change la substitution pour la prochaine lettre qui sera pressée. Cela permet de faire un chiffrement qui est réversible, donc la saisie du message chiffré va nous donner le message clair (Brisson & Théberge, 2000). En 1977, le standard de chiffrement de données DES (Data Encryption Standard) qui était créé à l'origine par IBM (International Business Machines) est proposé comme standard par l'organisme NIST (National Institute of Standards and Technology).

Le standard de chiffrement de données DES est un algorithme de cryptage symétrique qui fonctionne par blocs; il découpe virtuellement le texte clair en blocs de 64 bits qu'il code séparément. On peut décrire l'algorithme comme assez simple, puisqu'il ne combine que des permutations et des substitutions. Il est en faite un algorithme de cryptage à clé secrète. Ce qui veut dire que la clé sert à la fois à crypter et à décrypter le message. Plus précisément, elle a une longueur de 64 bits, c'est-à-dire 8 octets, mais seulement 56 bits sont utilisés pour chiffrer le message et les autres 8 bits de parité servant en réalité à vérifier l'intégrité de la clé. De plus, la clé possédant 64 bits est utilisée pour générer 16 autres clés de 48 bits chacune, qui seront utilisés lors de chacune des 16 itérations du DES (Blanc & De Georges, 2004). D'ailleurs, dans la même année l'algorithme RSA qui signifie Rivest-Sharmir-Adleman, soient les noms de ses inventeurs Ron Rivest, Adi Shamir et Léonard Adleman (Rivest *et al*., 1978) a fait son apparition à travers les nouvelles méthodes de chiffrement.

Le chiffrement RSA (Rivest-Sharmir-Adleman) est pour sa part un algorithme de chiffrement asymétrique. Pour fonctionner, il utilise une paire de clés (des nombres entiers) composée d'une clé publique pour chiffrer et d'une clé privée pour déchiffrer des données confidentielles. La personne qui souhaite recevoir un message confidentiel va créer la paire de clés, la clé publique va être envoyée aux correspondants pour chiffrer le message et l'ensemble des clés, soit les deux, vont servir à déchiffrer le message reçu. Aussi, la clé privée peut être utilisée pour signer des messages, la clé publique va alors permettre à ses correspondants de vérifier la signature (Henry-Labordère, 2021).

Par la suite, le chiffrement IDEA (International Data Encryption Algorithm) a été conçu par Xuejia Lai et James Massey en 1991, à Zurich en Suisse (Lai & Massey, 1991). Il est actuellement considéré parmi les systèmes les plus performants. Le principe de fonctionnement de cet algorithme est un chiffrement par blocs de 64bits, la longueur de la clé est de 128 bits, c'est-à-dire le double du système de chiffrement de données DES (Data Encryption System). L'algorithme IDEA comporte trois opérations élémentaires : « le OU exclusif ou (XOR); l'addition modulo 2^{16}; la multiplication modulo $2^{16} + 1$». Cependant, dû aux multiplications qui sont plus lentes, le système IDEA est moins rapide que le système DES (Bartz & Markey, 2012).

En 1997, Joan Daemen et Vincent Rijmen ont proposé un système de cryptage avancé AES (*Advanced Encryption Standard*), soit un système de chiffrement basé sur le système Rijndael. Il a été adopté comme norme fédérale de traitement de l'information par l'institut national des normes et de la technologie NIST (National Institute of Standards and Technology) en 2001 (Dworkin *et al.*, 2001). Le standard AES permet un chiffrement plus rapide et plus sécurisé que les autres systèmes, tels que le système DES (Data Encryption System) et le TDES (Triple Data Encryption System) (Aleisa, 2015), ce qui le rend parfaitement adapté pour les applications de systèmes embarqués. L'algorithme AES est moins complexe et nécessite peu de mémoire, donc plus facile à implanter. Il utilise des blocs et des clés de différentes tailles 128, 192 ou 256. Chaque bloc de message est traité dans un processus de 10, 12 ou 14 tours sur des clés respectivement de 128, 192 ou 256 bits. Chaque tour utilise une sous-clé différente et il est composé de quatre étapes : *Substitution d'octets, Décalage des lignes, Mélange des colonnes et Ajout d'une clé ronde* (Daemen & Rijmen, 2020). Il est considéré actuellement comme le protocole de chiffrement symétrique le plus utilisé dans le domaine.

2.3 MÉTHODES DE CRYPTOGRAPHIE

La cryptographie est une méthode de sécurité qui consiste à chiffrer et déchiffrer les données afin de les protéger contre les utilisateurs non autorisés. On y distingue notamment deux principaux types de chiffrement de données : le chiffrement à clé symétrique et le chiffrement à clé asymétrique. Ils se différencient au niveau de la méthode dont les données sont déchiffrées.

2.3.1 Chiffrement à clé symétrique

Dans le chiffrement à clé symétrique, une seule clé est responsable du chiffrement et du déchiffrement d'un message ou d'un fichier (Figure 2.2). Les parties concernées partagent cette clé secrète et peuvent l'utiliser pour chiffrer ou déchiffrer les messages de leur choix (Blanc & De Georges, 2004).

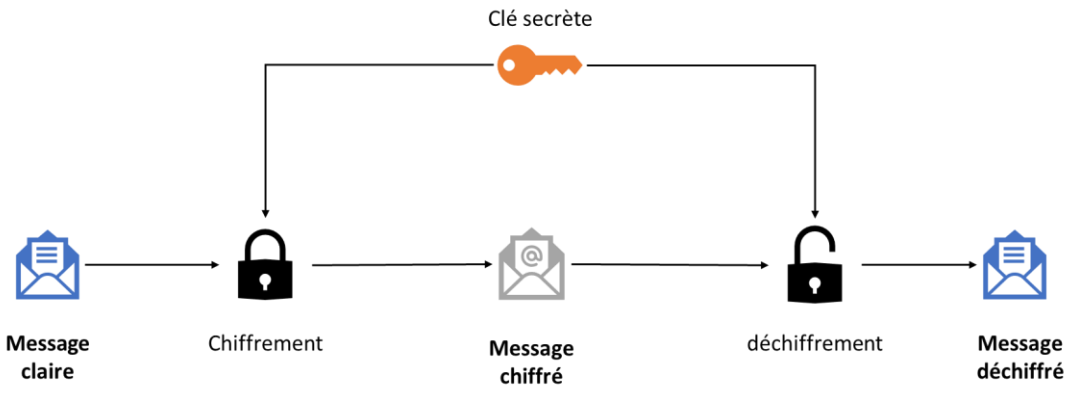

Figure 2.2 : Principe de fonctionnement de chiffrement symétrique

2.3.2 Chiffrement à clé asymétrique

Le chiffrement asymétrique est un moyen de chiffrement des données qui utilise deux clés, une clé publique et une clé privée (Figure 2.3). Par définition, la clé publique peut être partagée avec n'importe qui, mais il est crucial que la clé privée reste privée et inconnue de quiconque (Thawte, 2013).

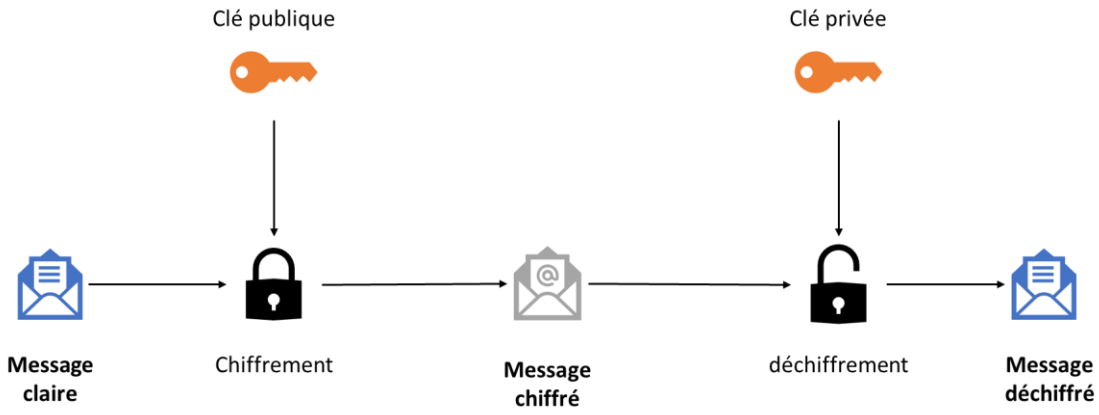

Figure 2.3 : Principe de fonctionnement de chiffrement asymétrique

2.4 COMPARAISON ENTRE LES DEUX CHIFFREMENTS À CLÉ SYMÉTRIQUE ET ASYMÉTRIQUE

Les chiffrements symétriques et asymétriques présentent chacun des avantages et des inconvénients (Maqsood *et al.*, 2017). Les algorithmes de chiffrement symétrique présentent l'avantage d'être sûrs, efficaces, rapides et capables de chiffrer et de déchiffrer de grandes quantités de données en un temps record. De plus, ils consomment peu de ressources matérielles. Cependant, le fait qu'une clé commune soit partagée est considéré comme un inconvénient, car la gestion des clés devient problématique.

L'inconvénient de chiffrement asymétrique est que cela prend beaucoup de temps et consomme plus de ressources que le chiffrement symétrique; il ne convient pas aux grandes quantités de données. Si vous perdez votre clé privée, vous ne saurez jamais être en mesure de la récupérer. Cependant, il présente l'avantage de prendre en charge la signature

numérique, qui permet de vérifier l'identité du destinataire. De plus, il est simple de gérer les clés d'une manière qui n'oblige pas l'utilisateur de partager sa clé privée. Il garantit la confidentialité, l'authenticité et la non-répudiation.

2.5 CHIFFREMENT PAR BLOC

Le chiffrement par bloc est un processus qui permet de crypter des données en blocs. Il prend un message et le décompose en blocs de taille fixe qu'il code séparément. Il se caractérise par un degré élevé de sensibilité à la propagation d'erreurs inter-symboles. Chaque erreur de symbole dans un bloc de texte clair ou chiffré peut gravement altérer le chiffrement ou le déchiffrement des blocs voisins (Blanc & De Georges, 2004).

2.6 CHIFFREMENT PAR FLOT

Le chiffrement par flot, chiffrement par flux ou chiffrement continu, est un chiffrement basé sur un générateur de nombres pseudo aléatoires et un mécanisme de substitution bit à bit, qui opère sur un flux de données. Il arrive à traiter les données de longueur quelconque et n'a pas besoin de les découper. Il opère avec une fonction XOR entre un bit à la sortie du générateur et un bit provenant des données, toutefois, la fonction XOR n'est pas la seule opération possible. L'idée est de concevoir un système capable de chiffrer un message en un seul passage pour qu'il puisse être utilisé en continu, de sorte que le texte chiffré est produit au fur et à mesure que le traitement du message se poursuit. Le texte chiffré est généralement crypté en une seule passe, mais il est possible d'utiliser un chiffrement par bloc de manière continue (Bénony, 2006; Naya-Plasencia, 2009).

2.7 CHIFFREMENT QUANTIQUE

Le chiffrement quantique est basé sur les propriétés de la physique quantique qui permet de sécuriser le transfert des informations entre deux personnes. La cryptographie quantique repose sur la transmission des éléments quantiques, comme le photon qui possède une des caractéristiques fondamentales de la physique quantique; c'est ce qu'on appelle le principe de superposition (Bhushan *et al*., 2022). Dans l'exemple de la distribution de la clé quantique, l'un des participants envoie à l'autre une série de photon qui sont dans un état superposé de polarisation. Si un espion essaye de voler la clé, alors il devra mesurer la polarisation des photons et donc la modifier, puisque toute mesure provoque une réduction des variables d'état. Cela se manifestera dans le signal et les deux participants doivent immédiatement se rendre compte.

2.8 FONCTION DE HACHAGE

Une fonction de hachage est un algorithme qui prend une entrée (par exemple un message) de n'importe quelle taille et produit une sortie de taille fixe, appelée valeur de hachage. Cette taille est généralement comprise entre 128 et 512 bits. La même entrée donnera toujours la même sortie. Les fonctions de hachage sont appelées fonctions « unidirectionnelles », car elles sont faciles à calculer, mais impossibles à inverser. Par conséquent, il est impossible de retrouver le message original qui a produit la valeur de hachage. Les messages peuvent être de n'importe quelle forme, y compris du texte, des chiffres, du son ou de l'image. Cependant, les données utilisées dans le hachage doivent être converties en une séquence binaire de bits avant d'être traitées par la fonction de hachage. Une fonction de hachage est similaire à une empreinte digitale en ce sens que les deux peuvent être utilisées à des fins d'identification ou de vérification. La sécurité du hachage dépend de la difficulté à produire des collisions (différentes entrées produisant la même

sortie). Pour être considérée comme sûre, une fonction de hachage doit résister aux collisions (Boura, 2012).

Les technologies de chiffrement de données ont évolué à une vitesse vertigineuse menant à des algorithmes beaucoup plus complexes et puissants, qui sont capables de cacher des données confidentielles dans un environnement numérique libre de toute surveillance. Les études ont révélé que la cryptographie est une science ancienne et puissante qui a permis à des personnes de cacher des informations confidentielles.

Au cours de l'histoire, les évolutions technologiques ont permis l'adoption de méthodes de chiffrement qui auraient été impossibles à réaliser auparavant. La cryptographie moderne est construite sur des principes fondamentaux qui sont restés les mêmes depuis les débuts de l'histoire de la cryptographie. Ces principes sont l'anonymat et la confidentialité. En d'autres termes, la cryptographie permettrait de cacher des informations confidentielles avec l'intention qu'elles soient déchiffrées par des personnes à des fins autres que les fondamentales de la cryptographie. Malgré la complexité des algorithmes de cryptage, il est possible avec une certaine expérience, de les déchiffrer et de les comprendre. Ces systèmes de chiffrement n'ont pas été créés pour rester secrets, mais pour permettre aux données de rester confidentielles. Néanmoins, il est important de garder à l'esprit que les algorithmes de cryptage ne sont pas faciles à déchiffrer, mais si l'on dispose des outils appropriés, il est possible de les analyser et de les déchiffrer.

À l'avenir, les technologies de chiffrement devraient continuer à évoluer et s'améliorer, car les gigantesques progrès accomplis dans la cryptographie révèlent les failles des systèmes de chiffrement actuels. Les développeurs de nouveaux algorithmes devraient tenir compte de cette réalité, par conséquent, trouver des méthodes pour la rendre moins vulnérable.

2.9 LA THÉORIE DU SYSTÈME DE CHIFFREMENT AVANCÉ AES

Le système de chiffrement avancé AES (*Advanced Encryption Standard*) est une méthode de chiffrement utilisée pour protéger les données ou les communications. Il s'agit d'un algorithme à clé symétrique, ce qui signifie que la même clé est utilisée pour le chiffrement et le déchiffrement. Le système AES utilise un type particulier de chiffrement; la clé est utilisée pour mélanger les données de manière à ce qu'elles puissent être stockées ou transférées en toute sécurité sur un réseau où seules les personnes possédant la clé peuvent déchiffrer les données.

Le système AES a été développé en 1998 par deux cryptographes belges, Joen Daemen et Vincent Rijmen (Daemen & Rijmen, 2020), lors d'une compétition organisée par le NIST (*National Institute of Standard and Technology*) pour trouver une nouvelle méthode de chiffrement sécurisée. Lors du processus de sélection, l'algorithme AES était l'un des finalistes de la compétition. Par la suite, l'AES est devenu le standard de chiffrement pour les données à haute sécurité le plus utilisé dans le monde, car il est considéré comme étant un algorithme de chiffrement fiable.

2.10 PRINCIPE DE FONCTIONNEMENT

Le système de chiffrement avancé AES est basé sur un algorithme de chiffrement par blocs qui utilise une clé de chiffrement pour transformer un bloc de données en un bloc de données chiffrées. Cette technique consiste à diviser les données à chiffrer en blocs de 128 bits arrangés en une matrice de 4×4 octets, puis à chiffrer chaque bloc de données séparément à l'aide de la clé de chiffrement en suivant exactement les mêmes étapes. Les blocs de données chiffrés sont ensuite concaténés pour former le message chiffré. Si le message n'est pas divisible par la longueur du bloc, on rajoute des données jusqu'à ce que le message soit divisible. Cette opération est appelée rembourrage (padding).

Pour déchiffrer un message chiffré, le système de chiffrement avancé AES utilise la même technique de « chiffrement par blocs ». Cette fois, cependant, les blocs de données chiffrés sont déchiffrés à l'aide de la clé de chiffrement, puis concaténés pour former le message déchiffré.

L'algorithme AES est actuellement le plus utilisé et le plus fiable. Il utilise peu de mémoire, moins complexe, et donc plus facile à implanter. Il offre des tailles de blocs et de clés de 128, 192 et 256 bits comme indiqué sur Figure 2.4. Chaque bloc de message est traité dans un processus de 10, 12 ou 14 tours sur des clés respectivement de 128, 192, 256 bits (Daemen & Rijmen, 2020) comme le montre le Tableau 1.1.

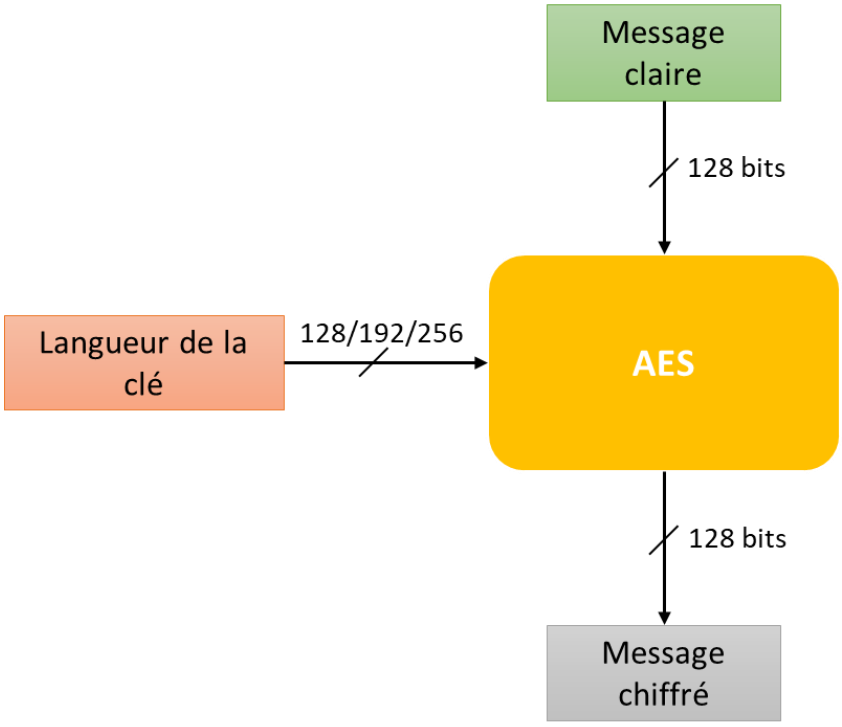

Figure 2.4 : Schéma représentatif du système AES

Tableau 1.1 : Différentes versions de l'algorithme AES (Daemen & Rijmen, 2020).

AES Version	**Longueur de la clé (Nk Mots)**	**Taille du bloc (Nb Mots)**	**Nombre de Tours (Nr)**	**Nombre de clés**
AES-128	4	4	10	11
AES-192	6	4	12	13
AES-256	8	4	14	15

Chaque tour utilise une sous-clé différente et il est composé de quatre étapes : Substitution d'octets, Décalage des lignes, Mélange des colonnes et Ajout d'une clé ronde.

La Figure 2.5 résume le fonctionnement de système de chiffrement avancé AES-128. Le tour initial utilise seulement l'opération AddRoundKey, ce qui signifie l'ajout de la clé au bloc de données. Pour les tours qui suivent, les données passent par les étapes (SubBytes, ShiftRows, MixColumns et AddRoundKey) 9 fois de suite avec une clé différente à chaque tour. Pour le dernier tour, l'algorithme ne passe que par les étapes de SubBytes, ShiftRows et AddRoundKey. Il laisse de côté l'étape de MixColumns. Parmi toutes les versions, le système AES-128 est souvent choisi, car il est rapide, efficace et moins susceptible aux attaques (Biryukov *et al.*, 2010).

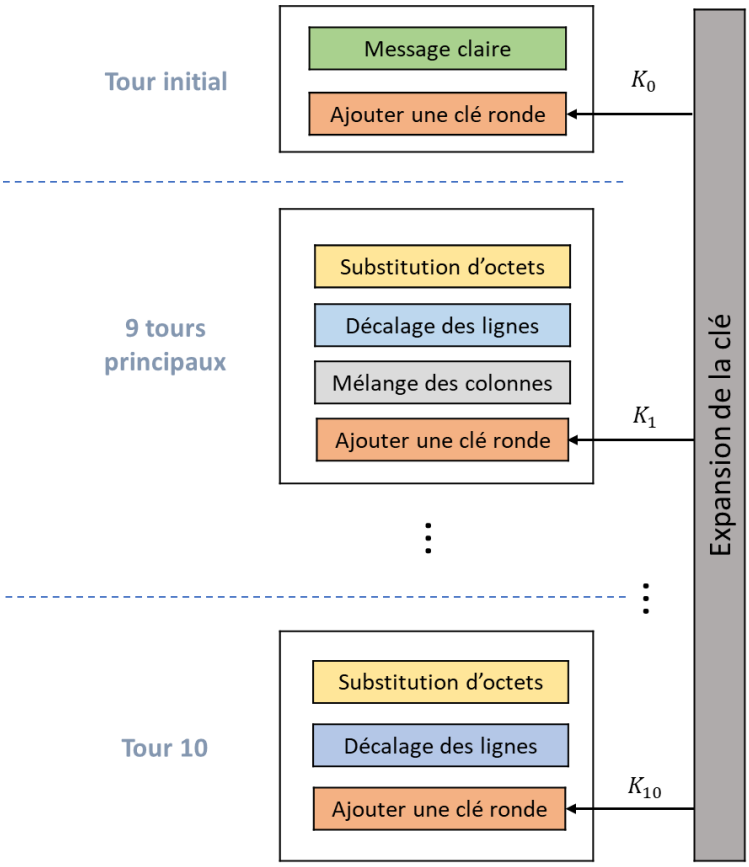

Figure 2.5 : Fonctionnements de chiffrement AES-128

2.11 SUBSTITUTION D'OCTETS (*S-BOX/SUBBYTES*)

L'opération *SubBytes* est une substitution d'octets non linéaires ou chaque octet d'entrée est remplacé par une valeur différente définie par une table de substitution nommée S-Box. La valeur de l'octet d'entrée est utilisée comme index pour rechercher la valeur de l'octet de sortie dans la table de substitution S-Box (voir la Figure 2.6). L'opération inverse (InvSubBytes) est identique à l'opération SubBytes, à l'exception qu'elle utilise la table de substitution S-Box inverse à la place (Daemen & Rijmen, 2020).

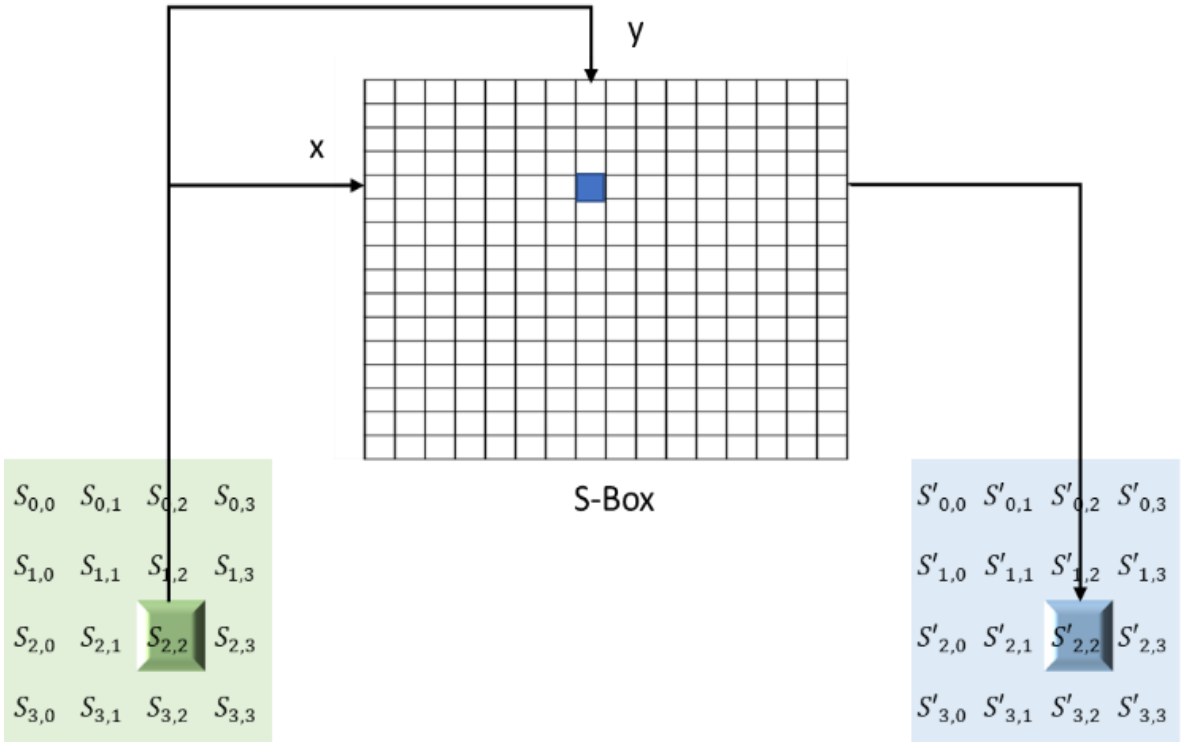

Figure 2.6 Représentation de la substitution de données

La table de substitution non linéaire utilisée dans plusieurs transformations de substitution d'octets et dans la routine d'expansion des clés pour effectuer une substitution d'une valeur d'octet par une autre. La Figure 2. 7 montre un exemple de table de substitution de système de chiffrement avancé (AES S-BOX).

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
x	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figure 2. 7 : Table de substitution (AES S-BOX)

2.12 DÉCALAGE DE LIGNES (*SHIFTROWS*)

Dans l'opération *ShiftRows*, chaque ligne du tableau d'états est cycliquement décalée vers la gauche d'un pas défini. La première ligne reste inchangée, la deuxième ligne est

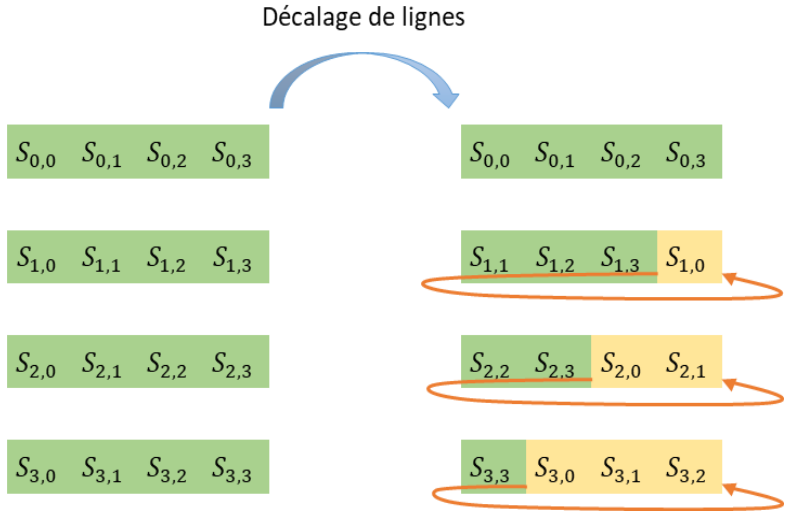

Figure 2.8 : Représentation de décalage cyclique de données (ShiftRows)

décalée d'un pas, la troisième ligne est décalée de deux pas et la quatrième ligne est décalée de trois pas comme indiqué sur la Figure 2.8. L'opération *InvShiftRows* est identique à l'opération *ShiftRows*, à l'exception du fait que chaque ligne est décalée cycliquement vers la droite.

2.13 MÉLANGE DES COLONNES (*MIXCOLUMNS*)

La transformation *MixColumns* opère sur le bloc de données, colonne par colonne, en traitant chaque colonne comme un polynôme sur Galois Field GF (2^8) et en la multipliant par un polynôme fixe (Daemen & Rijmen, 2020). La transformation est spécifiée par la matrice suivante où les valeurs sont en hexadécimal :

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.1}$$

L'état est multiplié par chaque ligne de la matrice, et les résultats sont combinés en utilisant XOR. De cette façon, il opère sur la colonne entière.

L'opération inverse est identique, mais ça va être multiplié par la matrice inverse suivante où les valeurs sont en hexadécimal :

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} \tag{2.2}$$

2.14 ADDITION D'UNE CLÉ RONDE (*ADDROUNDKEY*)

Dans cette transformation, une clé circulaire est ajoutée au bloc de message par une simple opération XOR, comme indiqué sur la Figure 2.9. La longueur de la clé est égale à la

taille du bloc de message, c'est-à-dire, que pour N_b = 4, la longueur de la clé ronde est égale à 128 bits/16 octets.

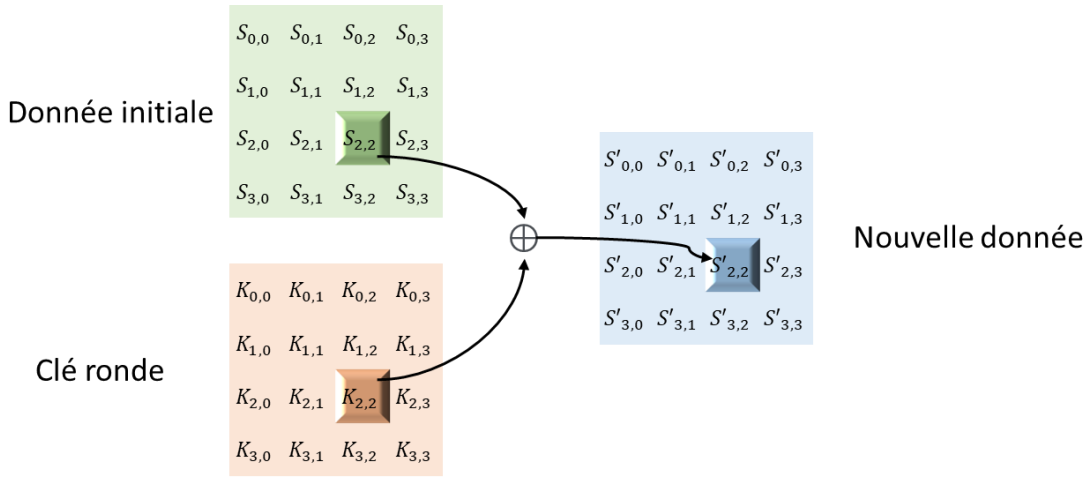

Figure 2.9 :. Représente l'ajout de la clé circulaire au bloc de données

2.15 EXPANSION DE LA CLÉ (*KEY EXPANSION*)

Comme indiqué à la Figure 2.10, cette opération génère un ensemble de clés rondes à chaque tour contenant un total $N_b\ (N_r + 1)$ mots. Ces clés sont dérivées de la clé de chiffrement originale pour les deux opérations (chiffrement et déchiffrement).

Le nombre N_b dans notre opération, signifie la longueur du bloc, en termes de mots, qui est égal à 4 (mots de 32 bits ou 4 octets) composant l'état. Dans notre ensemble, il y a aussi le nombre N_k qui correspond à la longueur de la clé divisée par 32. Cela indique le nombre de mots de 32 bits constituant la clé de chiffrement. Les clés de 128, 192 ou 256 bits correspondent respectivement aux valeurs 4, 6 ou 8 du nombre N_k.

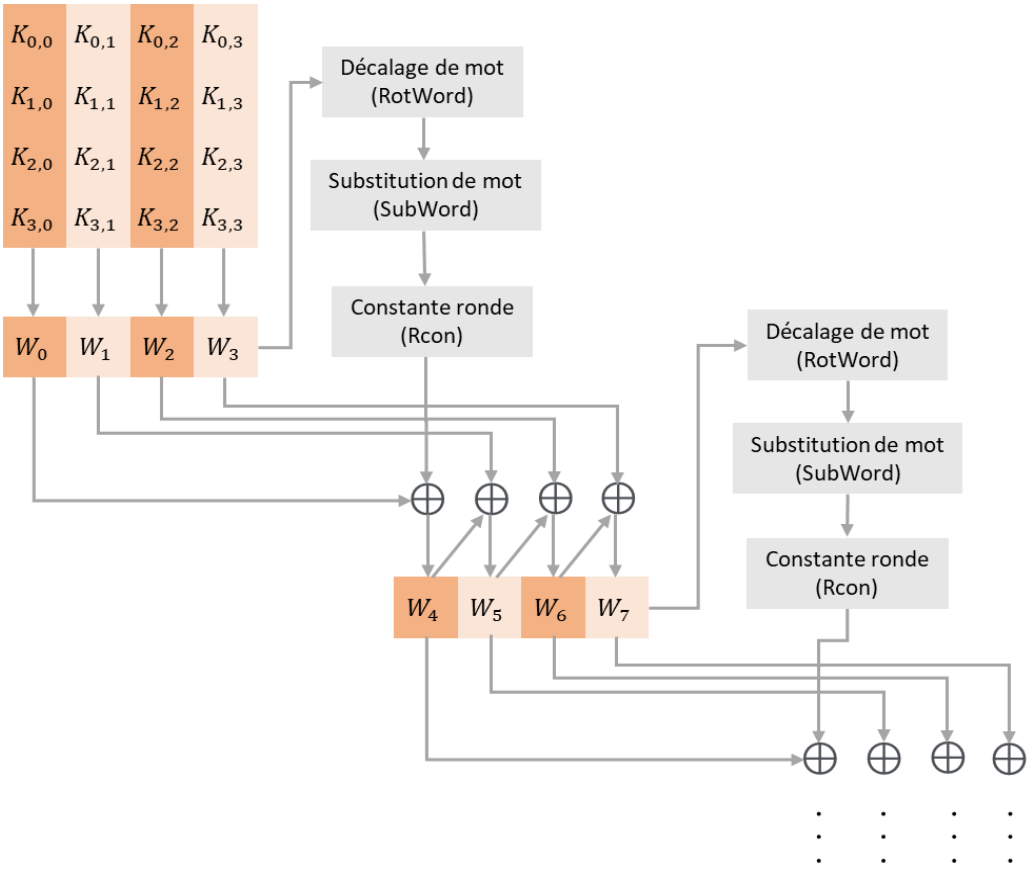

Figure 2.10 :. Représentation de l'expansion de clé

Pour terminer, on retrouve le nombre de tour N_r qui fonction de N_k et N_b. Pour N_k soit égale à 4, 6 ou 8, cela implique que N_r est respectivement égale à 10, 12, ou 14 tours.

$$N_r = 6 + max\ (N_b, N_k) \qquad (2.3)$$

La clé est divisée en 4 mots (W_0, W_1, W_2, W_3), où chaque mot est constitué de 4 octets. Le mot W_3 est copié avant de passer par les trois opérations; décalage de mot (RotWord), substitution de mot (*SubWord*) et la constante ronde Rcon (Round constant). Ensuite, les résultats sont soumis à une opération XOR avec le mot (W_0), afin d'avoir le premier mot (W_4) de la nouvelle clé. Ensuite, le mot W_4 sera soumis à une opération XOR avec le

deuxième mot W_1, afin d'obtenir le deuxième mot W_5 de la nouvelle clé. La même opération se répète jusqu'au quatrième mot W_3 qui sera soumis à une opération XOR avec le résultat du troisième mot W_6, afin de compléter la nouvelle clé qui est constituée aussi de quatre mots (W_4, W_5, W_6, W_7). Cette dernière sera utilisée dans le premier tour. La même opération sera répétée avec chaque nouvelle clé afin d'avoir 44 mots; ce qui signifie que 11 clés seront utilisées dans les 10 tours de chiffrement AES.

L'expansion de clé passe par plusieurs opérations, telles que RotWord, SubWord et Rcon. La fonction de décalage de mot *RotWord* fait un décalage, à gauche, de chaque octet d'un mot de la clé ronde de 32 bits. La fonction substitution de mot *SubWord* fait une substitution par octet des données d'un mot de la clé donner en table de substitution S-Box. L'opération *Rcon* est définie comme suit :

$$\text{Rcon(i)} = (\text{R[i]}, 0, 0, 0) \qquad \text{avec} \begin{cases} \text{R}[1] = 1 \\ \text{R}[\text{i}] = 2 * \text{R}[\text{i} - 1] \end{cases} \tag{2.4}$$

Chaque octet est substitué par une valeur constante ronde (Rcon[i]), où i représente le nombre de tours spécifié par la longueur de la clé. La table Rcon présentée dans le Tableau 2.1.

Tableau 2.1 : La table de la constante ronde (Rcon)

i	1	2	3	4	5	6	7	8	9	10
Rcon[i]	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1b	0x36

2.16 MODES D'OPÉRATION AES

L'algorithme AES possède différents modes de fonctionnement basés sur trois tailles de clé différentes qui peuvent être utilisées pour la sécurité de données. Les modes

d'opération les plus courants sont ECB, CBC, OFB, CFB et CTR; ce dernier est le plus simple et le plus populaire (Mohan et al., 2011).

Le mode ECB est l'élément de base utilisé dans la construction des autres modes CBC, OFB, CFB et CTR. Dans ces modes, un vecteur d'initialisation est utilisé pour les alimenter (Reddy, 2012). Dans ce qui suit, nous présentons les différents modes de fonctionnement utilisé dans l'implantation de l'algorithme AES.

2.16.1 Dictionnaire de codes ECB (*Electronic codebook*)

Le mode ECB (*Electronic CodeBook*) est le plus simple des modes de chiffrement. Dans ce mode, chaque bloc de texte en clair est chiffré séparément à l'aide d'une clé secrète (Figures 2.11 et 2.12). Cependant, les mêmes blocs de texte clair produisent toujours les mêmes blocs de texte crypté. Le texte en clair peut donc être facilement retrouvé en disposant des blocs chiffrés (Menezes et al., 1996).

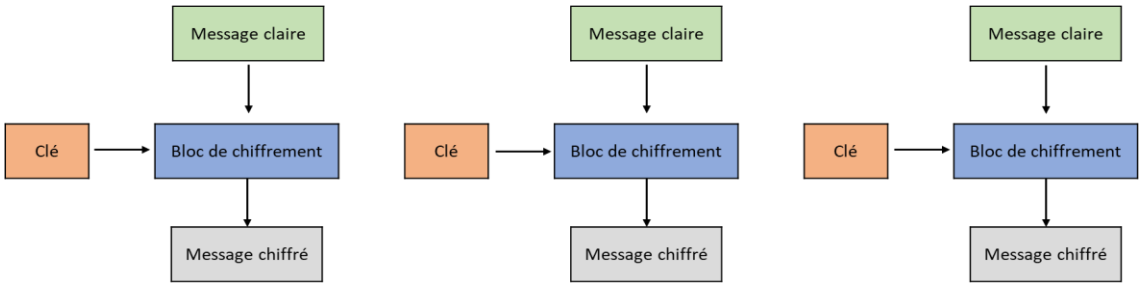

Figure 2.11 : Mode de chiffrement ECB illustré par 3 blocs

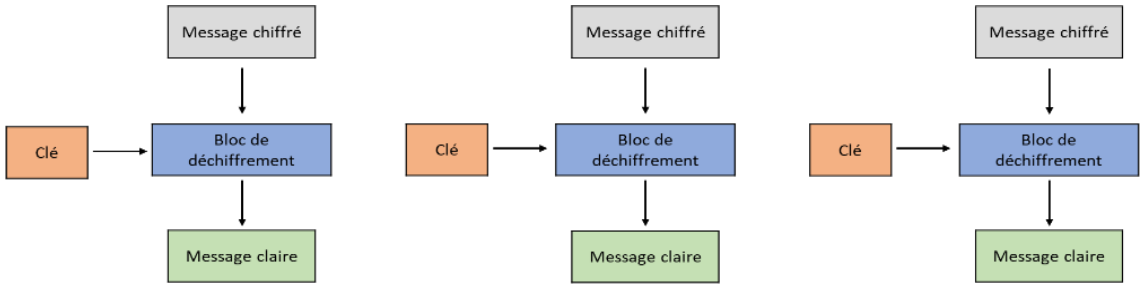

Figure 2.12 : Mode de déchiffrement ECB illustré par 3 blocs

2.16.2 Enchainement de blocs de chiffrement CBC (*Cipher Block Chaining*)

Le mode CBC (*Cipher Block Chaining*) est le mode de chiffrement le plus répandu. Le principe de fonctionnement est très simple, comme illustré dans la Figure 2.13. Le message clair est soumis à une opération XOR avec le vecteur d'initialisation, puis chiffré avec la clé. Le message clair suivant est soumis à l'opération XOR avec le premier bloc chiffré, puis chiffré avec la clé. Le bloc du message clair suivant est soumis à l'opération XOR avec le deuxième bloc chiffré, puis chiffré avec la clé, etc. Dans ce mode, même si nous chiffrons le même bloc de texte en clair, nous obtiendrons un bloc de texte chiffré différent.

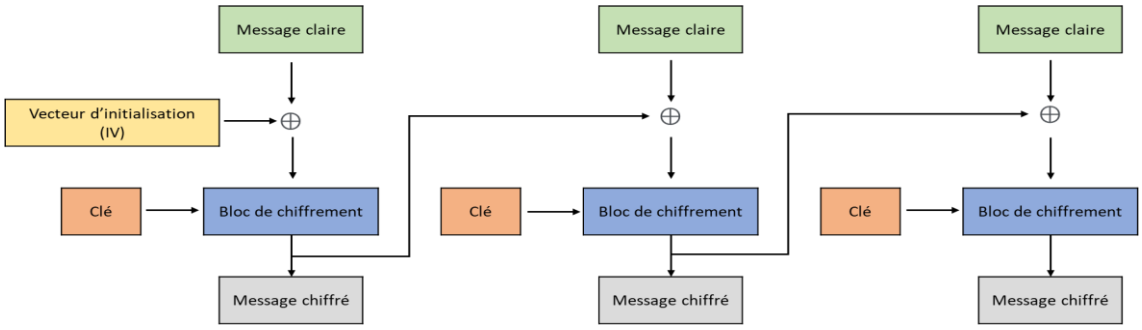

Figure 2.13 : Mode de chiffrement CBC

Lors de déchiffrement, le destinataire réalise les mêmes opérations en sens inverse (Figure 2.14). Le vecteur d'initialisation est généralement transmis en clair au destinataire.

Nous pouvons déchiffrer les données en parallèle, mais ce n'est pas possible lors du chiffrement des données, car l'entrée du prochain bloc dépend de la sortie du bloc précédent. Le mode CBC est sécurisé, car il est impossible de déchiffrer un bloc sans connaître le bloc précédent. Cependant, si un bloc de texte chiffré est cassé, cela affectera tous les blocs suivants.

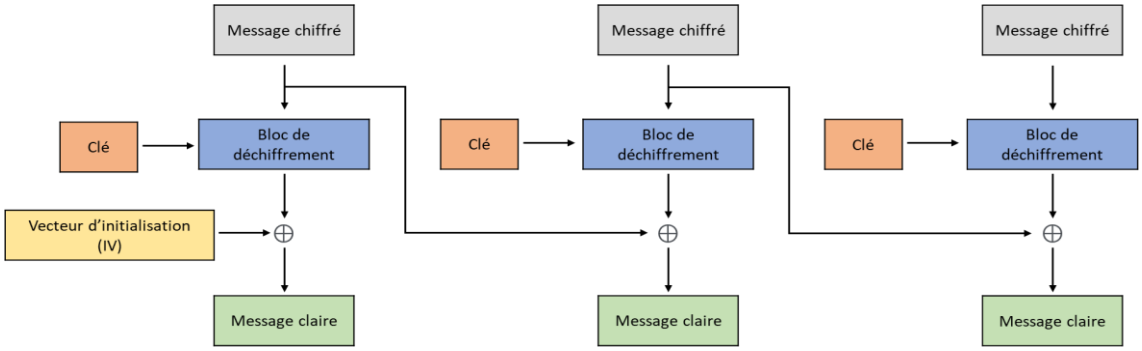

Figure 2.14 : Mode de déchiffrement CBC

2.16.3 Chiffrement à rétroaction CFB (*Cipher Feedback*)

Le mode de chiffrement à rétroaction CFB (*Cipher Feedback*) est un mode de chiffrement par flux. Nous ne pouvons pas chiffrer les données en parallèle, car les informations chiffrées dépendent les unes des autres. Cependant, nous pouvons déchiffrer les données en parallèle. Ce mode utilise un opérateur de XOR pour chiffrer le bloc de message clair avec un flux de chiffrement.

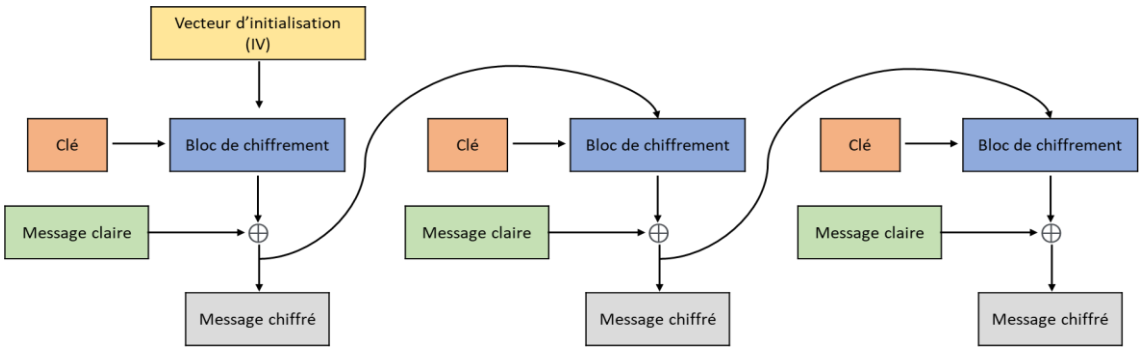

Figure 2.15: Mode de chiffrement CFB

Le principe de fonctionnement du mode CFB est le suivant : le vecteur d'initialisation est chiffré avec une clé secrète, puis le résultant est soumis à l'opération XOR avec le message clair qui est divisé en blocs. Ensuite, le résultat est chiffré avec une clé afin d'être soumis à l'opération XOR avec le deuxième bloc du message clair. Nous avons alors un nouveau bloc encore plus sombre que le précédent. Alors, nous appliquons la même technique aux autres blocs (Figure 2.15). Cela signifie que chaque bloc de texte chiffré contient des informations sur les blocs précédents, ce qui rend la tâche plus difficile lors d'une attaque. Le message chiffré sera envoyé au destinataire, qui déchiffre le message en faisant l'opération inverse (Figure 2.16).

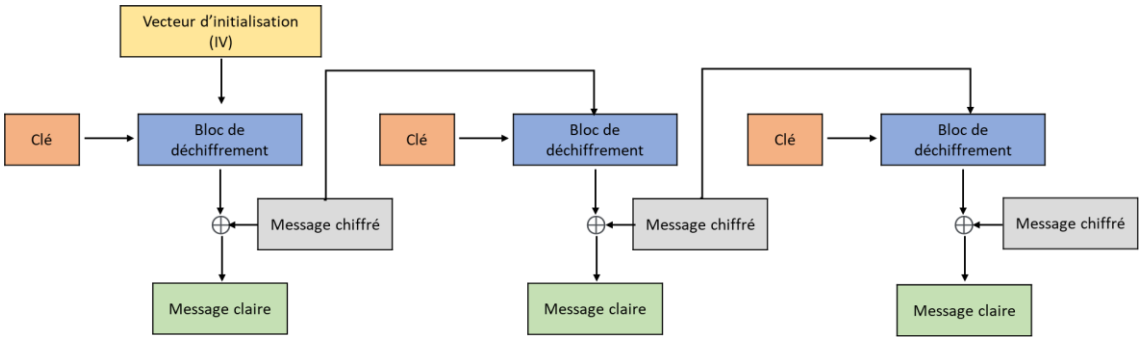

Figure 2.16 : Mode de déchiffrement CFB

2.16.4 Chiffrement à rétroaction de sortie OFB (*Output Feedback*)

Le mode chiffrement à rétroaction de sortie OFB (*Output Feedback*) est un mode de chiffrement par flux. Il est similaire au mode CFB, mais nous ne pouvons pas chiffrer ou déchiffrer les données en parallèle, car les informations dépendent les unes des autres.

Ce mode génère un bloc de flux à partir de vecteur d'initialisation chiffré avec la clé, qui est ensuite soumis à une opération XOR avec le bloc de message en clair pour obtenir le texte chiffré. Le bloc suivant sera généré à partir du bloc de flux précédent de vecteur d'initialisation et la clé qui va passer par le chiffrement et ensuite le résultat sera soumis à

l'opération XOR avec le deuxième bloc de message clair (Figure 2.17), et ainsi de suite. Lors de déchiffrement la même opération sera appliquée pour avoir le message clair (Figure 2.18).

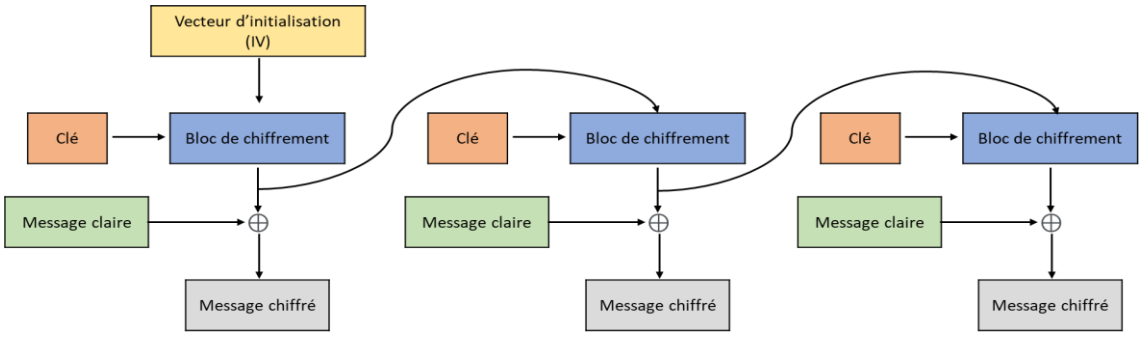

Figure 2.17 : Mode de chiffrement OFB

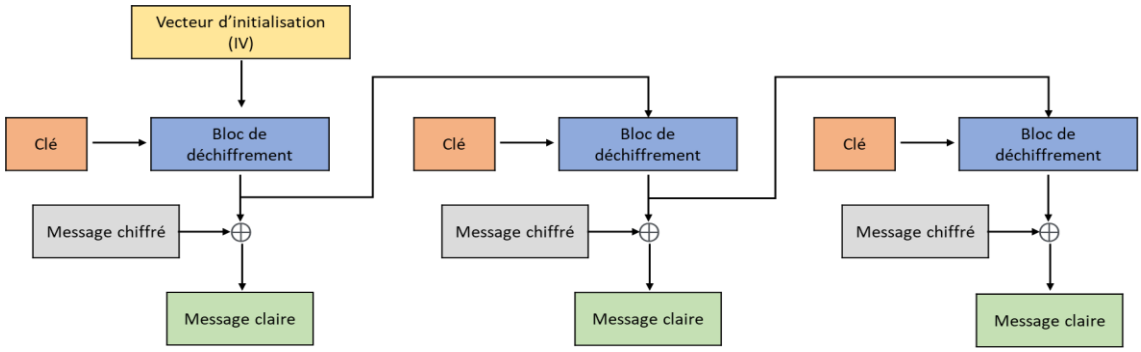

Figure 2.18 : Mode de déchiffrement OFB

2.16.5 Chiffrement basé sur un compteur CTR (*Counter mode*)

Le mode de chiffrement basé sur un compteur CTR (*Counter mode*) est aussi un chiffreur de flux qui est assez performant. Il utilise un compteur pour chiffrer le message clair. Le compteur est initialisé à zéro et est incrémenté à chaque bloc chiffré.

Son mode de fonctionnement est proche du mode CBC, mais avec une différence notable : le chiffrement de chaque bloc clair n'est pas précédé d'une opération XOR avec le

résultat du chiffrement du bloc précédent, ce qui permet de traiter les blocs de manière indépendante et parallèle (Figure 2.19).

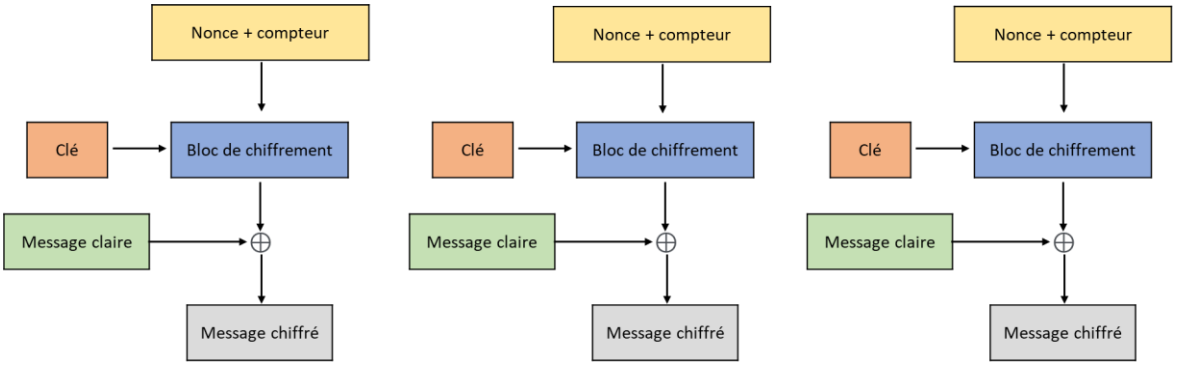

Figure 2.19 : Mode de chiffrement CTR illustré par 3 blocs

Le chiffrement d'un bloc clair n'est donc pas influencé par les blocs précédents. Le compteur est chiffré avec une clé sécurisée, avant que le résultat soit soumis à l'opération XOR avec le message clair. Cette opération sera répétée jusqu'à ce que le dernier bloc de message clair ait été chiffré. Le déchiffrement est le processus inverse, comme illustré dans la Figure 2.20. Le bloc de texte chiffré est soumis à l'opération XOR avec la sortie de valeur de compteur chiffrée.

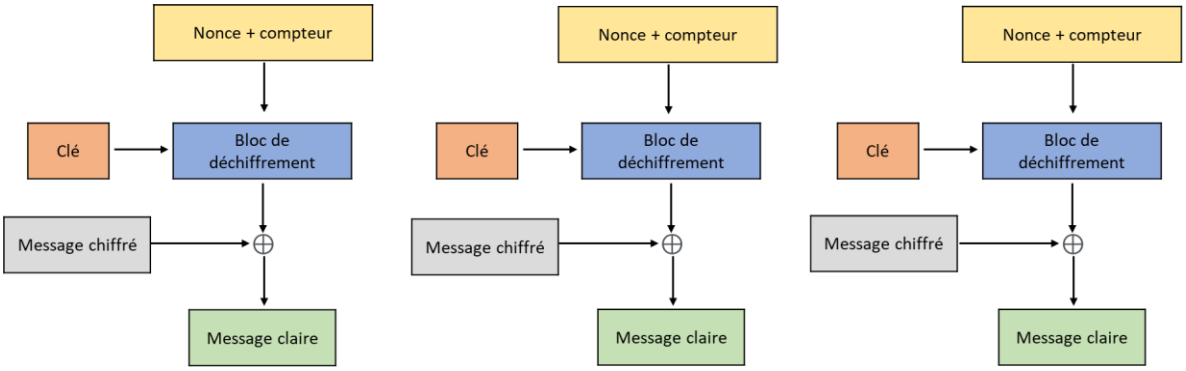

Figure 2.20 : Mode de déchiffrement CTR illustré par 3 blocs

Le Tableau 2.2 résume les modes de chiffrement les plus courants. Le mode dictionnaire de codes (ECB) est le mode le plus simple, mais il est aussi le plus vulnérable. Le mode d'enchainement de blocs de chiffrement (CBC) est le mode de chiffrement par bloc le plus couramment utilisé. Il est relativement sécurisé et offre un bon niveau de sécurité. Le mode de chiffrement à rétroaction (CFB) et le mode de chiffrement à rétroaction de sortie (OFB) sont deux modes de chiffrement assez sécurisés, cependant, ils sont un peu plus difficiles à mettre en œuvre que les modes CBC ou CTR. Pour finir, le mode de chiffrement basé sur un compteur (CTR) est facile à implanter et très sécurisé.

Tableau 2.2 : Comparaison des modes de chiffrement AES.

Modes d'opération de chiffrement	Chiffrement parallèle	Déchiffrement parallèle	Accès en lecture aléatoire
ECB	Oui	Oui	Oui
CBC	Non	Oui	Oui
CFB	Non	Oui	Oui
OFB	Non	Non	Non
CTR	Oui	Oui	Oui

CHAPITRE 3
IMPLANTATION DU SYSTÈME DE CHIFFREMENT AVANCÉ AES

L'implantation matérielle d'algorithmes de traitement numérique des signaux et des images sur puce électronique est un domaine de recherche et développement avancé qui tente de résoudre certaines des difficultés rencontrées par les systèmes existants de traitement de l'information. Dans ce chapitre, nous allons présenter brièvement les circuits FPGA, leurs avantages et inconvénients aussi que leurs domaines d'utilisation. Nous allons également présenter les différentes étapes de l'implantation matérielle du système de chiffrement avancé AES (*Advanced Encryption Standard*).

3.1 GÉNÉRALITÉ SUR LES FPGA

Les premiers circuits de réseaux de portes logiques programmables FPGA (*Field Programmable Gate Arrays*) ont été développés dans les années 80 (Rodriguez-Andina *et al.*, 2007). Les circuits FPGA se composent d'un ensemble de blocs logiques programmables CLB (*Configurable Logic Block*) reliés entre eux par des interconnexions configurables. Ils contiennent aussi des blocs d'entrées sorties IOB (*Input/Output Block*). Les circuits FPGA permettent de réaliser des circuits complexes de manière relativement simple et efficace (Deschamps *et al*., 2006). Ils sont utilisés dans les applications des systèmes embarqués au calcul haute performance HPC (High Performance Computing). Ils sont également utilisés dans le traitement numérique des signaux, les transferts de données et les systèmes de télécommunications.

Les circuits FPGA constituent une alternative très intéressante aux processeurs traditionnels, en particulier pour les applications nécessitant une forte puissance de calcul comme dans le domaine de traitement de signaux. Aussi, ils sont capables de fonctionner à des fréquences d'horloge plus élevées, ce qui les rend encore plus adaptés aux applications les plus exigeantes. Les circuits FPGA sont devenus incontournables pour le prototypage

rapide de systèmes électroniques complexes, mais aussi pour une grande variété d'applications industrielles nécessitant des systèmes évolutifs à coût et consommation réduits (Monmasson & Cirstea, 2007).

Les circuits FPGA présentent de nombreux avantages par rapport aux autres types de puces (Ricci, 2002). Parmi les avantages les plus importants, il y a la flexibilité qui permet la reprogrammation des circuits, et la vitesse de traitement élevée par rapport à d'autres types de puces. Toutefois, les circuits FPGA ont également certains inconvénients. Ils sont en effet beaucoup plus chers que les processeurs conventionnels. De plus, leur programmation nécessite une certaine expertise, ce qui limite leur utilisation courante.

Les circuits FPGA sont des dispositifs complexes dotés de millions de portes logiques et sont utilisés dans une variété d'applications. Au départ, les circuits FPGA étaient des dispositifs uniquement logiques, avec quelques ressources d'E/S de base. Au fil du temps, ils ont été dotés de plus en plus de ressources logiques et de mémoire, ainsi que des blocs DSP (*Digital Signal Processing*), et des interfaces analogiques. Ceci a permis de réduire le temps de développement. Les circuits FPGA sont ensuite devenus des systèmes sur puce SoC (System-on-Chip) avec un processeur intégré, ce qui a permis d'accélérer le temps de développement. Aujourd'hui, l'évolution de ces systèmes sur puce SoC à base de circuit FPGA a permis d'avoir le système sur puce ZYNQ (Crockett *et al*., 2014).

Le circuit ZYNQ est un système complet sur une seule puce SoC (System on Chip) qui comprend deux parties principales : un système de traitement PS (Processing system) formé autour d'un processeur ARM Cortex-A9 à double cœur et une partie logique programmable PL (Programmable Logic), qui est équivalente d'un circuit FPGA classique. Ce circuit ZYNQ dispose également des mémoires intégrées, de divers périphériques et d'interfaces de communication à haut débit. Les liens entre la partie PL et la partie PS se fondent à l'aide de l'interface AXI (*Advanced eXtensible Interface*), un standard de l'industrie (Crockett *et al*., 2014).

La solution ZYNQ est plus polyvalente qu'un circuit FPGA classique. En effet, le ZYNQ est flexible et beaucoup plus puissant qu'un circuit FPGA classique. Ce circuit peut être utilisé pour une grande variété d'applications, notamment le traitement vidéo et le traitement d'images (Crockett *et al.*, 2014). Comme indiqué à la Figure 3.1, la différence entre les deux étant que le circuit FPGA est purement logique et ne comporte pas de processeur intégré, alors que le circuit ZYNQ possède un processeur ARM Cortex A9 à double cœur, en plus de la partie logique programmable (Crockett *et al*., 2014).

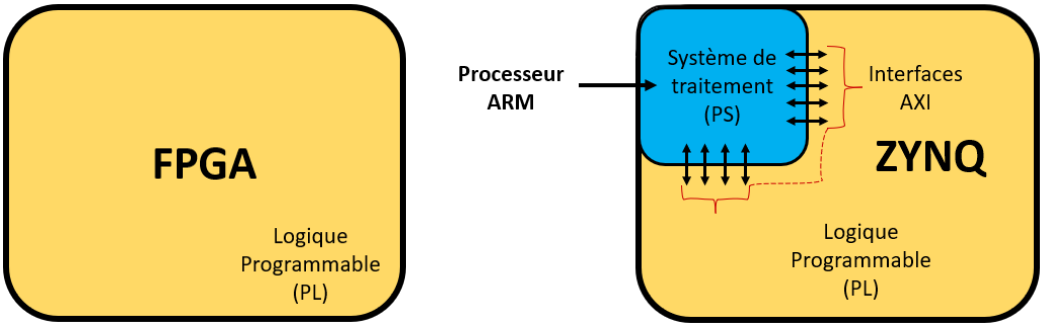

Figure 3.1 : Comparaison de haut niveau entre un circuit FPGA classique et un dispositif ZYNQ avec processeur ARM Cortex A9 (Crockett *et al*., 2014)

3.2 DOMAINES D'UTILISATION DES FPGA

Les circuits FPGA sont des dispositifs très polyvalents qui peuvent être utilisés pour un large éventail d'applications; notamment dans les domaines dans lesquels la flexibilité et la capacité de changement sont critiques (tels que les équipements industriels, les systèmes de contrôle et les appareils médicaux) (Altaf & Mane, 2017; Hameed *et al*., 2020). Il y a aussi les domaines dans lesquels les coûts de développement sont élevés et où la rapidité de mise sur le marché est importante (tels que les appareils électroniques grand public et les consoles de jeux vidéo) (Gabrick *et al*., 2006). Enfin, les domaines dans lesquels la vitesse et les

caractéristiques de performance sont critiquent (tels que les systèmes de télécommunications, les ordinateurs et les périphériques informatiques) (Kee et al., 2017) .

Les circuits FPGA peuvent donc servir dans plusieurs domaines tels que :

- Le prototypage rapide;

- L'émulation du matériel;

- L'accélération de fonctions.

Le concept de prototypage rapide est relativement simple, il s'agit d'implanter sur un circuit FPGA une application pour un test rapide. Le prototypage rapide permet de réduire considérablement les cycles de développement et la durée de réalisation de l'application. L'utilisation d'un circuit FPGA pour le prototypage rapide permet de répondre aux besoins des applications en termes de temps, de coût et de performance (Dombkowski & Kocan, 2004). Ensuite, l'émulation du matériel consiste à utiliser un circuit FPGA pour reproduire un comportement donné d'un équipement. Cela permet de tester des applications sans avoir à utiliser le matériel réel. En outre, elle réduit le temps de développement et le risque d'erreurs. De plus, l'accélération de fonctions est possible grâce à l'utilisation des circuits FPGA. Ceci est particulièrement utile pour les applications qui requièrent une forte puissance de calcul, comme les traitements d'images et de vidéos, les simulations et les jeux vidéo (Bailey, 2011). Enfin, il est clair que les circuits FPGA apportent de nombreux avantages pour les implantations logicielle et matérielle.

3.3 IMPLANTATION DU SYSTÈME AES SUR CIRCUITS FPGA

L'implantation du système de chiffrement/déchiffrement avancé AES (*Advanced Encryption Standard*) sur circuits FPGA est une solution adéquate pour la sécurisation temps-réel des données, car elle permet de réaliser des circuits cryptographiques de haute performance. Les circuits FPGA sont des puces électroniques programmables, autrement dit très flexibles, qui peuvent être facilement adaptés à des besoins spécifiques, ce qui par le fait même les rend très difficiles à pirater. Aussi, cela permet de réaliser des circuits

cryptographiques qui correspondent exactement aux exigences du projet. Les circuits FPGA ont également l'avantage d'être très puissants. Ils peuvent facilement traiter des milliers d'opérations en parallèle, ce qui les rend parfaits pour l'implantation du système de cryptographie AES. Ils ont également l'avantage d'être relativement économiques en termes de puissance consommée, ce qui les rend idéaux pour les applications embarquées.

3.4 PRÉSENTATION DE LA CARTE PYNQ-Z2

Pour notre projet de recherche, nous allons utiliser la carte PYNQ-Z2 de TUL (Figure 3.2), qui est basée sur un circuit ZYNQ XC7Z020-1CLG400C de AMD-Xilinx. La carte PYNQ-Z2 est conçue pour prendre en charge PYNQ, qui est un nouveau cadre source ouvert permettant aux ingénieurs de développer facilement des systèmes embarqués sur les dispositifs ZYNQ en utilisant le langage Python. Il permet le développement de bibliothèque matérielle ou *Overlay* pour l'accélération des calculs dans un environnement *Jupyter Notebook*. Une *Overlay* décrit une conception matérielle sur la partie PL regroupée de telle sorte qu'elle puisse être facilement contrôlée par Python et exécutée sur la partie PS. Cela permet aux développeurs de travailler au niveau Python, en utilisant la bibliothèque matérielle (*Overlay*).

Le système sur puce ZYNQ se compose donc d'une partie logique programmable (PL) basée sur circuit FPGA Artix-7 et d'un système de traitement (PS) basé sur un processeur ARM Cortex-A9 à double cœur (Xilinx Inc., 2018). À partir d'une carte mémoire Micro-SD, la carte de développement PYNQ-Z2 est capable de charger et d'exécuter une image PYNQ (composé du *Ubuntu boot, Python 3.6.5* et *Jupyter Notebook*) sur le processeur ARM Cortex, ce qui permet une interaction facile avec la partie PL via l'interface AXI. Bien que les circuits FPGA disposent d'une puissance de calcul massive, grâce à leur nature hautement parallèle, ils sont généralement difficiles à programmer.

Comme indiqué sur la Figure 3.2, cette carte de développement est dotée de deux connecteurs Pmod, d'un connecteur Arduino, d'un connecteur Raspberry, d'un emplacement pour carte microSD, d'une sortie HDMI, d'un port USB 2.0, d'une sortie audio stéréo, d'une

connectivité Ethernet, d'une entrée/sortie audio (Headphone + MIC), des boutons poussoirs et de LEDs. La carte PYNQ-Z2 est conçue pour faciliter l'implantation des applications temps réel sur circuit ZYNQ.

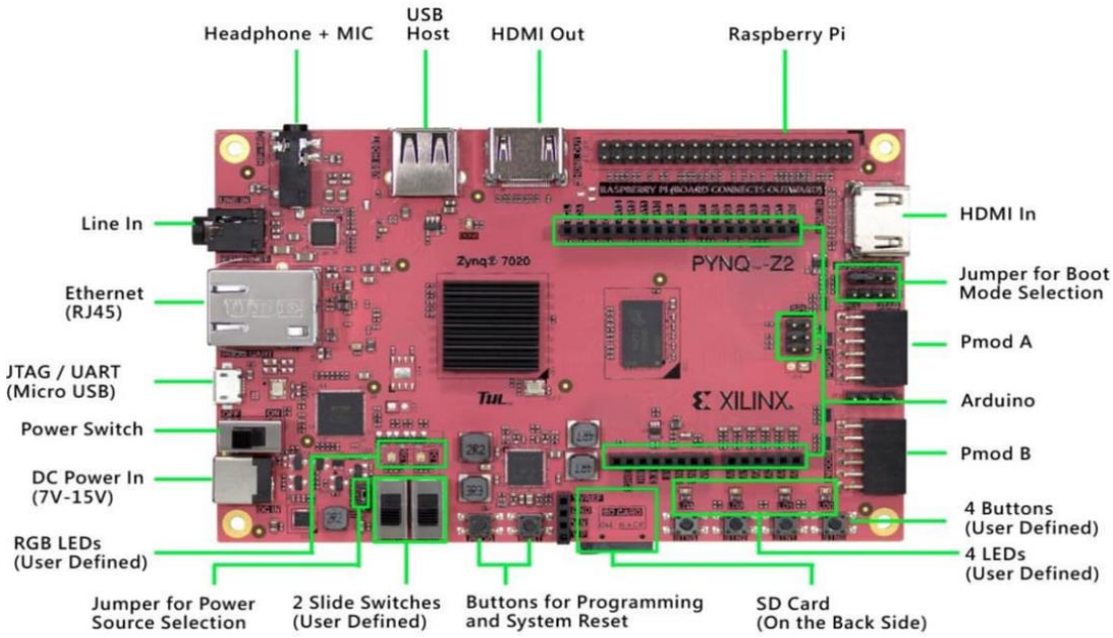

Figure 3.2 : Carte FPGA PYNQ-Z2 (DFRobot, 2022)

3.5 CRÉATION D'UN MODULE IP DU SYSTÈME AES

Vivado HLS est une plateforme logicielle qui transforme une conception développée en langage C/C++ en un niveau de transfert de registre RTL (*Register Transfer Level*), qui peut ensuite être synthétisée et implantée sur la logique programmable (*PL*) d'un dispositif ZYNQ de AMD-Xilinx (Crockett *et al.*, 2014). Comme il est démontré dans la Figure 3.3, le flux de conception Vivado HLS comprend plusieurs étapes depuis la conception en C/C++ jusqu'à la création des sorties pour la synthèse RTL. L'entrée principale du processus HLS (*High Level Synthesis*) est une fonction codée en C/C++, ainsi qu'un banc d'essai (*TestBench*) basé sur le langage C/C++ qui peut être développé pour exécuter la fonction et vérifier son

bon fonctionnement. Ensuite, la vérification fonctionnelle qui est nécessaire pour tester l'intégrité fonctionnelle du code C/C++. L'étape suivante consiste à effectuer le processus de synthèse haut niveau HLS (*High Level Synthesis*) qui implique l'analyse et le traitement de la fonction codée en C/C++, ainsi que des directives et des contraintes définies par l'utilisateur, afin de créer une description RTL (*Register Transfer Level*) du circuit. Une fois que la synthèse HLS a été effectuée et que le modèle RTL équivalent de la fonction C/C++ a été produit, il peut être vérifié par rapport au code C/C++ original via le processus de Co-simulation C/RTL dans la plateforme Vivado HLS. Ensuite, il passe par l'évaluation de la sortie RTL en termes d'implantation et de performances (Crockett *et al*., 2014).

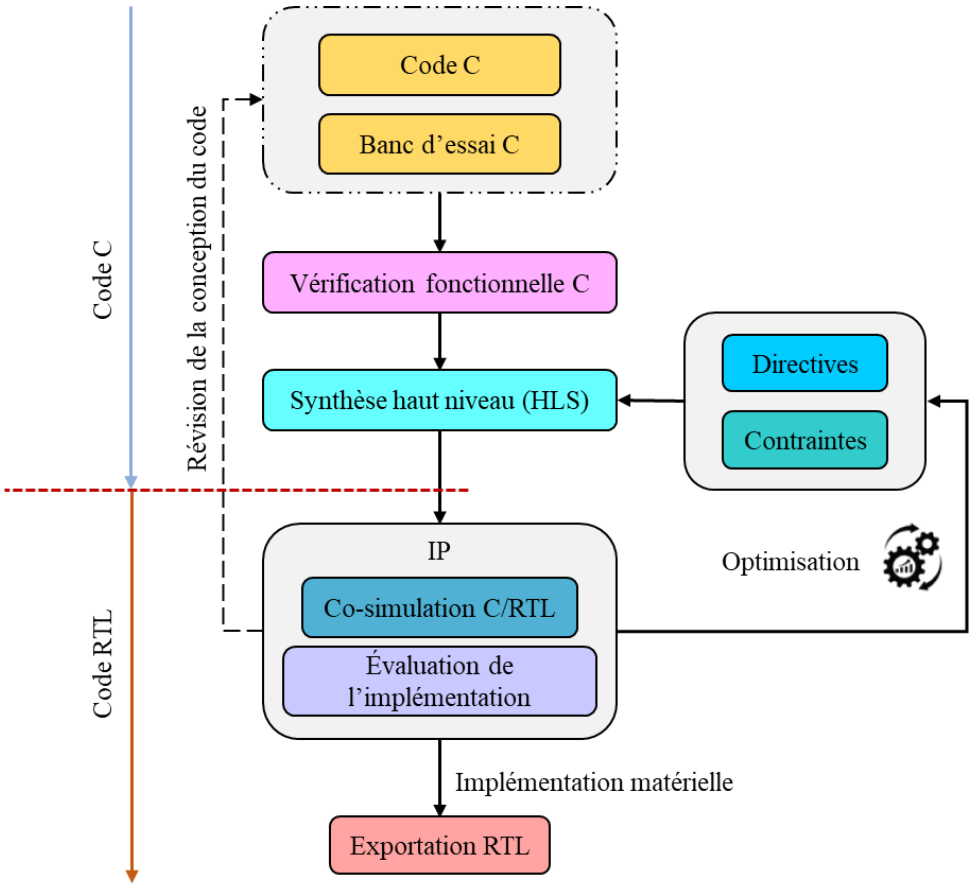

Figure 3.3 : Une vue d'ensemble du flux de conception Vivado HLS (Crockett *et al*., 2014).

Le chemin de retour indiqué sur le côté droit du diagramme représente l'optimisation en utilisant les directives et les contraintes définies par l'utilisateur dans Vivado HLS pour trouver une meilleure solution sans modifier le code source. Si le concepteur est invité à modifier le code C/C++ d'entrée, il doit faire un pas en arrière plus important dans le processus de conception, ce qui est indiqué par la flèche sur le côté gauche du diagramme. Toute modification du code C/C++ nécessite une nouvelle vérification fonctionnelle, avant que les processus ultérieurs de synthèse HLS, de vérification C/RTL et d'évaluation de l'implantation ne soient à nouveau exécutés et itérés si nécessaire (Crockett *et al*., 2014).

Dans une conception au niveau du transfert de registre RTL (*Register transfer level*), les opérations d'entrée et de sortie doivent être effectuées via un port dans l'interface de conception et fonctionnent généralement à l'aide d'un protocole d'entrée/sortie (E/S) spécifique (Xilinx, 2018). Le pragma HLS INTERFACE spécifie comment les ports RTL sont créés à partir de la définition de fonction lors de la synthèse d'interface (Xilinx, 2020). Le pragma HLS INTERFACE est suivi d'autres paramètres, tels que les modes, le nom du port, le registre, la profondeur, le décalage, l'horloge, etc.

Dans la Figure 3.4, on peut voir que l'outil Vivado HLS comporte une fenêtre principale d'édition des fonctions C/C++ ainsi qu'un volet permettant de configurer et de gérer les directives qui influencent le comportement du processus HLS. Le volet directives reflète uniquement la solution "active" et n'est visible que lorsque le code source est ouvert dans la fenêtre principale. Notez que les directives peuvent être intégrées dans le fichier source comme des pragmas. L'outil Vivado HLS produit un rapport de synthèse pour chaque solution, qui représente un ensemble consolidé de statistiques relatives à cette implantation particulière.

La fonction de niveau supérieur (*top level function*) comporte les arguments comme des ports d'entrée et de sortie (E/S), ainsi que les commandes ou directives pragmas HLS INTERFACE. Dans notre projet, les arguments de la fonction sont des ports de données d'entrée et de sortie (E/S) non signées avec une largeur de données respectivement de 384 bits et 128 bits (voir Figure 3.5).

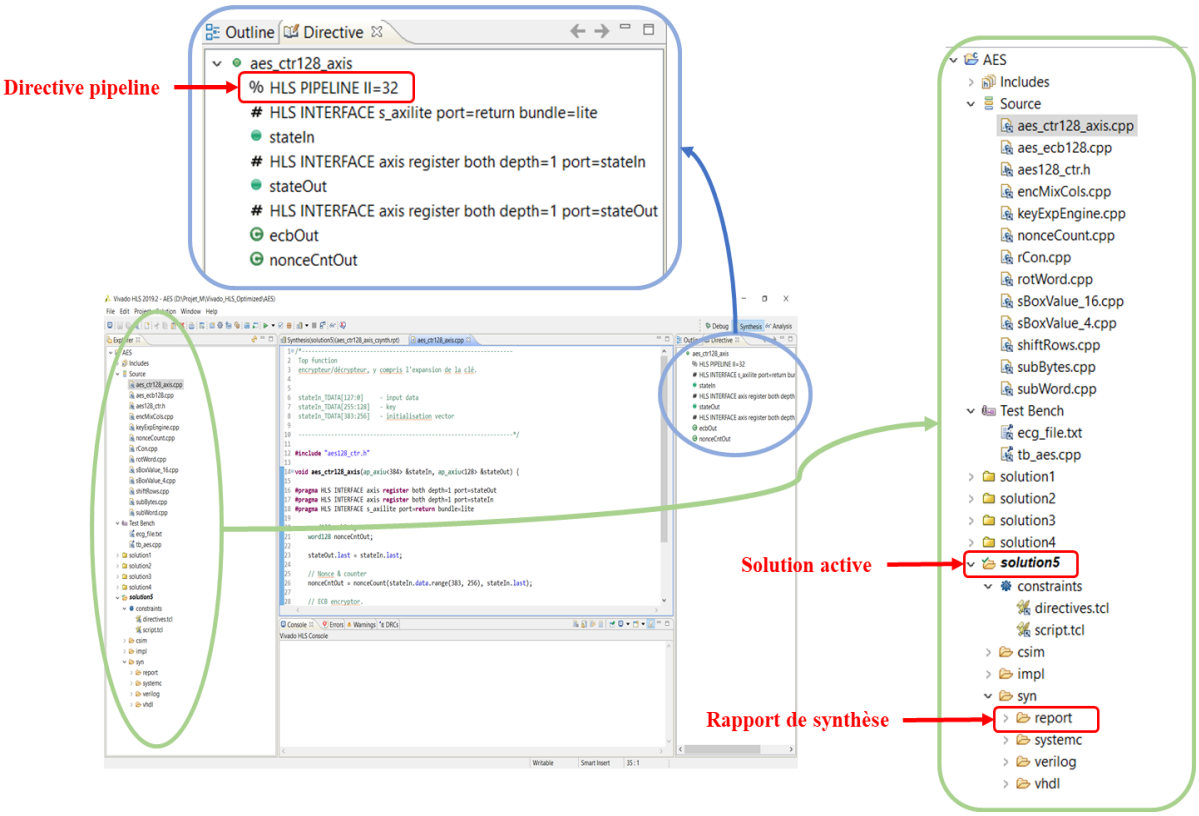

Figure 3.4 : Un aperçu de Vivado HLS

Dans la Figure 3.5, on peut voir les commandes ou directives pragma HLS INTERFACE utilisées dans la fonction de niveau supérieur (*top level function*) du code HLS. Cette commande pragma crée des ports *stateIn* et *stateOut*; Le paramètre *axis* implante les ports comme une interface AXI4-Stream. Le paramètre *register* fait persister les signaux au moins jusqu'au dernier cycle d'exécution de la fonction. L'option *both* est utilisée avec *register* pour le placement des registres sur les chemins (TDATA, TVALID, et TREADY). Le paramètre *depth* spécifie le nombre d'échantillons à traiter par le système de chiffrement. Le paramètre *port* spécifie le nom du port. Pour la 7ieme ligne, une autre directive pragma INTERFACE est utilisée dans la même fonction haut-niveau. Cette directive est employée pour créer une interface AXI-Lite avec le nom "s_axilite" et le port "return" créera la broche d'interruption pour le niveau du transfert de registre RTL (*Register transfer level*). Toutes les interfaces AXI4-Lite "s_axilite" sont regroupées dans un seul port AXI4-Lite.

*aes_ctr128_axis.cpp

```
#include "aes128_ctr.h"

void aes_ctr128_axis(ap_axiu<384> &stateIn, ap_axiu<128> &stateOut) {

#pragma HLS INTERFACE axis register both depth=1 port=stateOut
#pragma HLS INTERFACE axis register both depth=1 port=stateIn
#pragma HLS INTERFACE s_axilite port=return bundle=lite

    word128 ecbOut;
    word128 nonceCntOut;

    stateOut.last = stateIn.last;

    // Nonce & compteur
    nonceCntOut = nonceCount(stateIn.data.range(383, 256), stateIn.last);

    // Chiffreur ECB
    ecbOut = aes_ecb128(stateIn.data.range(255, 128), nonceCntOut);

    // XOR ECB output avec plaintext pour le texte chiffré ou le texte déchiffré
    stateOut.data = stateIn.data.range(127, 0) ^ ecbOut;

}
```

Figure 3.5 : Un exemple de pragmas insérés dans le code source C/C++ pour la synthèse d'interface

Une fois la conception sous Vivado HLS validée, elle sera convertie en paquet IP (*Intellectual Property*) pour être intégrée dans un système plus grand. Ceci peut être réalisé directement à l'aide des fichiers RTL créés automatiquement par le processus HLS (c'est-à-dire le code VHDL ou Verilog), afin de les exporter vers un autre environnement comme l'intégrateur de blocs de l'outil Vivado design (Crockett *et al.*, 2014). Cette dernière étape de l'outil Vivado HLS consiste donc à exporter la conception RTL sous une forme pouvant être utilisée par d'autres outils de conception. Dans le menu principal, on clique sur *Solution*, ensuite sur *Export RTL*. La boite de dialogue *Export RTL as IP* s'ouvre comme indiqué dans la Figure 3.6. Cela rendra le code RTL généré utilisable dans Vivado Design en tant que paquet IP matériel personnalisé.

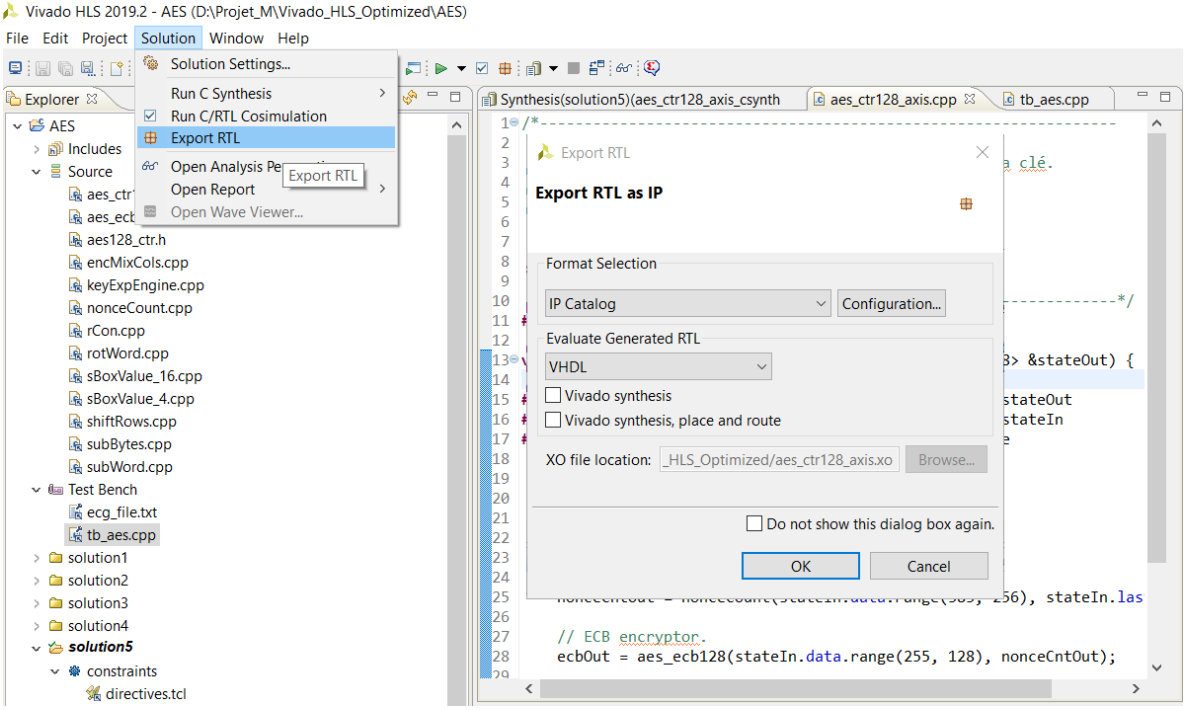

Figure 3.6 : Exportation du design RTL (*Register transfer level*) en paquet IP

Dans notre projet, nous nous sommes basés sur l'algorithme de chiffrement AES-CTR 128 bits, implanté dans l'environnement Vivado HLS par Harvey (2016). Cette implantation permet de générer le bloc AES IP qui sera intégré par la suite dans un projet plus complexe sous l'environnement Vivado Design afin de créer notre architecture matérielle.

Le bloc AES IP permet d'effectuer les deux opérations de chiffrement et de déchiffrement, dépendamment du besoin. Pour ce faire, le bloc AES fonctionne en mode compteur (CTR) et utilise une clé de 128 bits avec l'expansion de la clé. Comme indiqué à la Figure 3.7, il possède des interfaces d'entrée et de sortie AXI-Stream pour les données d'entrée/sortie. Il a une période d'horloge pour une simulation fixée à 10 ns et il est synchronisé à partir d'une seule ligne d'horloge (entrée ap-clk). Il est aussi réinitialisé par l'entrée ap-rst-n active à l'état bas et synchronisé en interne avec ap-clk (Figure 3.7).

Les données d'entrée ont une largeur de 384 bits qui sont divisés en trois champs de 128 bits chacun, comme indiqué dans la Figure 3.8. Les bits 127:0 sont le bloc de données (Message original), les bits 255:128 représentent la clé de 128 bits et les bits 383:256 représentent le vecteur d'initialisation. Cependant, les données de sorties ont une largeur de 128 bits.

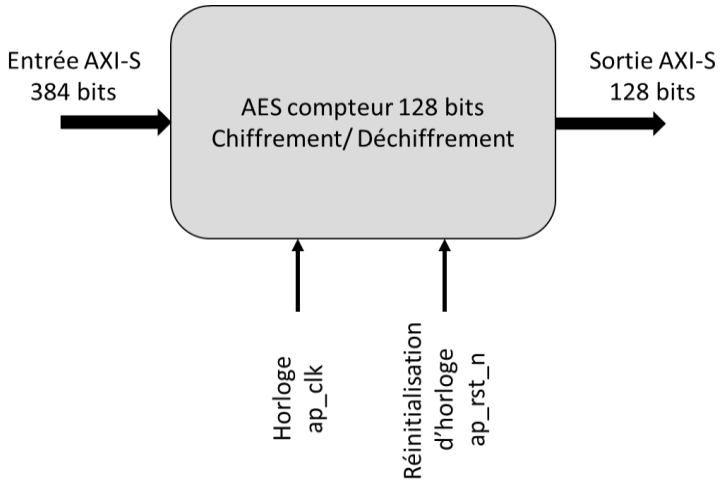

Figure 3.7 : Schéma représentatif du paquet IP de chiffrement/déchiffrement AES-128 CTR (Harvey, 2016)

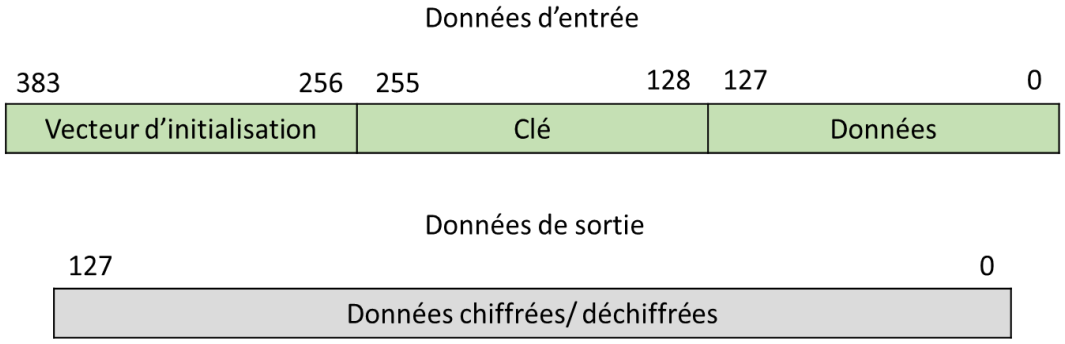

Figure 3.8 : Représentation de format de données d'entrée/sortie (Harvey, 2016)

Nous avons créé un fichier banc d'essai (*Test Bench*) pour tester les opérations de chiffrement/déchiffrement de l'algorithme AES dans la plateforme Vivado HLS (Figure 3.9).

Nous avons commencé par définir la clé secrète et le vecteur d'initialisation, puis nous avons procédé à la lecture des données d'entrée d'un signal ECG (ElectroCardioGram) format texte (*.txt) de vingt échantillons, ensuite nous avons procédé à l'opération de chiffrement qui nous produit les données de sortie de message chiffré. Pour l'opération de déchiffrement, nous avons utilisé la même clé et le même vecteur d'initialisation pour récupérer les données en clair à partir de message chiffré. Les résultats sont prouvés dans la Figure 3.10.

*tb_aes.cpp

```
#include "../src/aes128_ctr.h"

int main() {

    ap_uint<1> tbLast = 0;
    ap_axiu<384> tbStateIn;
    ap_axiu<128> tbStateOut;

    string plainTextFile = "ecg_file.txt";
    string cipherTextFile = "text_chiffré.txt";
    string dummy1;
    string dummy2;
    string readHexData;

    ifstream inData;
    ofstream outData;

    static word128 ecgData;
    stringstream hexSString, ss;
    static string plainTextHexString;

    int lineCharCount[100];
    int plainTextLineCount;

    static word128 plainTextArray[20];
    static word128 cipherTextArray[20];
    static word128 cipherText;

    /* Initialization Vector*/
    static word128 vectorIn = ap_uint<128>("f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff",
            16);
    /* key */
    static word128 key = ap_uint<128>("2b7e151628aed2a6abf7158809cf4f3c", 16);
```

Figure 3.9 : Une partie du code de banc d'essai C/C++

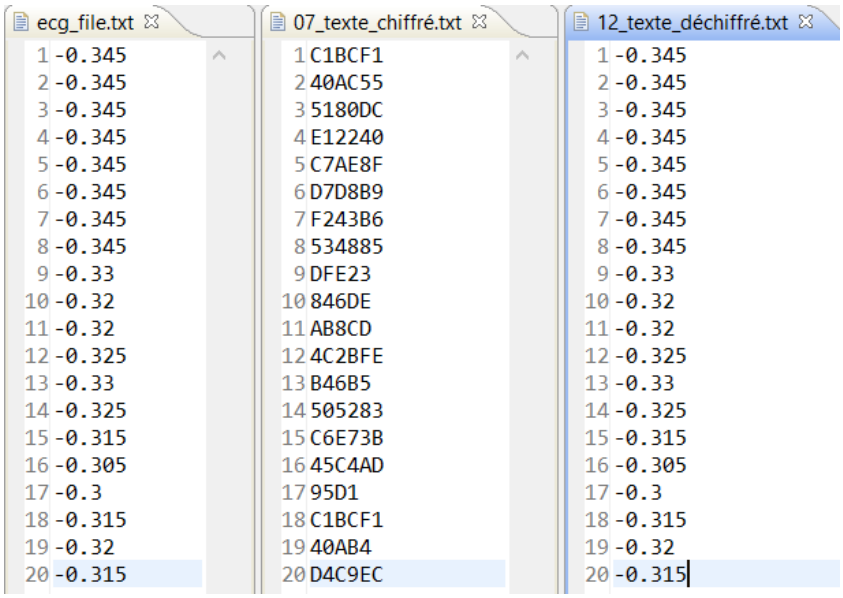

Figure 3.10 : Comparaison entre les trois fichiers de données original, chiffré et déchiffré respectivement de gauche à droite

3.6 ARCHITECTURE DU BLOC IP AES ET SES CARACTÉRISTIQUES

Le bloc IP AES 128 bits se compose d'un chiffreur en mode compteur (CTR) qui prend sa sortie à partir d'une clé secrète et d'un vecteur d'initialisation IV (*Initialisation Vector*). À partir de la clé secrète principale, il se génère 10 sous clés de chiffrement différentes afin d'en avoir une pour chaque tour. Le vecteur d'initialisation est composé de 128 bits ayant une partie fixe de 96 bits qui représente le nonce (numéro qui ne peut être utilisé qu'une seule fois) et une deuxième partie de 32 bits qui représentent le compteur. Au début de chaque nouveau bloc de message, le nonce sera rechargé depuis le vecteur d'initialisation. Le compteur sera incrémenté de 1 à chaque bloc, il est également réinitialisé au début de chaque nouveau bloc. Ces deux parties seront combinées pour former le vecteur d'initialisation du prochain bloc. Pour générer le chiffrement du premier bloc, la sortie du chiffreur est suivie par une opération XOR avec le bloc de message clair. Pour les blocs suivants, on applique la même opération jusqu'à la fin de message. L'opération XOR est une fonction réversible. En effet, lorsqu'on applique la même opération à un bloc chiffré et la clé utilisée pour le chiffrer,

on obtient le message dans son état initial. Cela permet donc de déchiffrer les messages chiffrés avec ce système.

3.6.1 Co-Simulation C/RTL

Le Tableau 3.1 résume le temps de traitement de la durée spécifiée par la solution. Dans la synthèse, la période d'horloge est fixée à 10 ns. Le programme Vivado HLS cible une période d'horloge de 10 ns moins l'incertitude de l'horloge pour avoir l'horloge estimée. Dans notre situation, cela est démontré comme ceci :

$$10 - 1.25 = 8.75\ ns \tag{3.1}$$

Tableau 3.1 : Estimation de période d'horloge

Horloge	Cible	Estimé	Incertitude
ap_clk	10.00 ns	8.724 ns	1.25 ns

La période d'horloge estimée est de 8,724 ns. Cependant, l'incertitude de l'horloge quant à elle garantit une certaine marge de temps disponible pour les retards inconnus dus au placement et au cheminement. Le Tableau 3.2 résume l'estimation des ressources matérielles utilisées. La conception utilise 2218 Flips-Flops et 6933 LUTs correspondant respectivement à 2% et 13% des ressources disponibles sur le circuit ZYNQ de la carte PYNQ-Z2. En revanche, il n'y a aucune utilisation de la mémoire ni de blocs DSP (Digital Signal Processing).

Tableau 3.2 : Estimation de ressources utilisées par la conception

Nom	BRAM	DSP	FF	LUT	URAM
Total	0	0	2218	6933	0
Utilisation (%)	0	0	2	13	0

3.6.2 Optimisation de l'algorithme AES

Les directives HLS peuvent être utilisées pour optimiser et configurer la latence, les performances et d'autres paramètres liés à la synthèse HLS de la conception. En particulier, le pipeline peut être utilisé avec différents intervalles d'initialisation pour obtenir des performances assez élevées sans modifier le code source. La directive "pipeline" est utilisée pour réduire l'intervalle d'initialisation d'une fonction ou d'une boucle en permettant une exécution concurrentielle des opérations. Une fonction ou une boucle en pipeline peut traiter de nouvelles entrées à chaque N cycles d'horloge, où N est l'intervalle d'initialisation de la boucle ou de la fonction. Le Tableau 3.3 présente les résultats de performance de la configuration pipeline de chaque cycle d'intervalle avec une période d'horloge de 10 ns sur la carte PYNQ-Z2.

Tableau 3.3 : Résultats des ressources utilisées pour différentes configurations

Configuration pipeline (N)	LUTs	FF	BRAM_18K	Période d'horloge estimée
2	11097	5519	11	9.806 ns
4	6172	3296	11	9.989 ns
16	2531	2314	0	10.419 ns
32	2973	2203	0	9.432 ns

Le Tableau 3.3 présente les ressources utilisées pour chaque configuration pipeline avec une période d'horloge cible de 10 ns sur la carte PYNQ-Z2. Comme on peut le voir, en utilisant un cycle d'intervalle plus élevé, on obtient des résultats meilleurs que lorsqu'on utilise un cycle d'intervalle plus faible. En utilisant une configuration pipeline à 16 et 32 cycles, on ne fait aucune utilisation de la mémoire BRAM (Block Random Access Memory). Cependant, au niveau du pipeline 16, il y a une violation du temps pour la période d'horloge estimé (10.419 ns). L'utilisation de la configuration pipeline 32 nous permet d'employer moins de ressources système et obtenir des performances plus élevées. Nous pouvons

conclure que la directive "pipeline" est très bénéfique pour améliorer les performances du système.

3.7 CONCEPTION DE L'ARCHITECTURE MATÉRIELLE VIVADO DESIGN

À l'aide de l'outil Vivado HLS, nous avons exporté le RTL de notre conception en tant que bloc IP. Une fois ce dernier est généré, il peut être intégré à la plateforme Vivado Design Suite et utilisé comme un bloc IP standard dans la conception d'un système plus vaste en tant que composant matériel. L'importation de ce bloc IP dans Vivado Design est représentée par le symbole de la Figure 3.11.

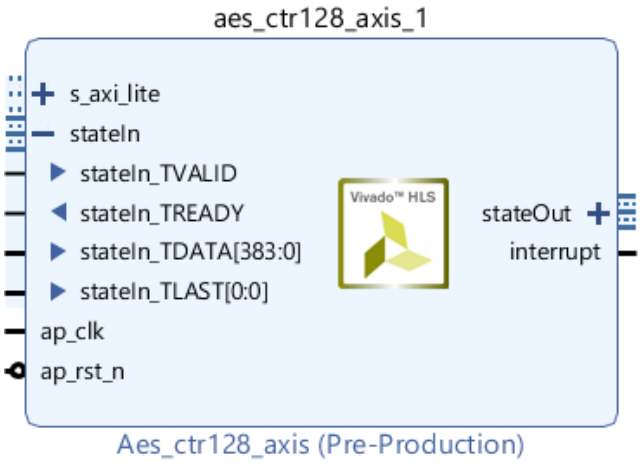

Figure 3.11 : Symbole du bloc IP AES dans Vivado Design

Lorsque nous lançons Vivado design, la page de démarrage nous permet de créer un nouveau projet, ensuite nous avons sélectionné la carte PYNQ-Z2, comme illustrée dans la Figure 3.12. Une fois le projet est créé, nous avons ajouté le bloc IP ZYNQ7, le système de traitement ZYNQ7, qui nous permet d'implanter notre système sur le processeur ZYNQ, ainsi que le bloc IP AES pour les opérations de chiffrement/déchiffrement comme illustré dans la Figure 3.13.

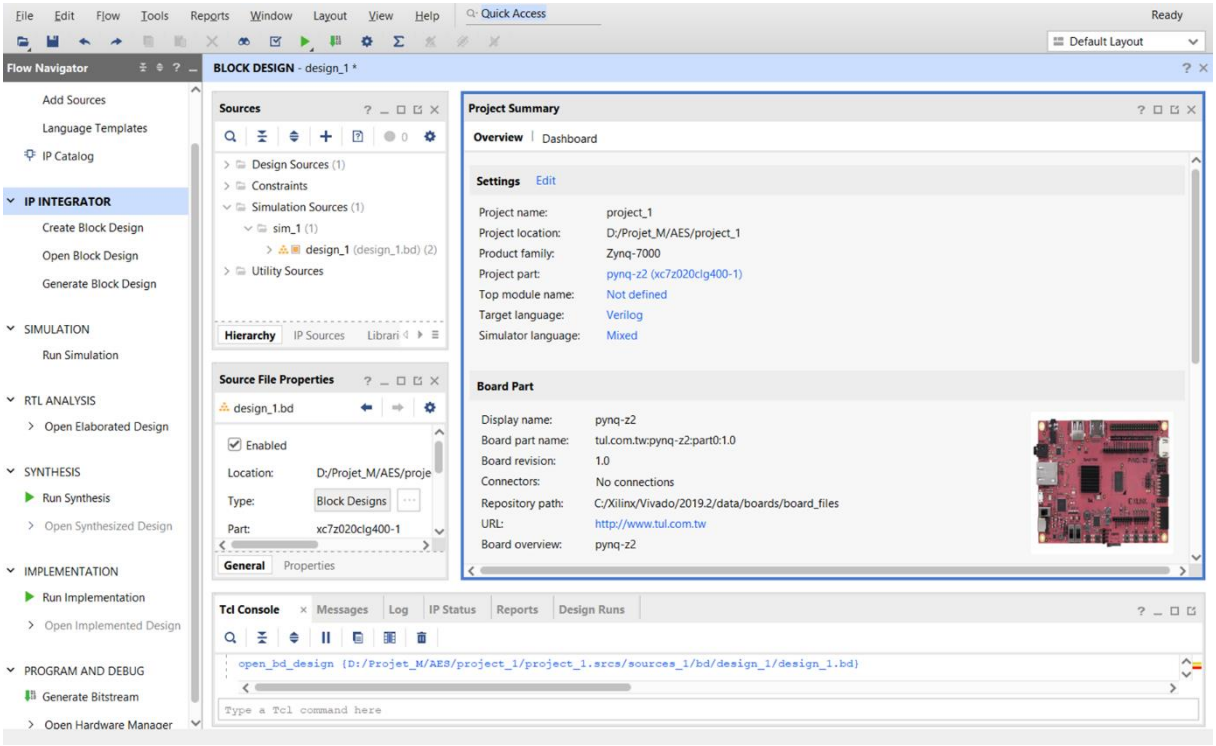

Figure 3.12 : Page de démarrage d'un nouveau projet sous Vivado Design.

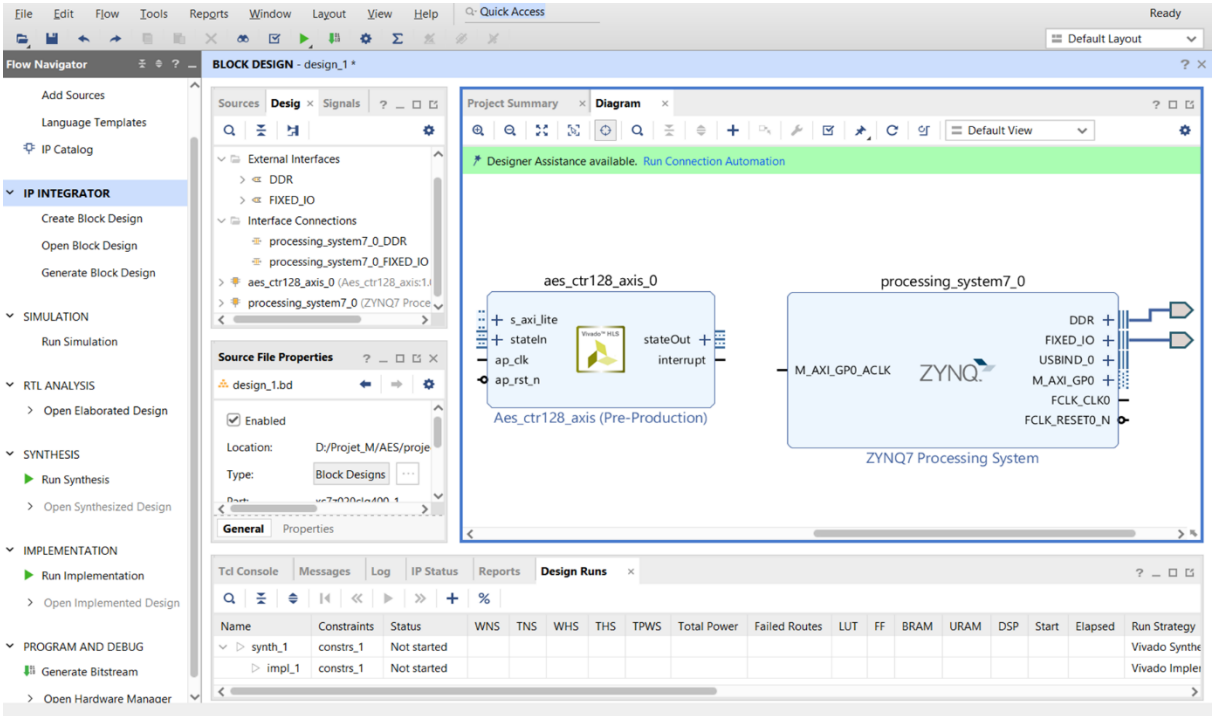

Figure 3.13 : Création d'un nouveau bloc design et ajout de 2 IPs.

Comme indiqué dans la Figure 3.11 et Figure 3.13, le bloc IP AES contient plusieurs ports d'entrée et de sortie lui permettant d'échanger avec les autres blocs IP du système qui seront ajoutés. La description de chaque port du bloc IP AES a été résumée dans le Tableau 3.4.

Tableau 3.4 : Les ports du bloc AES IP (Harvey, 2016)

Nom du port	**Direction**	**Description**
s_axi_lite	In/Out	Interface esclave AXI4-Lite pour la configuration des registres du noyau IP.
stateIn_TVALID	In	Signal d'établissement de liaison AXI4-Stream pour indiquer que stateIn_TVALID et stateIn_TLAST sont valides.
stateIn_TREADY	Out	Signal d'établissement de liaison AXI4-Stream pour indiquer que ce bloc IP est prêt à accepter de nouvelles données d'entrée.
stateIn_TDATA[383:0]	In	384 bits de données d'entrée AXI4-Stream incluant les données d'entrée, la clé et le vecteur d'initialisation, échantillonnés sur le front montant de ap_clk lorsque stateIn_TVALID et stateIn_TREADY sont tous les deux actifs.
stateIn_TLAST	In	Signal d'établissement de liaison du flux AXI4 pour indiquer la fin de la trame (EOF).
ap_clk	In	Cette horloge (ap-clk) permet la synchronisation de toutes les opérations et les interfaces.
ap_rst_n	In	Reset asynchrone active à l'état bas, synchronisé en interne avec ap_clk.
stateOut	Out	Les données de sortie AXI4-Stream qui transportent des données cryptées/décryptées de 128 bits.
interrupt	Out	Broche de sortie d'interruption du bloc IP à utiliser lorsqu'il est configuré en mode interruption. Cette broche indique l'état d'interruption du bloc IP.

L’intégrateur de blocs IP est une interface graphique dans Vivado Design qui permet de créer un bloc design, ainsi que de concevoir et développer des systèmes à base de circuit FPGA, en utilisant des blocs prêts à l'emploi et personnalisables appelés IP. De plus, il permet d’établir des liens avec les outils logiciels qui assurent le design des fonctionnalités associées au microprocesseur. Lorsque nous créons un projet dans l’intégrateur de blocs, nous pouvons ajouter des blocs IP à notre design pour intégrer les fonctionnalités dont nous avons besoin. Vivado Design Suite offre une grande variété de IP logiciels et matériels prêts à l'emploi pour choisir le bon type de traitement numérique dont nous avons besoin.

L’architecture proposée de système de chiffrement AES-128 bit, basée sur le FPGA ZYNQ-XC7Z020 SoC, est présentée à la Figure 3.14. Le matériel est conçu à l'aide de Vivado Design Suite, la conception du matériel consiste en une panoplie de blocs IP configurés et connectés les uns aux autres pour répondre aux exigences requises. Il est nécessaire d’implanter cet assemblage de blocs IP créé, afin de générer et exporter le fichier de configuration bitstream qui sera téléversé sur la carte FPGA pour réaliser une implantation matérielle de système de chiffrement AES-128 bits.

Le diagramme complet est présenté à la Figure 3.14. Cependant, les Figures 3.15-3.18 présentent les agrandissements de différentes parties de ce diagramme. Ce dernier est principalement basé sur le système de traitement ZYNQ7 qui représente le processeur ARM CORTEX A9 de la partie PS sur laquelle sera exécuté le code C/C++ ou Python, et le bloc AES qui sera implanté sur la partie PL du circuit ZYNQ. Les détails des différents composants sont donnés dans les sous-sections suivantes.

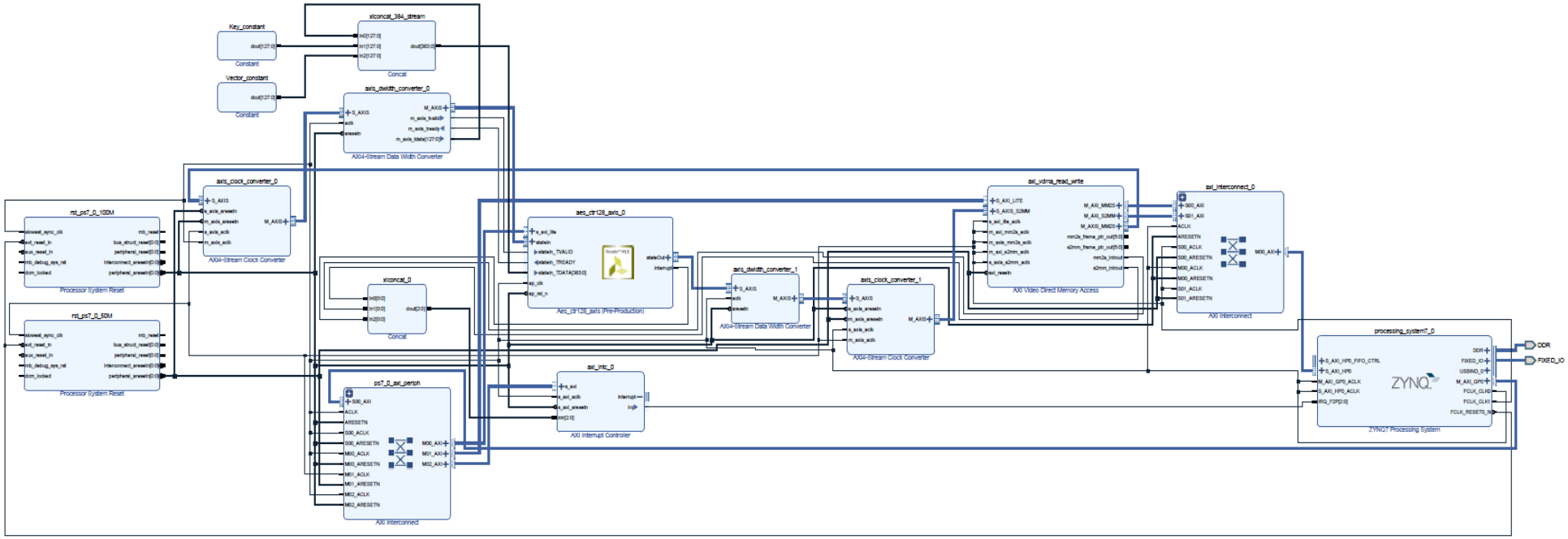

Figure 3.14 : Diagramme bloc de l'implantation proposée dans Vivado Design

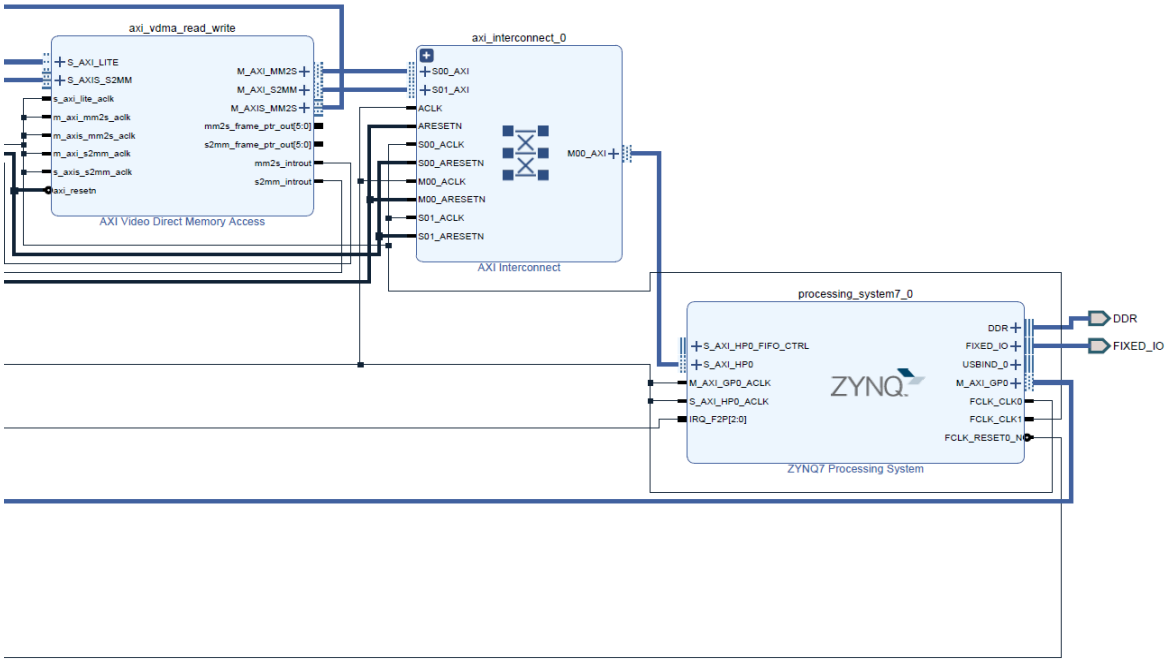

Figure 3.15 : Zoom 1 du diagramme bloc d'implantation

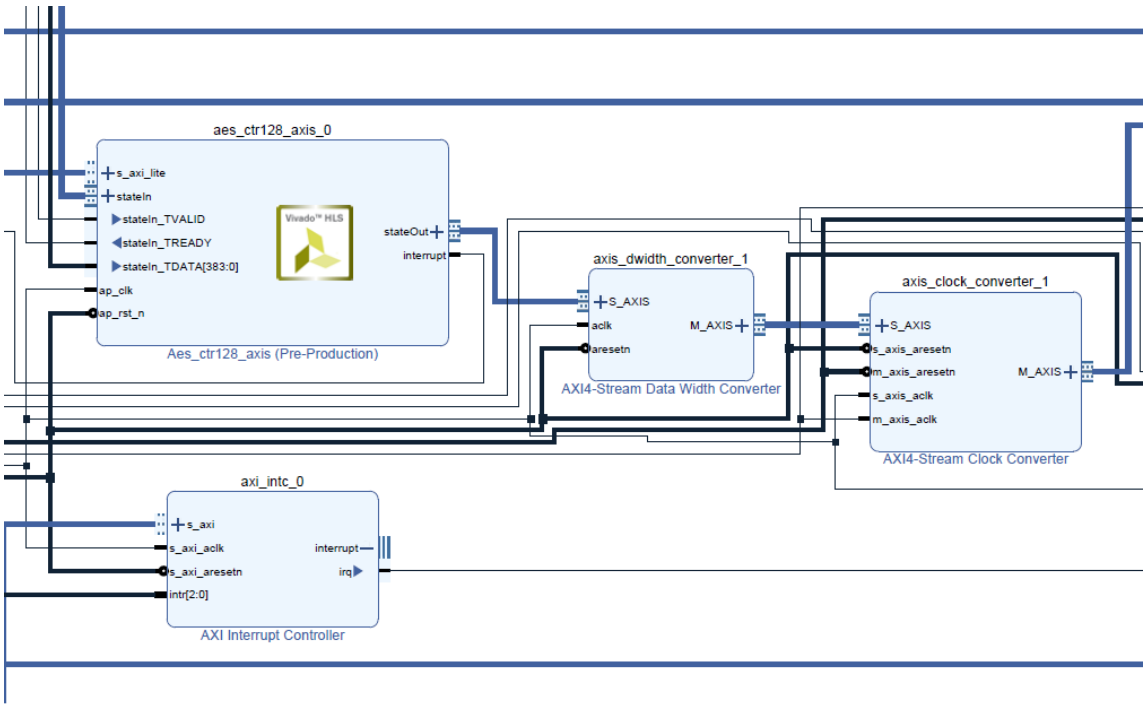

Figure 3.16 : Zoom 2 du diagramme bloc d'implantation

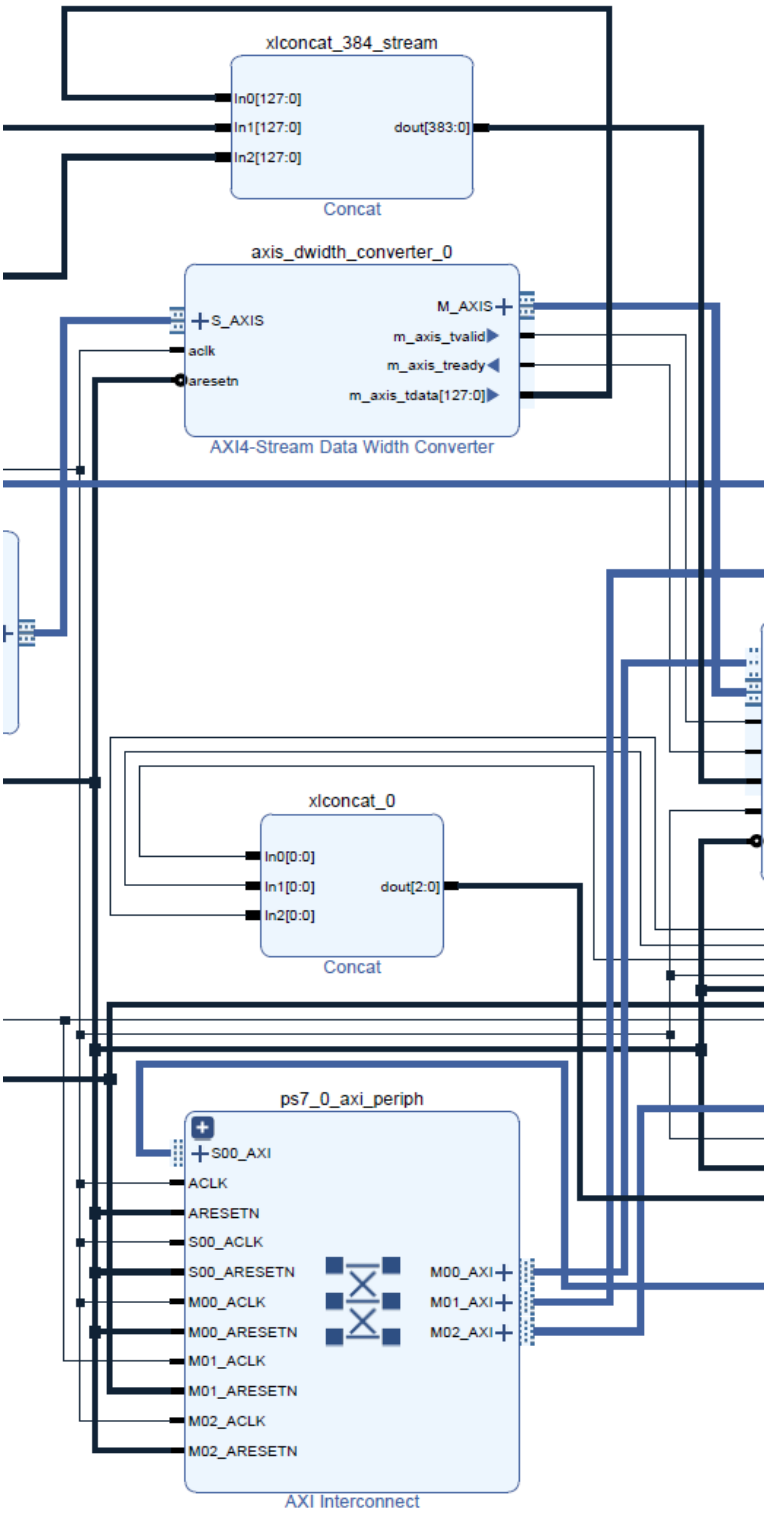

Figure 3.17 : Zoom 3 du diagramme bloc d'implantation

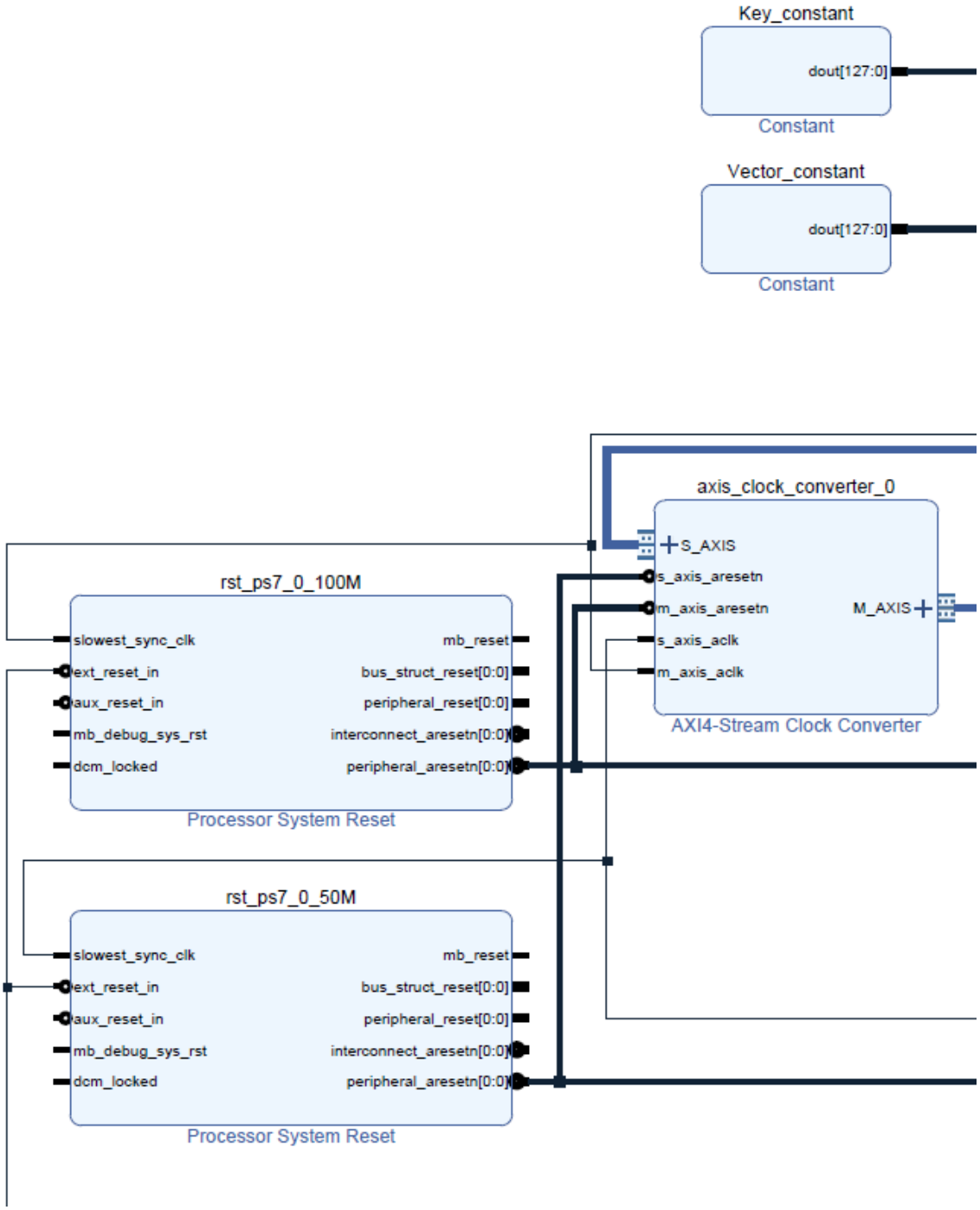

Figure 3.18 : Zoom 4 du diagramme bloc d'implantation

3.8 DESCRIPTION DES DIFFÉRENTS BLOCS DE L'ARCHITECTURE MATÉRIELLE

Notre architecture se compose d'un bloc IP AES avec d'autres blocs IPs prédéfinis qui sont reliés entre eux par des bus AXI. En effet, ils représentent des blocs matériels qui seront implantés sur le circuit ZYNQ de la carte PYNQ-Z2. Les blocs matériels sont des modules HDL (Hardware Description Language), prêts à l'emploi, qui simplifient grandement la configuration de circuits FPGA. Ces blocs matériels sont développés par Xilinx et représentent un certain nombre de fonctionnalités qui seront utilisées par notre circuit. Pour notre projet, les blocs matériels utilisés seront décrits par la suite. Il est à noter que les blocs IP peuvent également être créés par l'utilisateur à partir du langage HDL dans Vivado Design ou à partir du langage C/C++ dans Vivado HLS.

3.8.1 Système de traitement ZYNQ7

Le système de traitement ZYNQ7 fait référence à la partie PS du circuit ZYNQ-XC7Z020 de notre carte PYNQ-Z2. Il doit être ajouté au diagramme bloc qui a besoin du processeur ARM CORTEX A9 pour réaliser des tâches de traitement logiciel. Il permet également d'implanter et de configurer l'interface entre le processeur Cortex-A9 double cœur et la partie FPGA programmable. Le bloc ZYNQ7 est présenté à la Figure 3.19, il est personnalisé pour accéder à la mémoire DDR (Double Data Rate) et aux sources d'horloge 100 MHz et 50 MHz. De plus, un port esclave haute performance HP (*High Performance*)

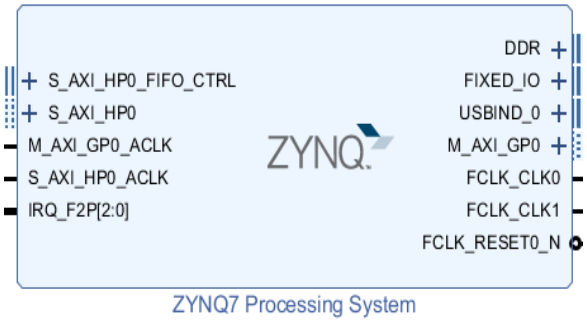

Figure 3.19 : Système de traitement ZYNQ7

est activé. Grâce à ce port, les blocs IP seront capables d'effectuer des opérations de lecture/écriture de la mémoire.

3.8.2 Bloc d’accès direct à la mémoire vidéo VDMA

Le bloc AXI VDMA (*Video Direct Memory Acces*), présenté dans la Figure 3.20, fournit un accès direct à la mémoire à large bande passante entre la mémoire et les périphériques cibles vidéo de type AXI4-Stream. De plus, il lit les données pixel par pixel de la mémoire et les convertit en flux AXI4-Stream. Dans notre conception, cet IP est utilisé pour la lecture et l’écriture de mémoire tampon. Il permet de transmettre les données au bloc AES sous forme de flux AXI4-Stream par le biais de convertisseur d’horloge et de convertisseur de largeur de données.

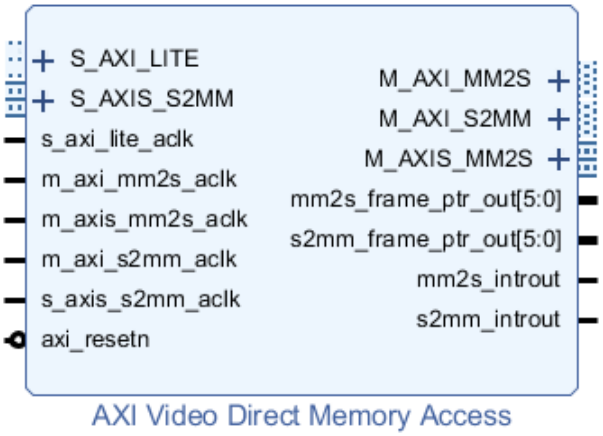

Figure 3.20 : Bloc d’accès direct à la mémoire vidéo (VDMA)

3.8.3 Bloc convertisseur de flux d'horloge

Le bloc convertisseur de flux d’horloge (*AXI4-Stream Clock Converter*) illustré à la Figure 3.21, a pour fonction principale la conversion des signaux d’horloge entre l'interface maître et l'interface esclave (une conversion ascendante ou descendante). Dans notre projet, nous avons utilisé deux IP Clock Converter aux sources d'horloge 100 MHz et 50 MHz. Pour atteindre une meilleure performance de notre système, le bloc AES IP doit faire des

opérations plus intensives, ce qui fait en sorte qu'il utilise les sources d'horloge de 100 MHz et que le bloc IP VDMA fonctionne avec l'horloge de 50 MHz.

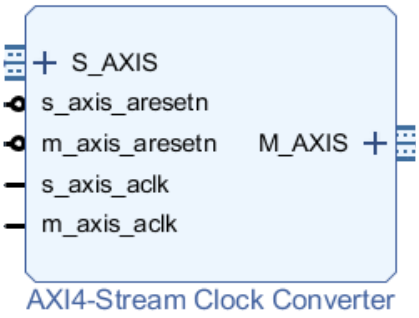

Figure 3.21: Convertisseur d'horloge

3.8.4 Bloc convertisseur de largeur de données

Le bloc convertisseur de largeur de données (*AXI4-Stream Data Width Converter*), illustré à la Figure 3.22, a pour principale fonction l'augmentation ou la diminution de la largeur des données. Les interfaces maîtres et esclaves ont des largeurs de données de flux différentes. Le bloc IP VDMA (*Video Direct Memory Access*) ne fournit que 32 bits, alors que le bloc AES nécessite des données d'entrée de 128 bits. Dans ce cas, la largeur de données doit être augmentée. De la même manière, le bloc IP d'écriture VDMA nécessite des données de 32 bits pour écrire dans la mémoire, mais le bloc AES fournit des données cryptées de 128 bits. Dans ce cas, la largeur des données doit être réduite.

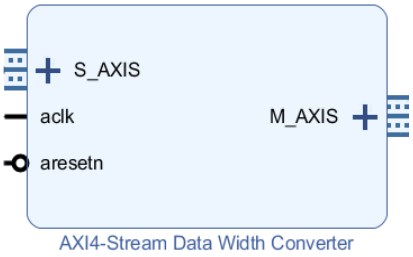

Figure 3.22 : Convertisseur de largeur de données

3.8.5 Bloc de chiffrement/déchiffrement AES

Le bloc de chiffrement/déchiffrement AES est généré à l'aide de Vivado HLS avant d'être importé dans Vivado Design (Figure 3.23). L'opération principale de ce bloc est de crypter/décrypter les données afin d'améliorer leur sécurité. Ce module IP permet d'exécuter l'algorithme AES par segment de 128 bits (16 octets). D'ailleurs, cet IP prend 384 bits de données en entrée, dont les premiers 128 bits sont pour les données, les 128 bits suivants sont pour la clé et les derniers 128 bits sont pour le vecteur d'initialisation. Si les données chiffrées sont fournies en entrée, alors le bloc IP AES produira des données déchiffrées et vice versa.

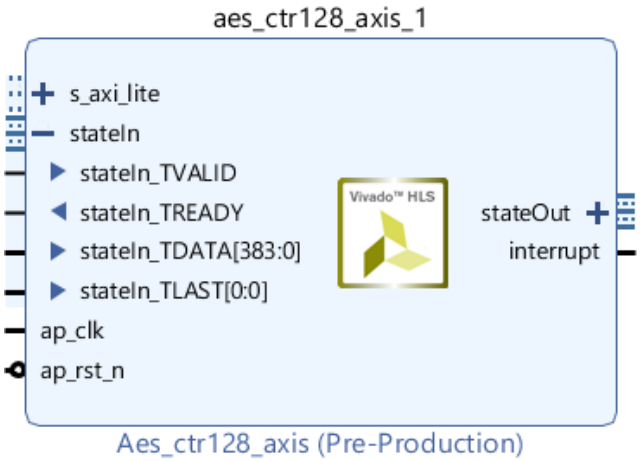

Figure 3.23 : Bloc IP AES

3.8.6 Autres IPs utilisés

Le Tableau 3.5 présente la description des autres blocs IP utilisés dans notre architecture implantée sous Vivado Design. Il s'agit des modules IP suivants : Concate, Processor System Reset, AXI_Interconnect, AXI_Interrupt Controller et Constant.

Tableau 3.5 : Bloc IP de Vivado utilisé dans notre architecture

Blocs IP	Description des blocs
Concate	Concaténation des données d'entrées, la clé et le vecteur d'initialisation en un seul bus de données de 384 bits et aussi pour fusionner le signal d'interruption en un seul signal combiné.
Processor System Reset	Réinitialisation synchrone pour les blocs IP qui sont connectés à une horloge spécifique. Deux de ces IP sont utilisés pour la réinitialisation synchrone de 50 MHz et 100 MHz.
AXI InterConnect	Connexion d'un ou plusieurs dispositifs maîtres AXI à mémoire avec un ou plusieurs dispositifs esclaves à mémoire pour créer un design complet fonctionnel.
AXI Interrupt Controller	Réception de plusieurs entrées d'interruption en provenance de périphériques et les fusionner en une sortie d'interruption vers le processeur système.
Constant	Ce bloc fournit une valeur constante. Il est utilisé deux fois pour piloter la clé qui est de largeur de 128 bits et de même pour le vecteur d'initialisation qui est de largeur de 128 bits.

3.9 DESCRIPTION COMPORTEMENTALE DU CIRCUIT

Le circuit ZYNQ utilisé pour l'implantation de notre projet est composé de deux parties; une partie système de traitement (PS) et une partie logique programmable (PL). Le système de traitement ce compose d'un microprocesseur CORTEX-A9 double cœur et d'un contrôleur pour la mémoire DDR (Double Data Rate). Cependant, l'architecture AES proposée sera implantée sur la partie logique programmable (PL). Le schéma fonctionnel sur la

Figure 3.24 illustre la conception que nous avons créée.

Le processeur ARM CORTEX-A9 et le contrôleur de mémoire DDR sont contenus dans la partie PS du circuit ZYNQ. La liaison entre l'IP VDMA et le contrôleur de mémoire

DDR se fait par le biais de l'IP AXI-Interconnect, les AXI_MM2S et AXI_S2MM sont des bus AXI4 mappés en mémoire et fournissent l'accès VDMA à la mémoire DDR (Double Data Rate). Le bus AXI-lite permet au processeur de communiquer avec le bloc AXI VDMA pour configurer, initialiser et surveiller les transferts de données.

Par la suite, le bloc IP VDMA lit les données pixel par pixel de la mémoire et les convertit en flux AXI4-stream afin de les transmettre au bloc AES, par le biais d'un convertisseur de largeur de données et d'horloge. Dans le cas où l'on désire obtenir la meilleure performance du système, le bloc VDMA nécessite 32 bits et une horloge de 50 MHz, tandis que le bloc AES en nécessite 128 bits et une horloge de 100 MHz.

À la suite de cela, le bloc Concate effectue une concaténation des données avec la clé et le vecteur d'initialisation. Le bloc AES sera en mesure d'accéder à ces données pour la suite du processus en communiquant avec le processeur ARM CORTEX-A9 par le biais de AXI-Lite.

Le bloc AES produit alors la forme chiffrée des données claires. Les données résultantes sont retransmises par le biais d'un convertisseur d'horloge et de largeur de données, afin d'être écrites par le bloc VDMA dans la mémoire DDR.

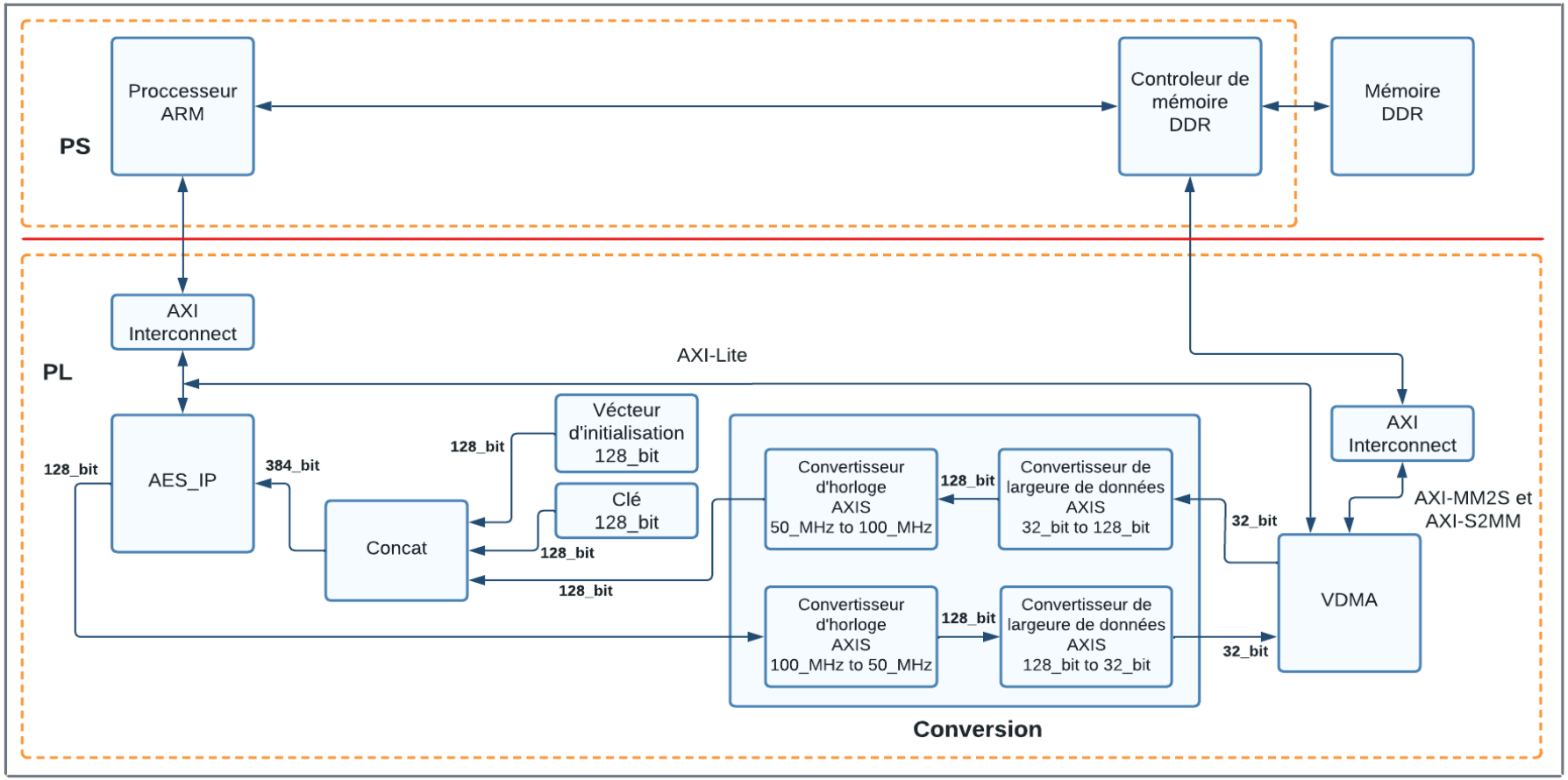

Figure 3.24 : Conception de l'architecture proposée

3.10 PERFORMANCE DE L'ARCHITECTURE PROPOSÉE

Dans cette section nous allons présenter les résultats d'implantation de l'architecture proposée obtenus dans Vivado Design Suite. Une analyse détaillée de l'utilisation des ressources, de la puissance consommée et la vérification de contrainte de temps.

3.10.1 Estimation des ressources utilisées

Après la conception du matériel dans Vivado Design, nous avons effectué la synthèse et l'implantation dans le but de générer le fichier (bitstream) nécessaire à la configuration du circuit ZYNQ. Le schéma-bloc du projet dans le logiciel Vivado Design est présenté à la Figure 3.14. Avec ce qui précède, un fichier ayant une extension (.bit) a été créé, qui contient la conception du matériel pour la programmation du circuit ZYNQ au démarrage de l'application.

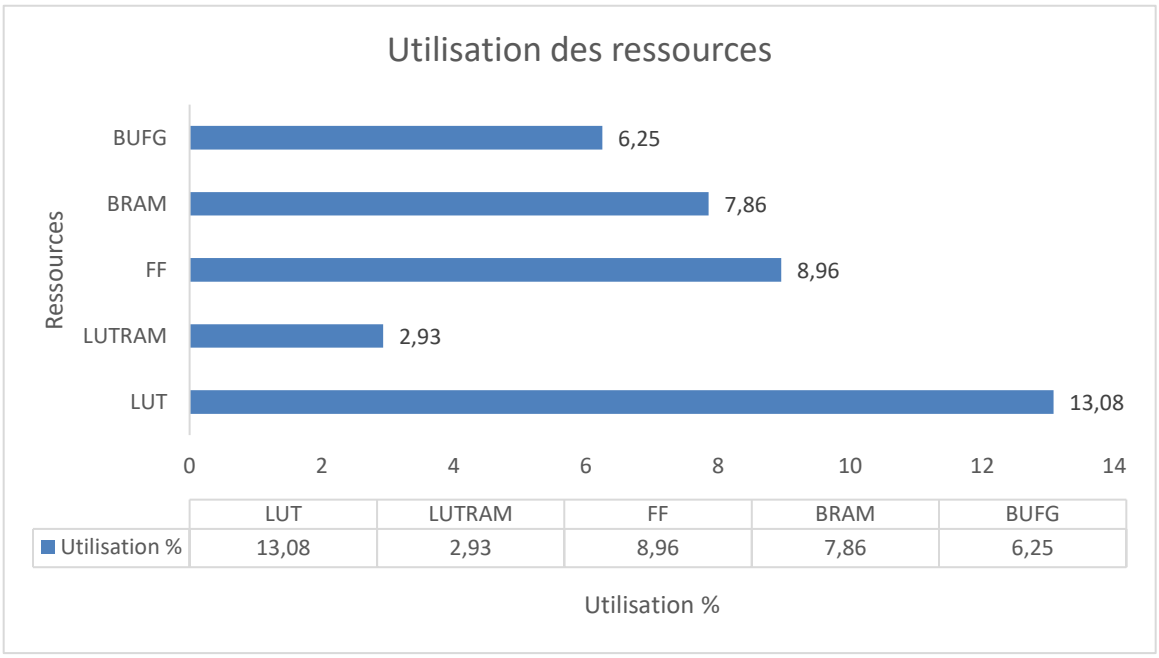

Figure 3.25 : Représentation graphique de l'utilisation de ressources

La Figure 3.25 montre comment les ressources du circuit ZYNQ sont utilisées par l'implantation de la conception proposée. Le rapport d'utilisation décompose l'utilisation de la conception en fonction du type de ressource. L'architecture consomme environ 13% des LUTs disponibles, 9% des Flip-Flop, LUTRAM 3%, BRAM 8%, ainsi que 6% de BUFG.

3.10.2 Estimation de la consommation de puissance

La Figure 3.26 montre à quel point les ressources de la conception proposée consomment de puissance.

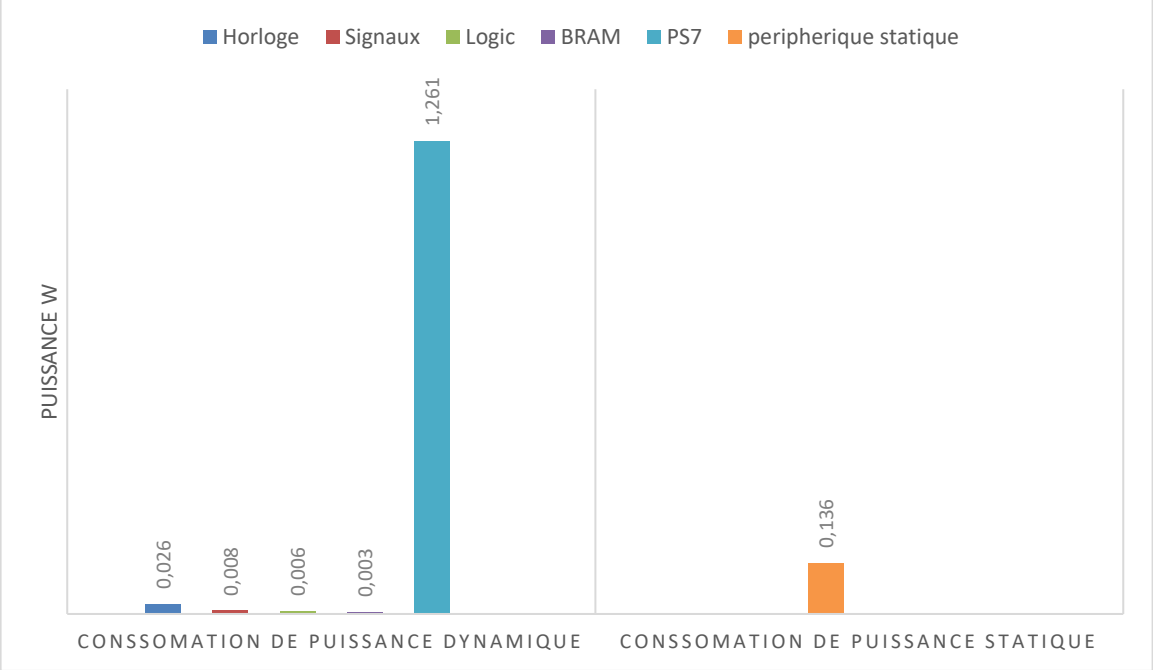

Figure 3.26 : Rapport de consommation de puissance

La puissance statique du dispositif est de 136mW (Device static power), c'est la puissance provenant des pertes de transistors sur tous les rails de tension connectés et les circuits nécessaires pour que le circuit ZYNQ fonctionne normalement après sa configuration. La puissance dynamique totale est de 1.305 W pour sa part, c'est la puissance

de la conception interne (selon le modèle de données d'entrée et à l'activité interne de la conception). Notre architecture consomme une puissance de 43 mW, ce qui est représenté par la somme de l'Horloge, des Signaux, de la Logique et des BRAM (Block Random Access Memory). Le processeur PS7 consomme beaucoup plus d'énergie que le PL ; cela est dû au fait que le système de traitement ARM a une fréquence de fonctionnement beaucoup plus élevée que le PL. L'addition des deux puissances statique et dynamique donne la puissance totale sur puce (Total On-Chip Power) qui est de 1.441 W. Pour ce qui est de la température de jonction, elle atteint 41.6°C.

3.10.3 Estimation de contraintes de temps

Le rapport d'analyse du temps d'exécution de la conception proposée est illustré à la Figure 3.27. Toutes les contraintes temporelles sont respectées.

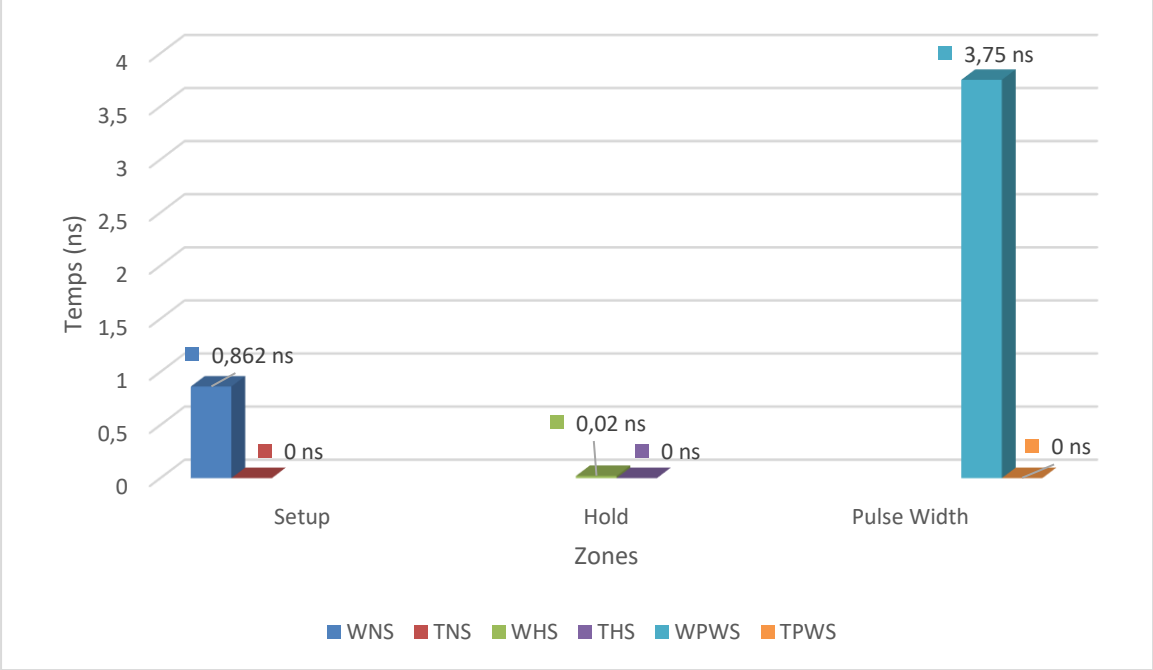

Figure 3.27 : Rapport de temps de la conception

La zone de configuration (*Setup*) affiche toutes les vérifications liées à l'analyse du retard maximum : configuration, récupération et vérification des données. On peut y retrouver le WNS (Worst Negative Slack), avec une valeur de 0.862 ns, qui correspond au pire des écarts de tous les chemins de synchronisation pour l'analyse du retard maximum. Il y a aussi le TNS (Total Negative Slack), avec une valeur de 0 ns, qui signifie que toutes les contraintes de temps sont respectées pour l'analyse du retard maximum.

La zone *Hold* affiche toutes les vérifications liées à l'analyse du retard minimum (retenue, suppression et vérification des données). On peut y retrouver le WHS (Worst Hold Slack), avec une valeur de 0.020 ns, qui correspond au pire des écarts de tous les chemins temporels pour l'analyse des retards mineurs. Il y a aussi le THS (Total Hold Slack), avec une valeur de 0 ns, ce qui signifie que toutes les contraintes de temps sont respectées pour l'analyse du retard minimum.

La zone *Pulse Width* affiche toutes les vérifications relatives aux limites de commutation des broches telles que la largeur minimale d'impulsion basse/haute, la période minimale/maximale et le décalage maximal. Le WPWS (Worst Pulse Width Slack) d'une valeur de 3.75 ns, qui correspond au pire des écarts de toutes les vérifications de temps listées ci-dessus en utilisant les retards min et max. De plus, le TPWS (Total Pulse Width Slack) avec une valeur de 0 ns, ce qui fait en sorte que toutes les contraintes liées sont respectées.

3.11 MISE EN ŒUVRE DU SYSTÈME COMPLET ET FONCTIONNEMENT

L'architecture matérielle du système complet a été décrite dans la section 3.10 et présentée à la Figure 3.14. À l'exception du bloc AES qui a été créé à partir du code C/C++ en utilisant Vivado HLS, les autres blocs de diagramme sont fournis par Vivado Design. La dernière étape consiste à développer une application logicielle qui permettra d'utiliser l'architecture matérielle réalisée pour chiffrer et déchiffrer les données (signaux ou images).

Deux approches ont été adoptées pour compléter le système. Elles diffèrent par leurs niveaux d'abstraction (ou leurs complexités).

Un système embarqué complet a été créé. Nous avons commencé par générer le bloc AES IP à partir de Vivado HLS qui permet d'effectuer les opérations soit de chiffrement et de déchiffrement. Ensuite à l'aide de Vivado design, nous avons importé le bloc AES afin d'implanter notre architecture matérielle pour générer le fichier de configuration bitstream, qui sera utilisé dans les deux approches. La première approche consiste à utiliser la plateforme Vitis IDE pour développer l'application logicielle et créer le boot image qui contient notre architecture. Par la suite le boot image sera copiée sur la carte Micro-SD afin d'établir la communication entre le terminal de la plateforme Vitis IDE et le circuit ZYNQ. La deuxième approche se base sur l'utilisation Jupyter Notebook qui est intégrée auparavant dans la carte Micro-SD grâce à l'image PYNQ de Xilinx. Les fichiers *.bit et *.hwh qui contient le bitstream et l'overlay seront copiés directement sur la carte Micro-SD afin d'établir une communication directe avec le ZYNQ.

Le système embarqué fonctionne de telle sorte que les données sont lues à partir de la carte mémoire Micro-SD et sont chargées dans la mémoire DDR (Double Data Rate) de la carte PYNQ-Z2. Plus tard, ces données sont relues et transmises au bloc IP AES qui, à son tour, produit des données chiffrées. Ces données sont écrites à un autre emplacement de la mémoire DDR et enfin, ces emplacements de mémoire sont lus séquentiellement pour réécrire les données chiffrées sur la carte Micro-SD. C'est la même opération utilisée pour le déchiffrement. La Figure 3.28 résume le schéma de fonctionnement de notre projet complet avec les deux approches.

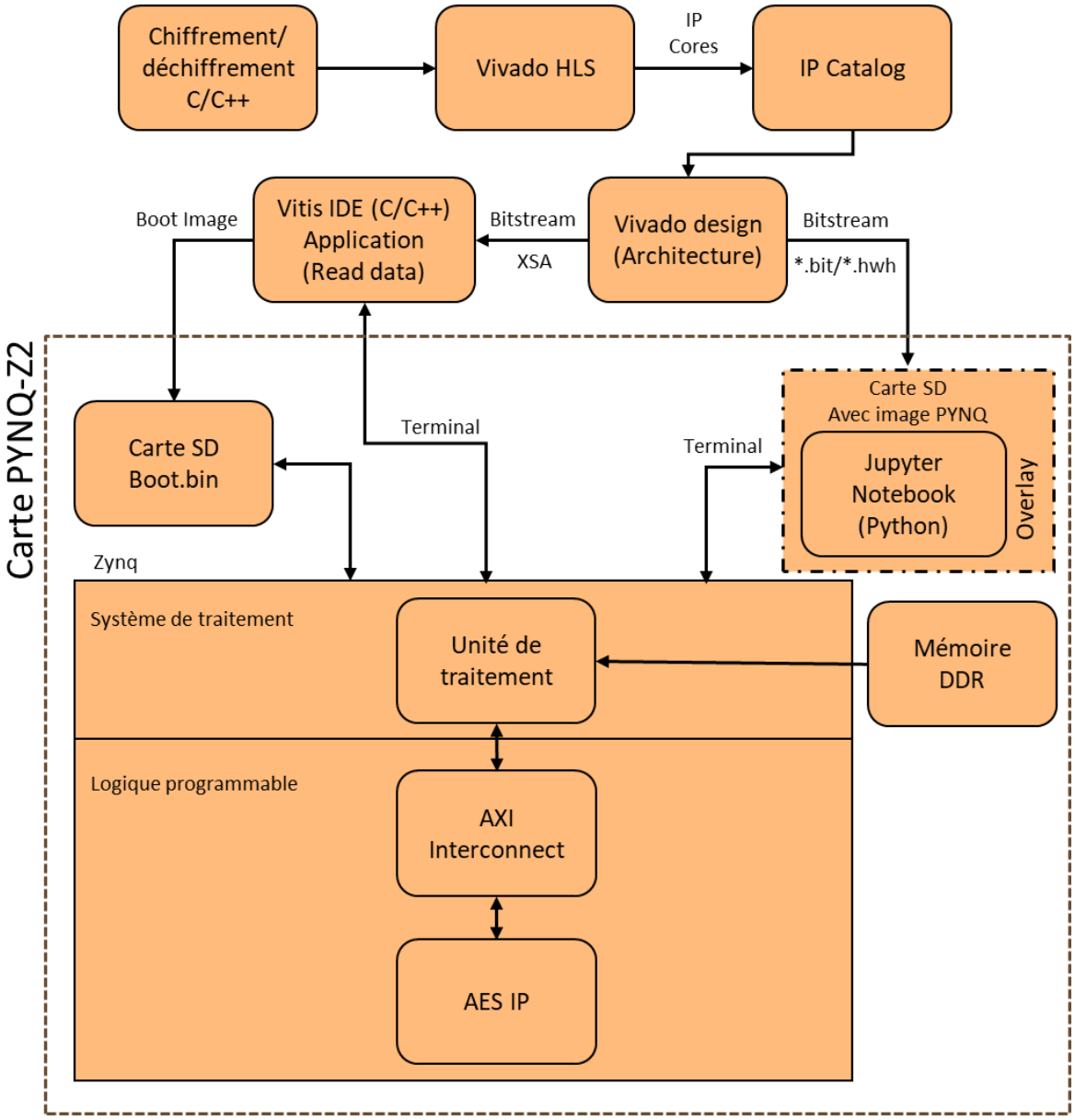

Figure 3.28 : Schéma de fonctionnement pour l'implantation du système sur le ZYNQ

3.11.1 Implantation logicielle bas niveau en utilisant Vitis IDE

Xilinx Vitis est un environnement de développement intégré IDE (Integrated Design Environment) qui permet de concevoir, de déboguer et de déployer des systèmes à base de

circuits FPGA et ASIC. Il comprend une interface graphique et une interface en ligne de commande. La Figure 3.29 montre le processus de développement d'applications sur Vitis IDE.

Lorsqu'on ouvre la plate-forme logicielle Vitis IDE, nous pouvons créer un espace de travail en ajoutant un projet application à partir du menu principal. Une fois notre projet application est créés, nous pouvons importent l'environnement XSA (Xilinx System Architecture) incluant le fichier bitstream, exporté à partir de Vivado design, qui contient les spécifications matérielles telles que les propriétés de configuration du processeur, les informations de connexion des périphériques, la liste des adresses et le code d'initialisation du périphérique. Ensuite, nous définissons le domaine dans lequel nous allons exécuter notre application. Le domaine est un paquet de support de carte BSP (Board Support Package) contenant le système d'exploitation OS (Operating System) et une collection de pilotes logiciels sur lesquels nous pouvons construire notre application. Enfin, nous pouvons créer un projet d'application à l'aide du le code C/C++ afin de générer le boot image (Xilinx, 2022a).

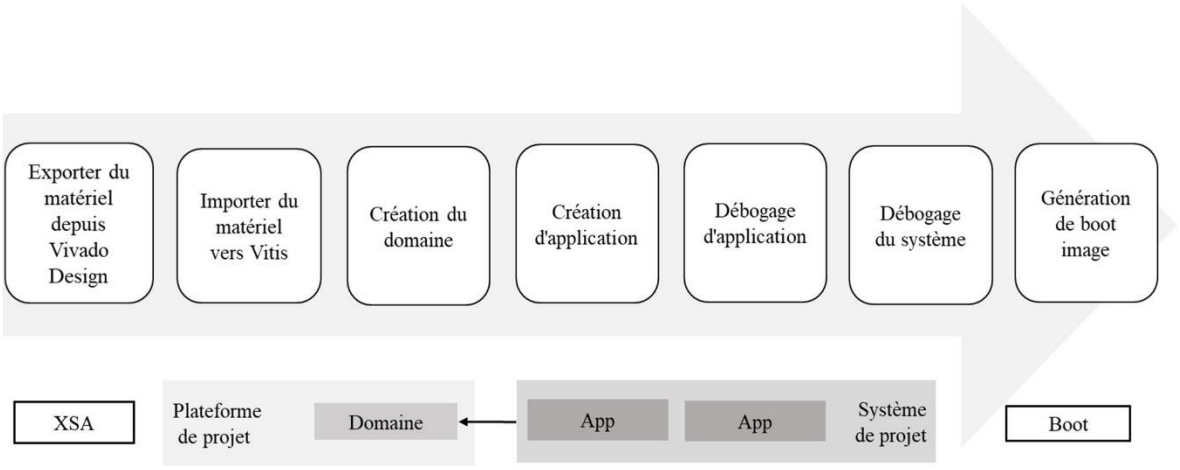

Figure 3.29 : Processus de développement d'applications sur Vitis (Xilinx, 2022a).

Dans l'interface de Vitis IDE, nous disposons de plusieurs fenêtres telles que présentées à la Figure 3.30. La fenêtre Explorer affiche une arborescence des dossiers du projet et de leurs fichiers sources associés, ainsi que des fichiers de compilation et des rapports générés par l'outil. Nous pouvons l'utiliser pour explorer la hiérarchie des fichiers de notre projet. La fenêtre Assistant fournit une arborescence de projet pour gérer et exécuter les configurations et définir les attributs de ces dernières. Il s'agit d'une vue associée à la vue explorée.

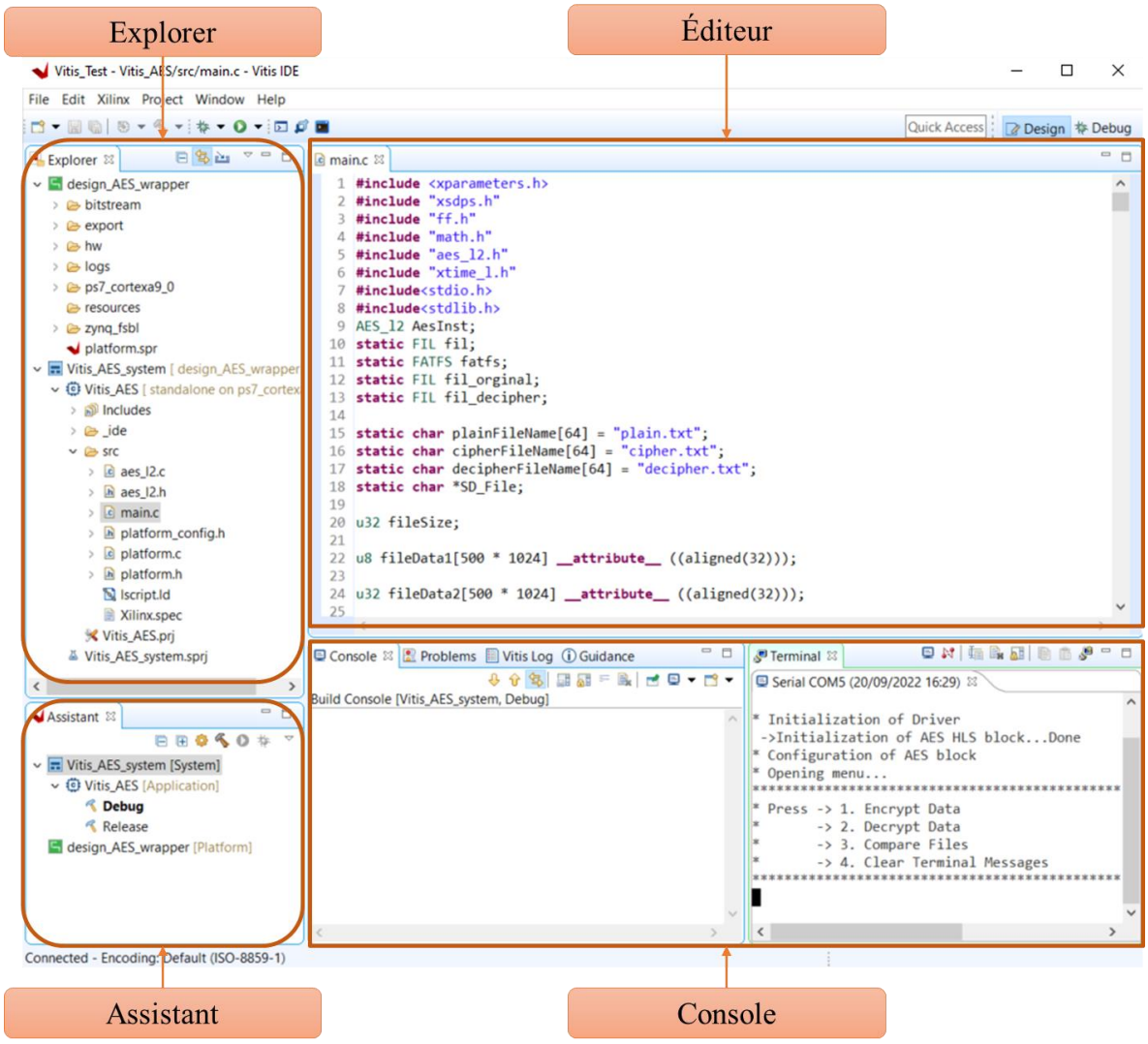

Figure 3.30 : Interface Vitis IDE

La fenêtre Éditeur affiche le projet en cours, la plate-forme cible, la configuration de compilation active et les fonctions matérielles spécifiées. Il permet également d'écrire, afficher ou modifier les codes dans le dossier source (src) et les paramètres du projet. La fenêtre Console présente plusieurs sous-fenêtres, notamment le terminal de commande, les instructions de conception, les propriétés du projet, les rapports et la vue du terminal (Xilinx, 2022). La fenêtre main.c est la fonction principale où le code est écrit, telle que les déclarations des en-têtes de fichier pour avoir accès aux fonctions de la bibliothèque de chaque bloc IP, la déclaration des variables d'instance et des variables globales. Elle contient également la boucle principale (main loop) responsable de l'exécution du code qui fait appel à des fonctions déclarées auparavant. Une fois le code achevé, nous passons à la construction du projet (Build Project). Ensuite, nous créons le boot image avant de passer au lancement du projet à partir de la carte Micro-SD de la carte PYNQ-Z2.

3.11.2 Implantation logicielle haut -niveau en utilisant Jupyter/Python

Pour la seconde approche, nous avons besoin d'installer une image PYNQ sur une carte Micro-SD pour démarrer notre PYNQ-Z2. L'image PYNQ (composé du *Ubuntu boot*, *Python 3.6.5* et *Jupyter Notebook*) contient toutes les bibliothèques et les pilotes permettant de gérer la carte PYNQ-Z2. Au niveau de la carte PYNQ-Z2 nous avons placé le jumper (JP4/Boot) sur la position SD. Cela permet le lancement du PYNQ à partir de la carte Micro-SD. Pour alimenter la PYNQ-Z2 à partir du câble micro-USB, nous avons placé le jumper (JP5/Power) en position USB. Rendu à cette étape, nous insérons la carte Micro-SD et en connecte le câble USB ainsi le câble Ethernet à un ordinateur en suivant les instructions fournit par Xilinx (2021). Nous avons configuré la carte réseau pour avoir un accès direct à la carte PYNQ-Z2. Cela nous permet de copier les fichiers (*.bit et *.hwh) incluant le bitstream généré par Vivado Design et les données que nous voulons crypter/décrypter (signaux ou images). Il y a deux méthodes pour copier les fichiers sur la carte PYNQ-Z2; la première méthode est de passer par le navigateur Web pour avoir accès à Jupyter on utilisant

l'adresse IP de votre configuration (Example http://192.168.2.99:9090), sur la page d'accueil il y a le bouton *upload* qui nous permet de charger nous fichiers sur la carte. La deuxième méthode et d'aller sur le poste de travail; emplacement réseaux/Xilinx (PYNQ)/jupyter_notebooks, pour copier nos fichiers sur la carte PYNQ-Z2 (Figure 3.31).

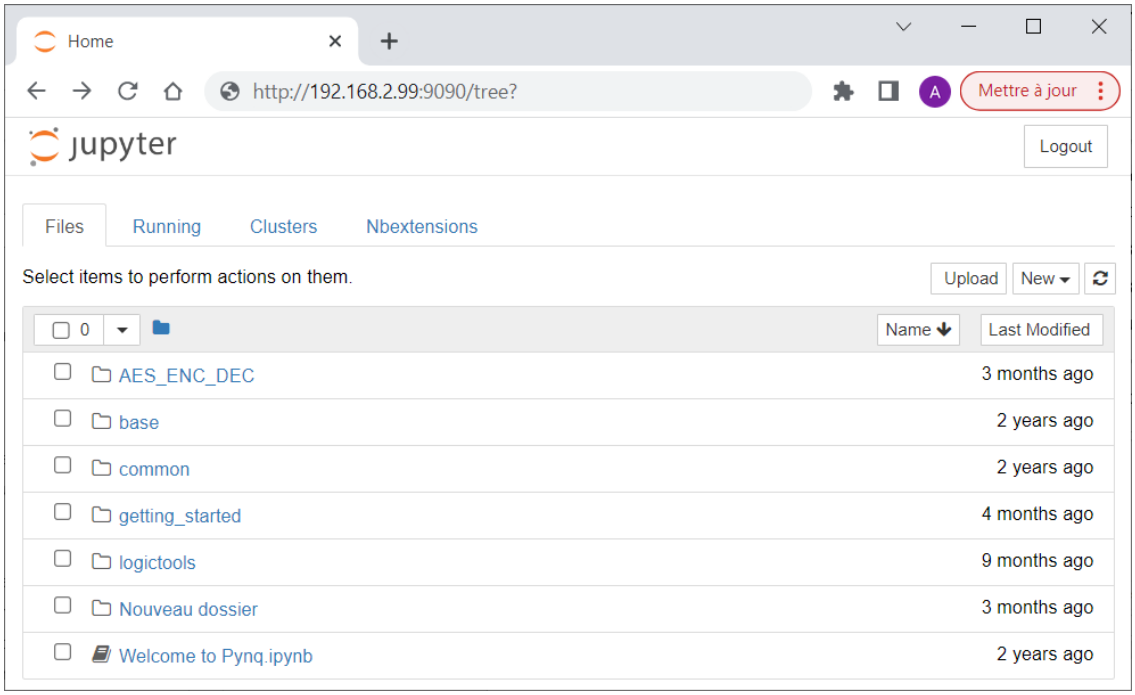

Figure 3.31 : Page d'accueil de l'interface Jupyter

L'étape suivante est de créer un nouveau notebook qui va nous permettre de travailler avec l'environnement Python et quelques modules supplémentaires nécessaires à la manipulation de la carte PYNQ-Z2.

Parmi les bibliothèques nécessaires au démarrage de PYNQ, on trouve l'Overlay de notre architecture matérielle réalisée sous Vivado Design. Les Overlays, ou bibliothèques matérielles, sont des conceptions FPGA programmables/configurables qui étendent l'application utilisateur du système de traitement du ZYNQ à la logique programmable. Les Overlays peuvent être utilisés pour accélérer une application logicielle ou pour personnaliser

la plate-forme matérielle pour une application particulière. L'image PYNQ fournit une interface Python permettant de programmer et contrôler les overlays dans le PL à partir de Python exécuté dans le PS.

La Figure 3.32 présente une cellule de code Jupyter dans le notebook. Le notebook est composé d'une séquence de cellules. Il offre des fonctionnalités d'édition et d'exécution de code Python dans un environnement interactif. Le fichier de notebook est enregistré avec l'extension ipynb.

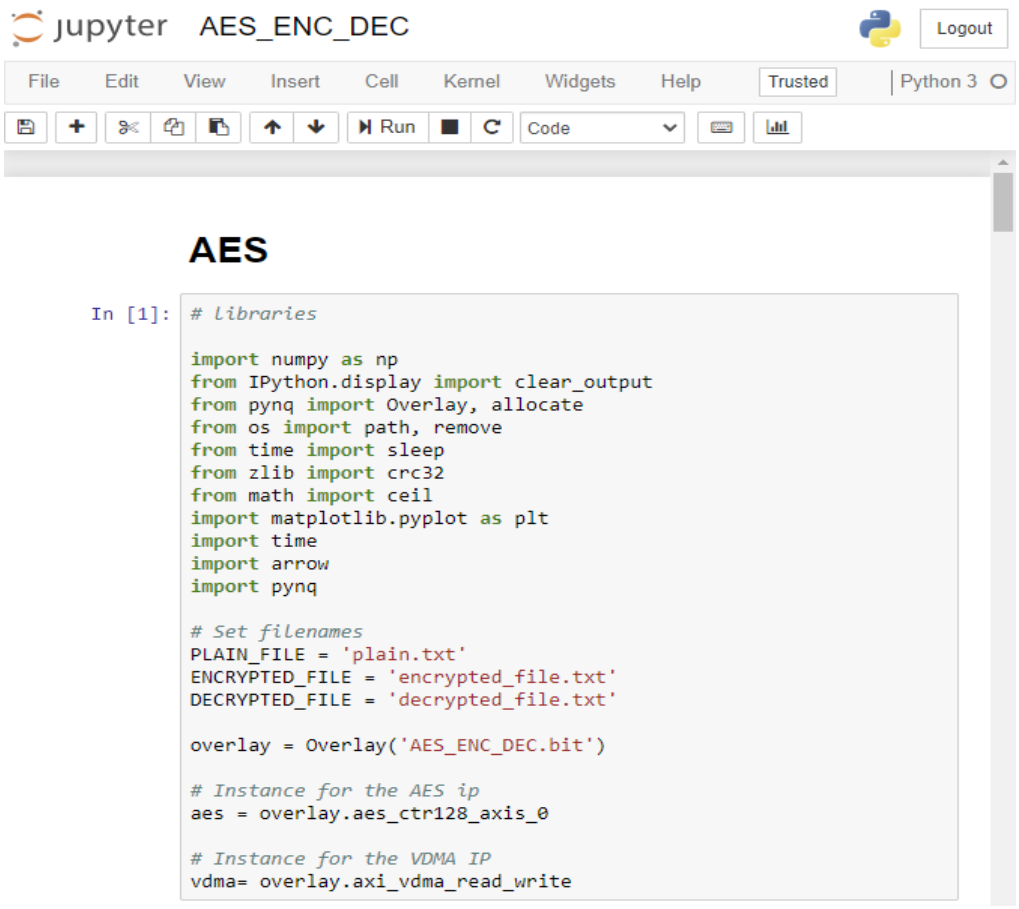

Figure 3.32 : Interface utilisateur Jupyter

CHAPITRE 4
EXPÉRIMENTATION ET RÉSULTATS

Dans ce chapitre, nous allons décrire les bancs de test de chacune des deux approches d'implantation de l'algorithme de chiffrement/déchiffrement AES-128-CTR. Les deux approches utilisent la même architecture matérielle développée sous Vivado Design et qui inclue le module AES IP, conçu sous Vivado HLS. Cependant, elles diffèrent par la partie logicielle qui exploite ou gère la partie matérielle pour son utilisation dans le chiffrement des signaux ou des images. La première approche, de bas-niveau d'abstraction, utilise l'outil Vitis IDE pour programmer en C/C++ la partie logicielle du projet. Tandis que la seconde partie, de haut niveau d'abstraction, utilise la plateforme Jupyter pour programmer en Python la partie logicielle du projet. Ce chapitre présente également les résultats de chiffrement/ déchiffrement obtenus par les deux approches, ainsi que les critères d'évaluation de leurs performances.

4.1 APPROCHES PROPOSÉES ET LEURS BANCS DE TEST

Dans cette section, nous allons présenter les deux approches logicielles utilisant deux plateformes différentes, soient Jupyter Notebook et Vitis IDE, dans le but d'exploiter notre architecture matérielle de chiffrement/déchiffrement et de la tester avec des données réelles (signaux et images).

Nous avons utilisé deux cartes-SD, la première contient le boot image (*.bin) généré par Vitis IDE et le fichier texte de données ECG (ElectroCardioGram). La deuxième, pour sa part, contient le boot image PYNQ version 2.6 et les fichiers de configuration (*.bit et*.hwh) et les deux fichiers (*.txt), (*.csv) de données ECG MIT-BIH (Normal Sinus Rhythm Database) de durée d'une minute fournie par la plateforme PhysioNet. Le fichier (*.csv) est un format de données tabulaires, ce qui signifie que les données sont stockées dans une table en ligne par ligne. Les valeurs d'une même ligne (colonnes) sont séparées par des

virgules, tandis que chaque ligne correspond à une nouvelle entrée. Cependant le fichier (*.txt) est un format de données texte, ce qui signifie que les données sont stockées dans une table en ligne par ligne. Les valeurs d'une même ligne (colonnes) sont séparées par des espaces, tandis que chaque ligne correspond à une nouvelle entrée.

4.1.1 Approche bas-niveau d'abstraction sous Vitis-IDE

Le banc de test de la première approche est présenté à la Figure 4.1. En plus de l'architecture matérielle développée sous Vivado Design (XSA), le développement de la partie logicielle nécessite un ordinateur muni de l'outil Vitis IDE. La programmation consiste à développer une application en langage C/C++ permettent d'acquérir, de lire et de préparer les données avant dès les envoyer vers la partie matérielle qui fera le chiffrement/déchiffrement. Cette partie logicielle peut également récupérer le résultat avant de l'afficher, l'envoyer ou le sauvegarder. Une fois le code C/C++ de cette partie logicielle est complété, il sera compilé. Une image de démarrage (Boot Image) inclut la configuration des deux parties (matérielle et logicielle), sera créée et sauvegardée sur une mémoire Micro-SD. Cette dernière peut contenir des données de test (ECG, Audio et images).

Le test de fonctionnement de l'implantation de l'algorithme AES sur le circuit ZYNQ se fait par la programmation automatique au démarrage de la carte PYNQ-Z2 à partir du boot image placée sur la mémoire Micro-SD. L'ensemble de processus peut être suivi par l'utilisateur. Par le biais du terminal série, l'utilisateur peut choisir le chiffrement ou le déchiffrement des données.

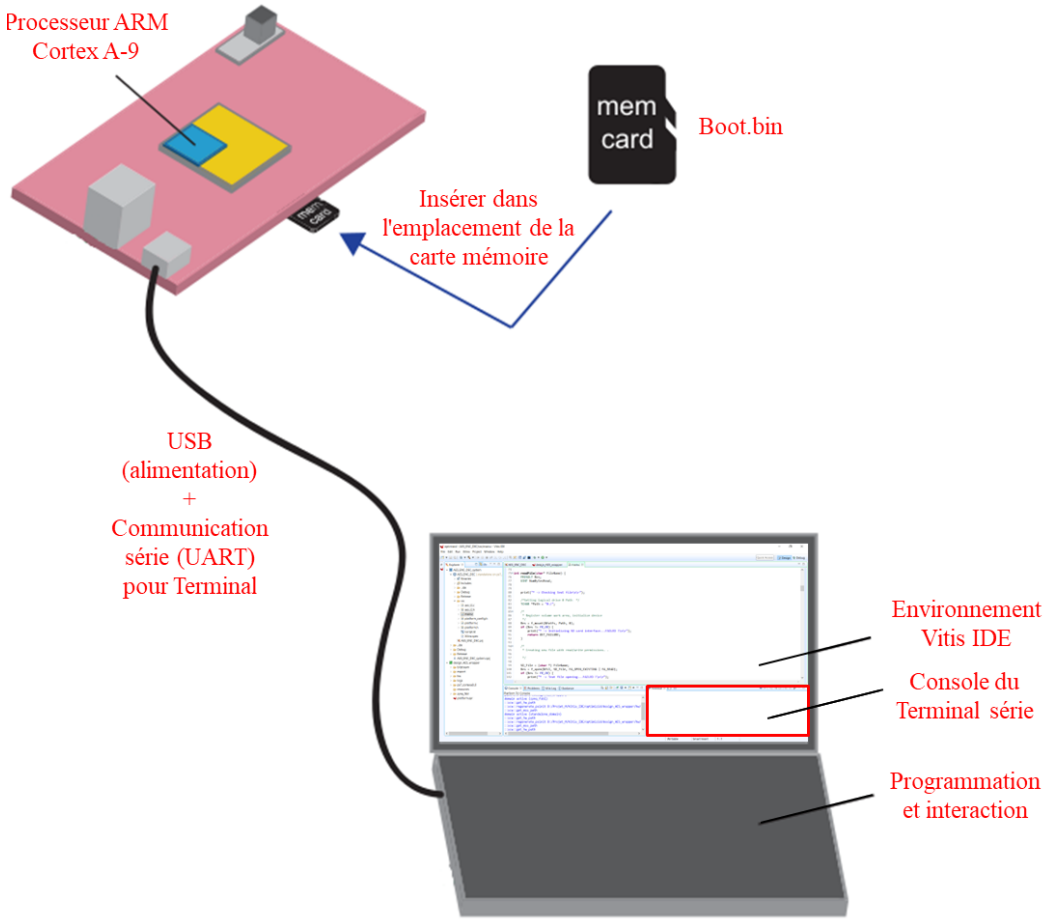

Figure 4.1 : Interaction de la carte PYNQ-Z2 avec Vitis IDE (RkBlog, 2020)

4.1.2 Approche haut-niveau d'abstraction sur Jupyter

Le banc de test de la seconde approche est présenté à la Figure 4.2. Dans cette seconde approche, le développement de la partie logicielle se fait directement sur la carte PYNQ-Z2 muni de l'environnement PYNQ (image contenant *Ubuntu boot*, *Python 3.6.5* et *Jupyter Notebook*). Cette carte peut être alimentée par un câble USB à partir d'un laptop. Le développement du programme Python se fait dans l'environnement Jupyter Notebook à partir d'un laptop relié à la carte PYNQ-Z2 via un câble réseau Ethernet. Les adresses IP (Internet

Protocole) du laptop et de la carte PYNQ-Z2 doivent être configurées pour appartenir au même réseau local Ethernet.

Le codage de la partie logicielle sous Python est plus simple (facile) à cause de la disponibilité des bibliothèques de fonctions. Comparativement au langage C/C++, Python ne nécessite pas d'être compilé. Jupyter Notebook le rend très convivial.

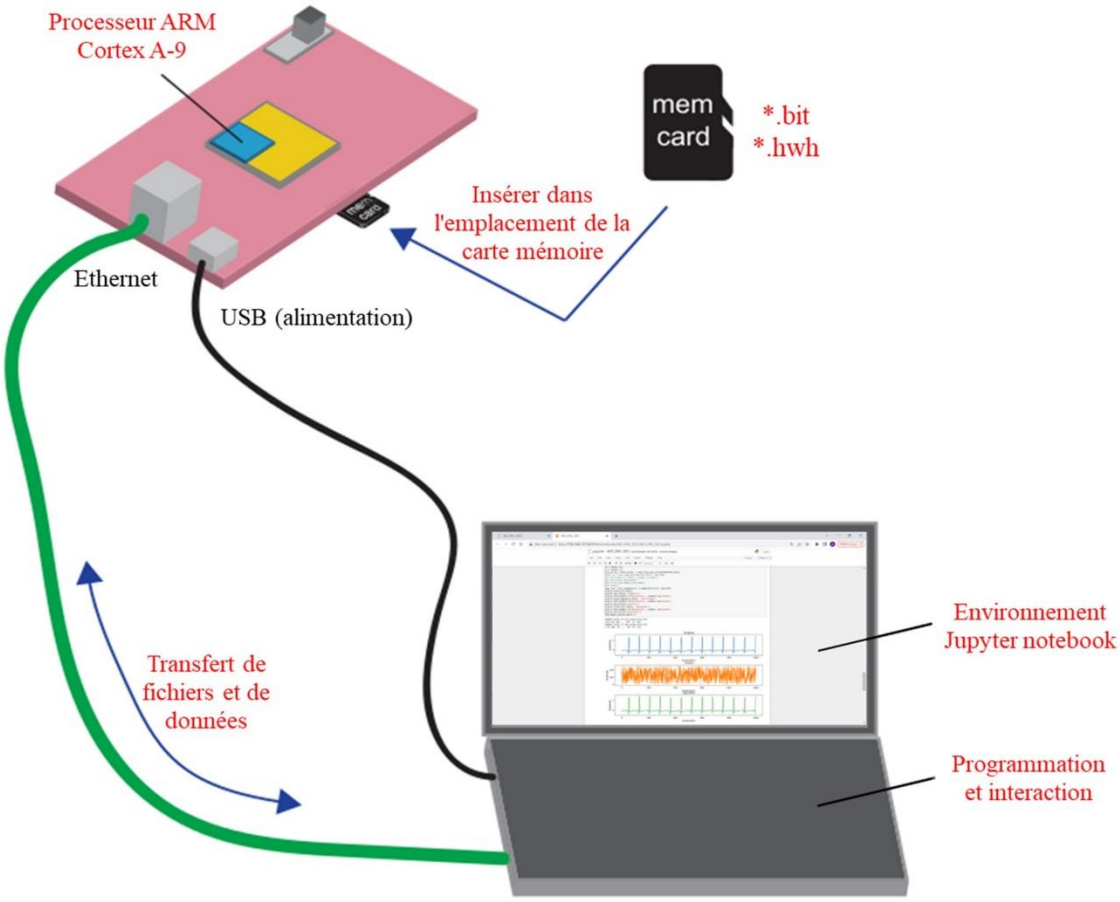

Figure 4.2 : Interaction de la carte PYNQ-Z2 avec Jupyter Notebook (RkBlog, 2020)

4.1.3 Implantation purement logicielle

L'implantation purement logicielle sur Jupyter utilise le même banc de test utilisé par la seconde approche de haut-niveau d'abstraction (Figure 4.2). Le développement et

l'exécution de la partie logicielle se font directement sur la partie PS de la carte PYNQ-Z2 sans avoir besoin de bibliothèque matérielle (Overlay) en lien avec la partie PL. Le code écrit, purement logiciel, est basé sur les bibliothèques prêtes à l'emploi de Python. L'implantation de cette dernière utilise la même unité de traitement PS que la deuxième approche matérielle. Elle nous permet d'avoir une comparaison de résultats de chiffrement/déchiffrement ainsi que le temps d'exécution. La Figure 4.3 représente notre espace de travail qui comporte les deux interfaces Vitis IDE et Jupyter Notebook sont affichés respectivement sur les écrans de droite et de gauche, un laptop et une carte PYNQ-Z2.

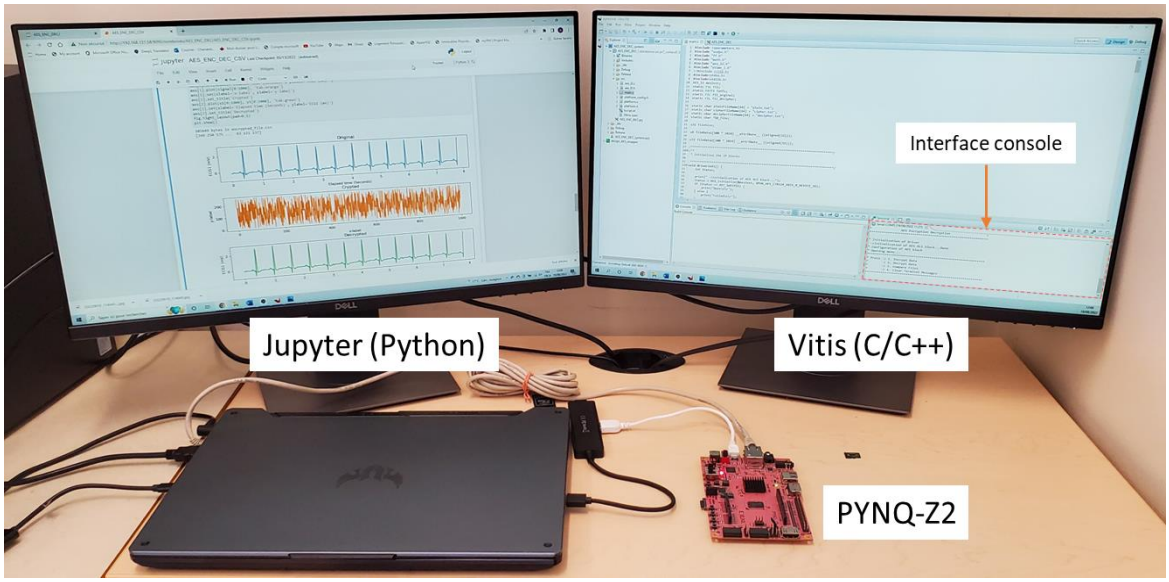

Figure 4.3 : Espace de travail

4.2 CRITÈRE DE FIABILITÉ

Nous avons utilisé le coefficient de corrélation de Pearson pour évaluer la similarité des données déchiffrées par rapport aux données originales. Le coefficient de Pearson est un indicateur qui représente une mesure de la linéarité de la relation entre deux variables. Le coefficient de Pearson peut varier entre -1 et 1. Une valeur proche de 0 indique une relation faible, ce qui signifie qu'il n'y a pas ressemblance. Une valeur proche de 1 indique une relation forte, ce qui signifie que les deux variables évoluent dans le même sens. Enfin une

valeur proche de -1 indique une relation inverse. Lorsqu'une variable augmente, l'autre variable diminue. La formule suivante est utilisée pour calculer le coefficient de corrélation de Pearson :

$$r_{xy} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y - \bar{y})^2}} \quad (4.1)$$

La valeur de r_{xy} est le coefficient de corrélation estimé entre deux variables continues. x_i et y_i représentent les points d'échantillonnage, $\bar{x}$ et $\bar{y}$ représentent la moyenne des deux variables $\bar{x}$ et $\bar{y}$, tandis que le N représente la taille de l'échantillon. La relation entre deux variables peut être visualisée à l'aide de nuages de points et la droite de régression. Les nuages de points représentent la dispersion des valeurs prises par les deux variables. La droite de régression montre la relation moyenne entre les deux variables. Plus la droite est proche des points, plus la relation est forte.

4.3 CHIFFREMENT/DÉCHIFFREMENT D'UN SIGNAL ECG

Dans cette section, nous allons évaluer les résultats des deux approches de nos implantations, Vitis IDE et Jupyter Notebook. En ce qui concerne Jupyter Notebook, il y aura l'utilisation de deux implantations, une purement logicielle et l'autre matérielle sur la carte FPGA ZYNQ-XC7Z020. Nous allons utiliser un enregistrement électrocardiogramme ECG de la base de données ECG MIT-BIH (Goldberger *et al.*, 2000) avec deux formats de fichier différent *.txt, *.csv. Cette démarche a pour objectif de pouvoir comparer les résultats obtenus en termes de vitesse d'exécution de l'architecture implantée sur FPGA.

4.3.1 Résultat de chiffrement

Les résultats de l'implantation chiffrement/déchiffrement des deux approches matérielles, ainsi que l'implantation purement logicielle sont présentés à la Figure 4.4. Le signal ECG est représenté dans un intervalle de 1000 échantillons où le premier graphe du haut représente le signal original.

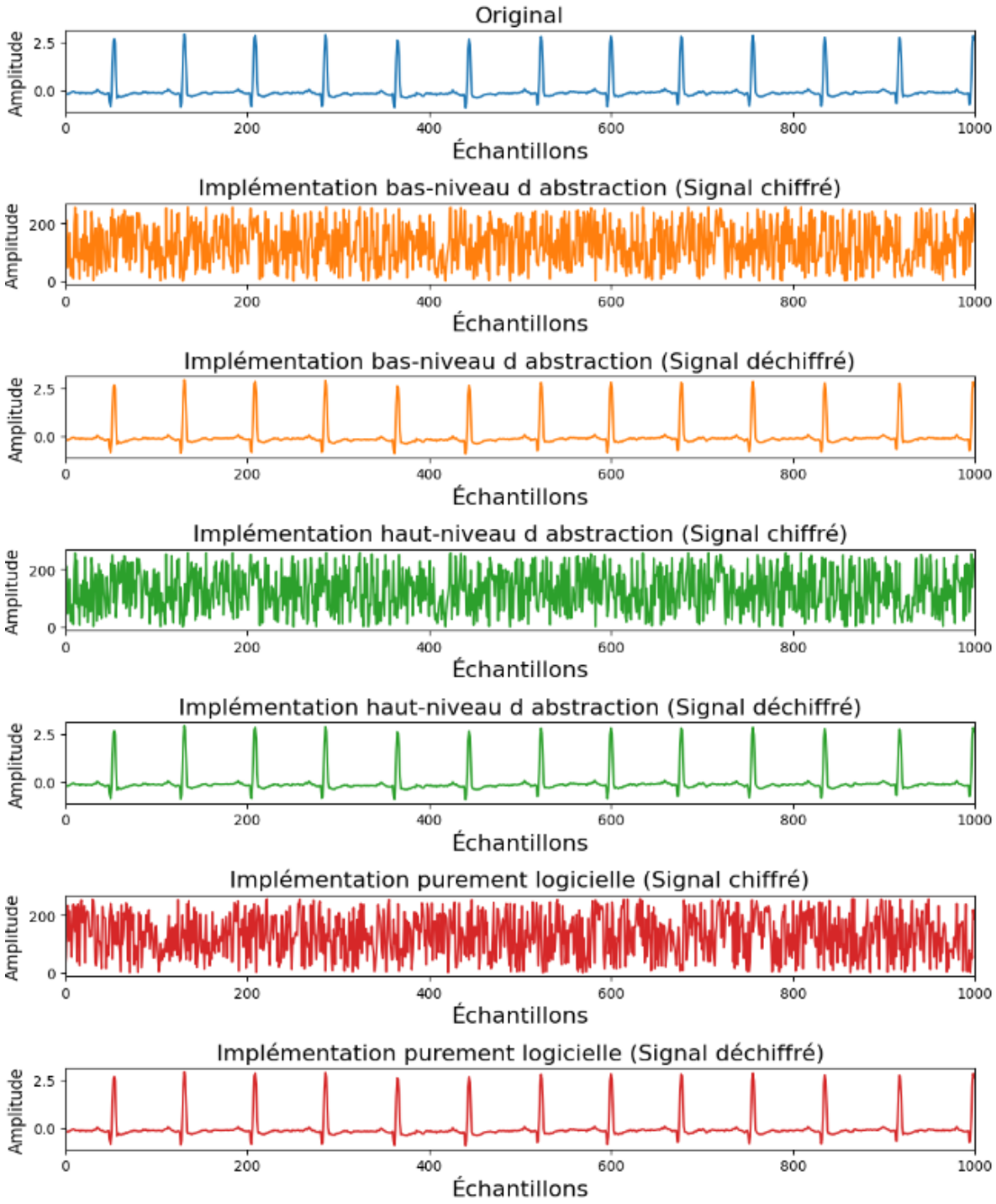

Figure 4.4 : Représentation de signal, original, chiffré et déchiffré entre 0 et 1000

Le signal original est parfaitement récupéré par le signal déchiffré dans les trois implantations en respectant l'amplitude et le temps. Cependant, dans les trois implantations, le signal chiffré a une amplitude d'environ 250 sur un intervalle de 1000 échantillons. Ce qui signifie que notre signal original a été chiffré avec succès. Pour mesurer la relation linéaire entre les deux signaux ECG (ElectroCardioGram) original, chiffré et déchiffré, nous avons utilisé le coefficient de corrélation de Pearson. Le calcul du coefficient de corrélation de Pearson a été fait en utilisant la bibliothèque Numpy. Les Figures 4.5, 4.6 et 4.7 représentent respectivement le taux de linéarité des trois implantations ; implantation matérielle en utilisant les deux approches bas-niveau, haut-niveau et l'implantation purement logicielle.

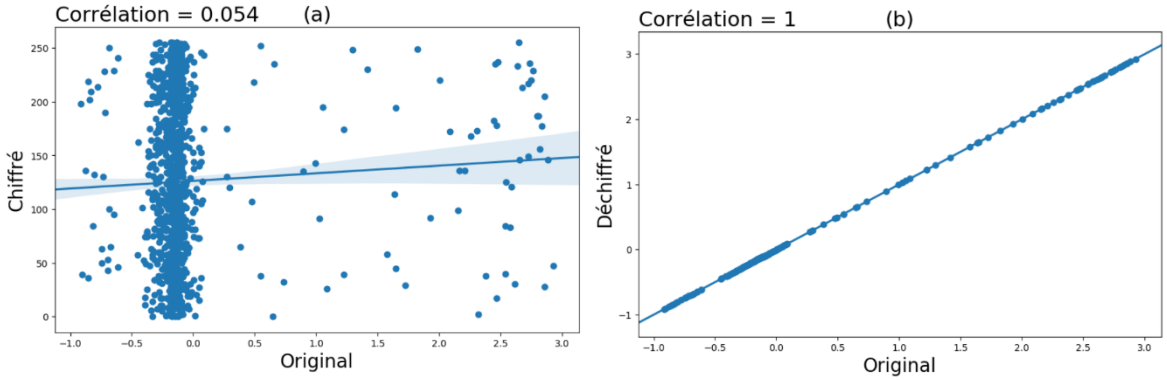

Figure 4.5 : Visualisation de la corrélation de l'approche matérielle bas-niveau, (a) corrélation entre le signal original et le signal chiffré et (b) corrélation entre le signal original et le signal déchiffré.

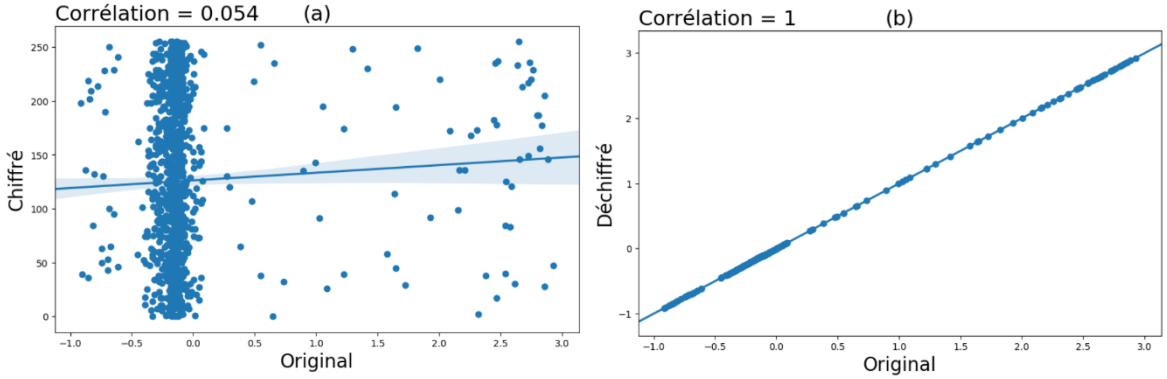

Figure 4.6 : Visualisation de la corrélation de l'approche matérielle haut-niveau, (a) corrélation entre le signal original et le signal chiffré et (b) corrélation entre le signal original et le signal déchiffré.

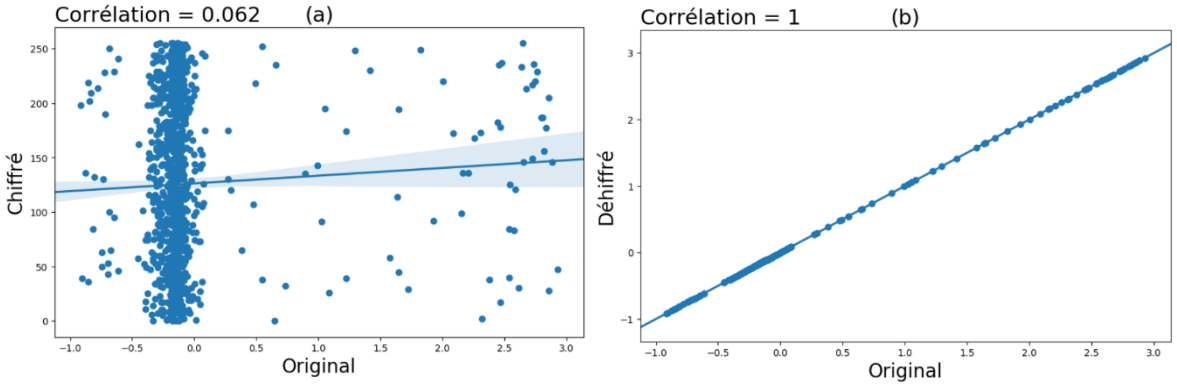

Figure 4.5 : Visualisation de la corrélation de l'approche purement logicielle, (a) corrélation entre le signal original et le signal chiffré et (b) corrélation entre le signal original et le signal déchiffré.

Nous pouvons voir que dans la partie (a) de chaque figure, on constate qu'il n'y a pas de relation entre les deux signaux. Le coefficient de corrélation tend vers 0. Cela signifie qu'ils ne sont pas similaires en aucune façon. Cependant, dans la partie (b) de chaque figure, on peut voir une relation positive avec un coefficient de corrélation qui est égale à 1 entre le signal original et le signal déchiffré. Cela signifie que les deux signaux ont une similarité parfaite.

4.3.2 Évaluation des performances

Les résultats du temps d'exécution de notre architecture utilisée dans les deux approches matérielles bas-niveau et haut-niveau est résumé dans le Tableau 4.1. Cela comprend l'implantation matérielle sur la carte PYNQ-Z2 de chiffrement/déchiffrement AES des deux fichiers (*.txt) et (*.csv) à l'aide de notre architecture exécuter sur deux plateformes Jupyter Notebook et Vitis IDE, ainsi l'implantation purement logicielle de l'algorithme AES.

Le Tableau 4.1 a pour but de comparer la rapidité d'exécution, en termes de temps de traitement, des trois implantations, plus précisément au niveau des deux opérations de chiffrement et de déchiffrement.

Tableau 4.1 : Comparaison de temps d'exécution de chiffrement et de déchiffrement

Opérations	Temps d'exécutions		
	Vitis IDE (C++) (Architecture proposée)	Jupyter Notebooks (Architecture proposée)	Jupyter Notebooks (Logicielle)
Chiffrement de données format (*.txt)	5.42 ms	6.56 ms	23.69 ms
Déchiffrement de données format (*.txt)	5.50 ms	6.52 ms	23.57 ms
Chiffrement de données format (*.csv)	3.18 ms	3.57 ms	12.67 ms
Déchiffrement de données format (*.csv)	3.26 ms	3.59 ms	12.69 ms

À travers les résultats obtenus au Tableau 4.1, nous pouvons voir que le temps d'exécution de chiffrement et de déchiffrement du fichier CSV est environ deux fois plus rapide que dans le cas du fichier TXT, pour les trois implantations. Nous pouvons aussi remarquer que notre architecture matérielle proposée est environ quatre fois plus rapide que l'implantation logicielle et qu'il y a une légère différence au niveau de l'implantation matérielle sur les deux plateformes présentées soit Jupyter et Vitis. L'approche bas-niveau (Vitis) est relativement plus performante que l'approche haut-niveau (Jupyter)

4.4 CHIFFREMENT DE DONNÉES AUDIO ET IMAGE

Nous allons aussi tester notre architecture avec d'autres types de données telles que les données audio et image. Nous avons utilisé la plateforme Jupyter Notebook pour sa simplicité de manipulation, ce qui veut dire ce n'est pas nécessaire de crée le boot image à chaque modification. Nous devons juste modifier notre algorithme selon le besoin de chaque type de donnée, afin de l'exécuter pour avoir les résultats désirés.

4.4.1 Chiffrement des données audio

Dans le cas d'échantillonnage des données audio, nous avons créé un enregistrement d'une durée de 7 secondes avec une fréquence de 8 KHz. Nous avons utilisé la bibliothèque « wav » de Python pour lire les fichiers audio. Ensuite, nous avons converti les données audio en format de données array avec une représentation interne de données, entier non signé 8 bits (uint8). Ce qui signifie que toutes les données seront codées sur 8 bits, avec une plage de valeurs comprise entre 0 et 255.

Une fois l'opération de chiffrement/déchiffrement réalisée, nous avons retrouvé les données originales. Nous avons représenté graphiquement les différentes étapes des deux implantations matérielle et logicielle sur la Figure 4.8. Le premier graphe représente le signal original suivi de résultats de chiffrement/déchiffrement des deux implantations.

Nous pouvons constater visuellement que les deux signaux audio, soient l'original et le déchiffré de chacune des deux implantations (matérielle et logicielle), sont similaires en respectant l'amplitude et la durée d'enregistrement. Cependant, le signal chiffré par chacune des deux implantations ressemble à un bruit blanc sur toute la durée d'enregistrement, avec une amplitude qui varie entre -2^{15} et $2^{15} - 1$.

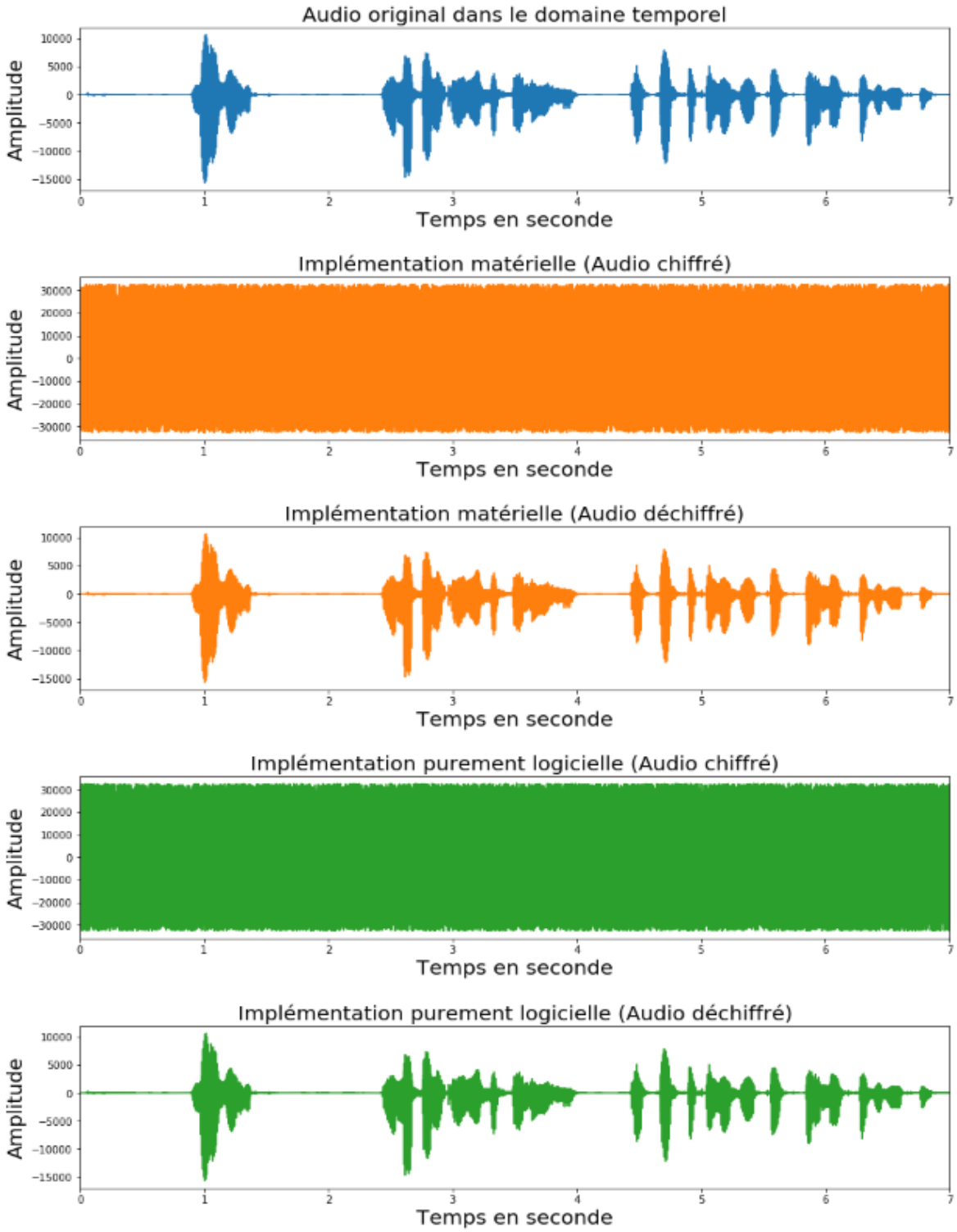

Figure 4.6 : Représentation des signaux décrivant les étapes de chiffrement/déchiffrement pour les deux implantations (matérielle et logicielle)

L’étape suivante est de vérifier la similarité entre les trois signaux audio avec le coefficient de corrélation. Pour chaque signal, nous avons défini le même intervalle temporel en termes d’échantillons [8000 : 12000], pour lequel nous calculons la corrélation entre les signaux. Nous pouvons voir que dans la partie (a) des deux Figures 4.9 et 4.10, il n'y a pas de relation entre les deux signaux (audio originaux et audio chiffrés). Le coefficient de corrélation tend vers 0 dans les deux implantations. Cela signifie qu'elles ne sont pas similaires en aucune façon. Cependant, sur la partie (b) des deux Figures 4.9 et 4.10, on constate une relation positive avec un coefficient de corrélation est égale à 1 entre le signal audio original et le signal audio déchiffré. Cela signifie que les deux signaux ont une similarité parfaite.

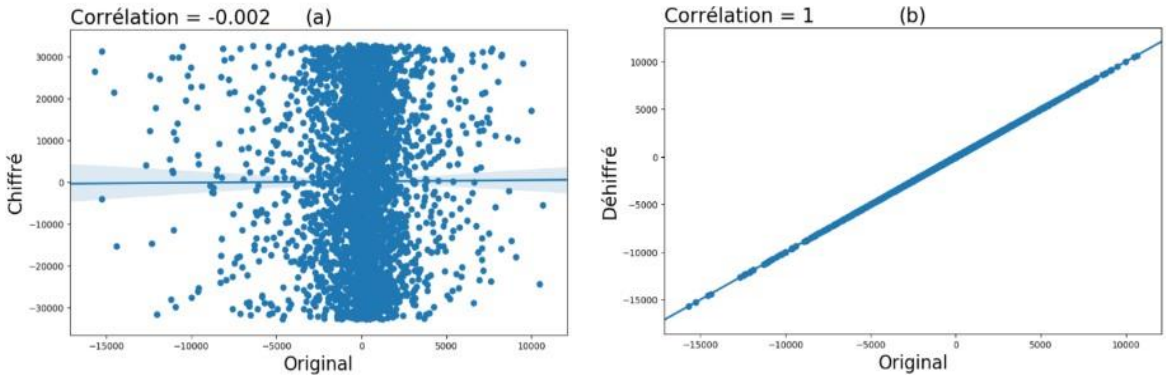

Figure 4.7 : Visualisation de la corrélation de l’implantation matérielle, (a) corrélation entre le signal original et le signal chiffré et (b) corrélation entre le signal original et le signal déchiffré.

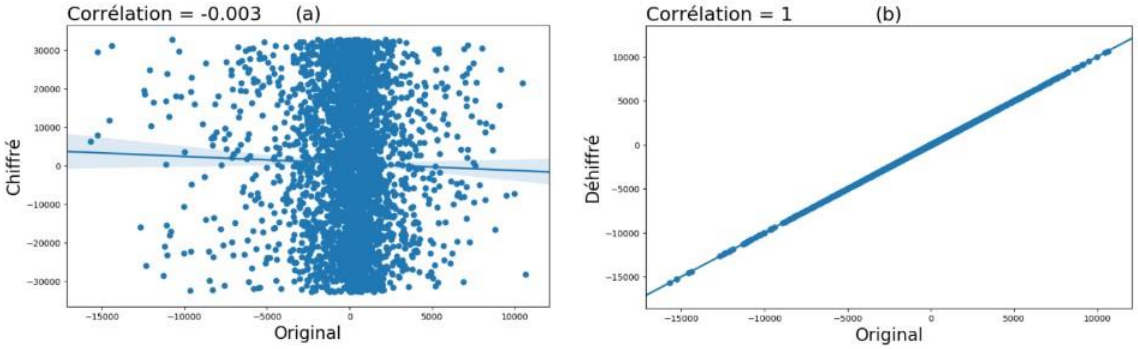

Figure 4.8 : Visualisation de la corrélation de l’implantation purement logicielle, (a) corrélation entre le signal original et le signal déchiffré et (b) corrélation entre le signal original et le signal chiffré.

4.4.2 Chiffrement d'image

Dans le cas de l'image, nous avons choisi une image en couleur (RGB) et une image en niveau de gris (Grey) de même résolution 512x512 pixels. Nous avons procédé à un prétraitement en deux étapes. La première étape est de convertir les données image en format de données array. La deuxième étape est de chiffrer et déchiffrer les données de l'image afin de représenter les trois images (originale, chiffrée et déchiffrée) des deux implantations matérielle et logicielle (Figures 4.11 et 4.12). Nous pouvons voir que l'image chiffrée dans les deux implantations est une image de couleur ressemblant à un bruit blanc, nous ne pouvons plus voir le contenu de l'image originale. Cependant, l'image déchiffrée est identique à l'image originale.

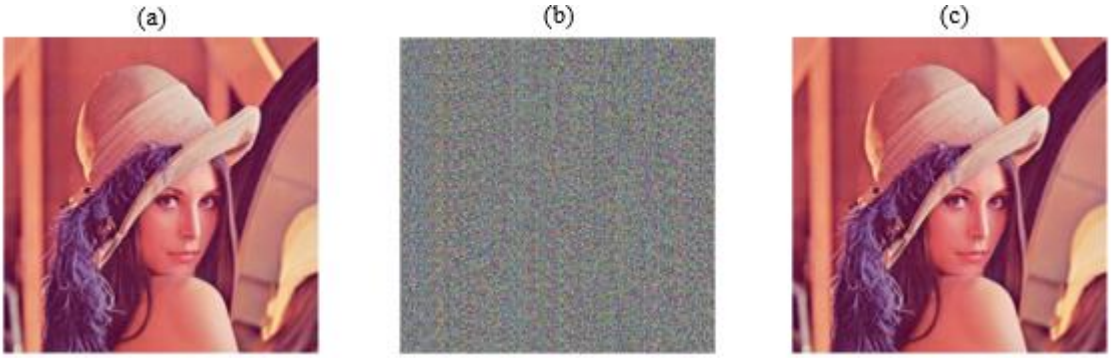

Figure 4.9 : Implantation matérielle de chiffrement/déchiffrement d'image couleur (RGB), (a) image originale, (b) image chiffrée, (c) image déchiffrée

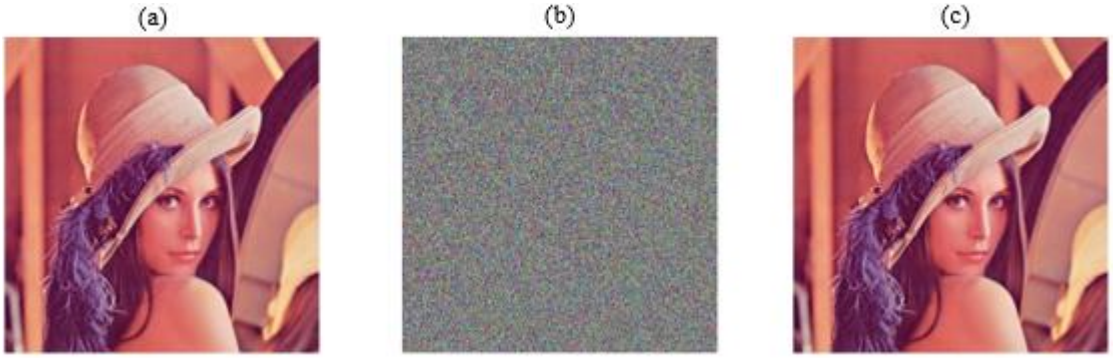

Figure 4.10 : Implantation purement logicielle de chiffrement/déchiffrement d'image couleur (RGB), (a) image originale, (b) image chiffrée, (c) image déchiffrée

Nous avons utilisé la bibliothèque « Numpy » pour calculer le coefficient de corrélation entre les trois images (originale, chiffrée et déchiffrée) des deux implantations matérielle et logicielle. Nous avons défini le même intervalle de chaque image de [0 : 1000] pixels, pour le quelle en calcul la corrélation entre les images.

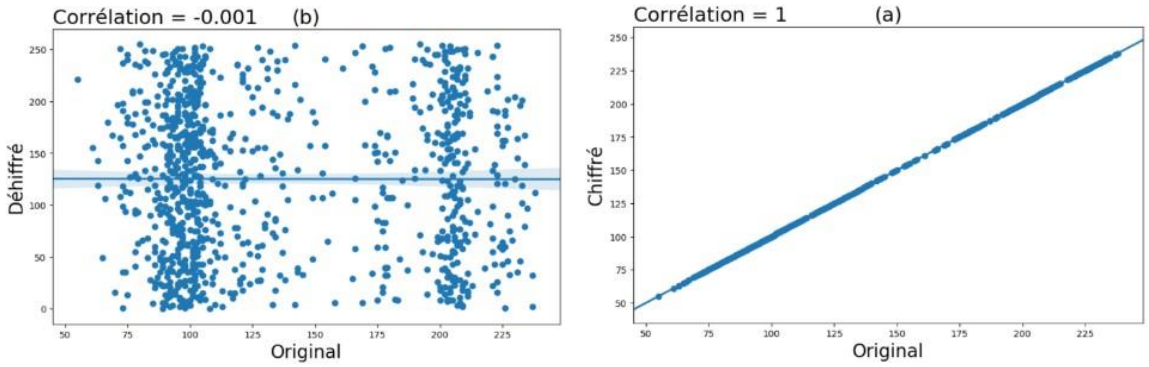

Figure 4.11 : Visualisation de la corrélation de l'implantation matérielle, (a) corrélation entre l'image originale et l'image chiffrée et (b) corrélation entre l'image originale et l'image déchiffrée.

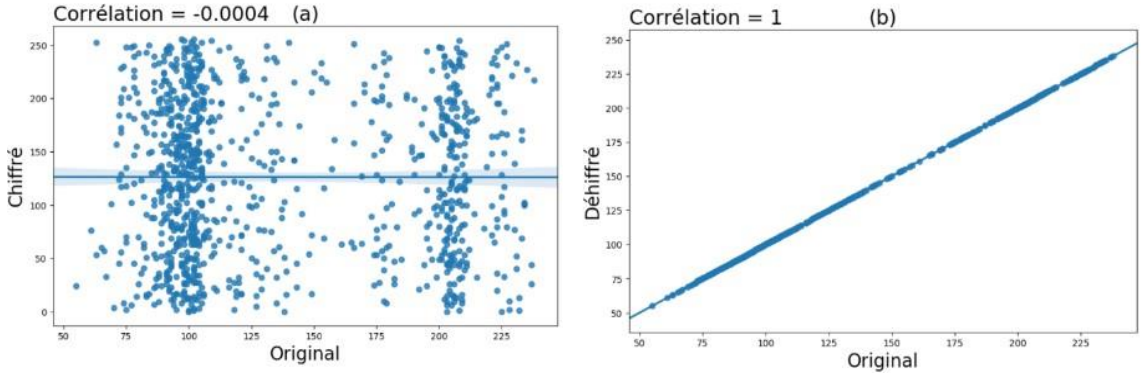

Figure 4.12 : Visualisation de la corrélation de l'implantation purement logicielle, (a) corrélation entre l'image originale et l'image chiffrée et (b) corrélation entre l'image originale et l'image déchiffrée.

Nous pouvons voir que sur .la partie (a) des deux Figures 4.13 et 4.14, le coefficient de corrélation tend vers 0 entre les deux images originales et chiffrées. Cela signifie que les deux images n'ont pas de similarité en aucune façon. Cependant, sur la partie (b) des deux

Figures 4.13 et 4.14, le coefficient de corrélation est de 1, entre les deux images originales et déchiffrées. Cela signifie que les deux images ont une similarité parfaite.

Il convient de noter que nous avons utilisé la bibliothèque PIL (Python Imaging Library) pour convertir l'image couleur en une image en niveaux de gris. Les Figures 4.15 et 4.16 représentent les trois images en niveau de gris (originale, chiffrée et déchiffrée) des deux implantations matérielle et logicielle. Nous pouvons voir que l'image chiffrée dans les deux implantations est une image niveau de gris de forme bruit blanc. Le contenu original n'est plus identifiable. Cependant, l'image déchiffrée est identique à l'image originale.

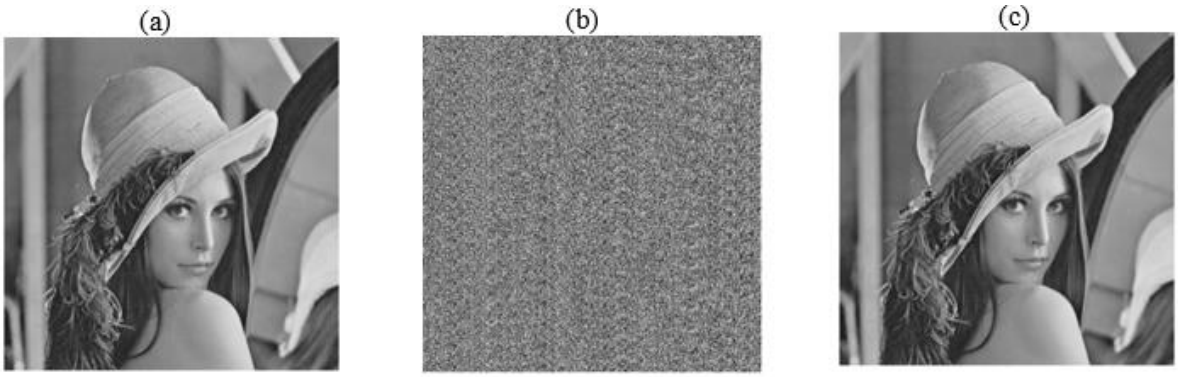

Figure 4.13 : Implantation matérielle de chiffrement/déchiffrement d'image niveau de gris (GREY), (a) image originale, (b) image chiffrée, (c) image déchiffrée

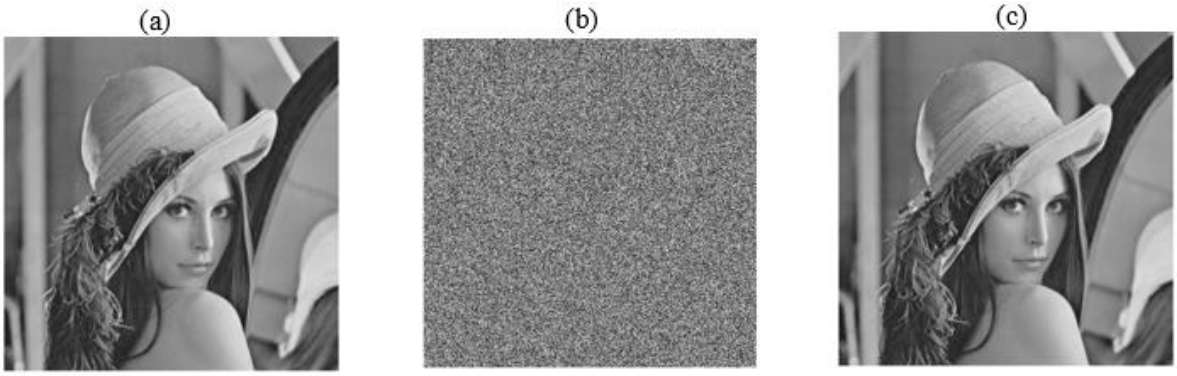

Figure 4.14 : Implantation purement logicielle de chiffrement/déchiffrement d'image niveau de gris (GREY), (a) image originale, (b) image chiffrée, (c) image déchiffrée

Pour vérifier la similarité entre les trois images, nous avons calculé la corrélation entre les trois images soit image originale, image chiffrée et image déchiffrée des deux implantations matérielle et logicielle. Le coefficient de corrélation entre l'image originale et l'image chiffrée de la partie (a) des deux Figures 4.17 et 4.18 tend vers 0, cela signifie que les deux images n'ont pas de similarité en aucune façon. Le coefficient de corrélation entre l'image originale et l'image déchiffrée de la partie (b) des deux Figures 4.17 et 4.18 est égale à 1, cela signifie qu'elles ont une similarité parfaite.

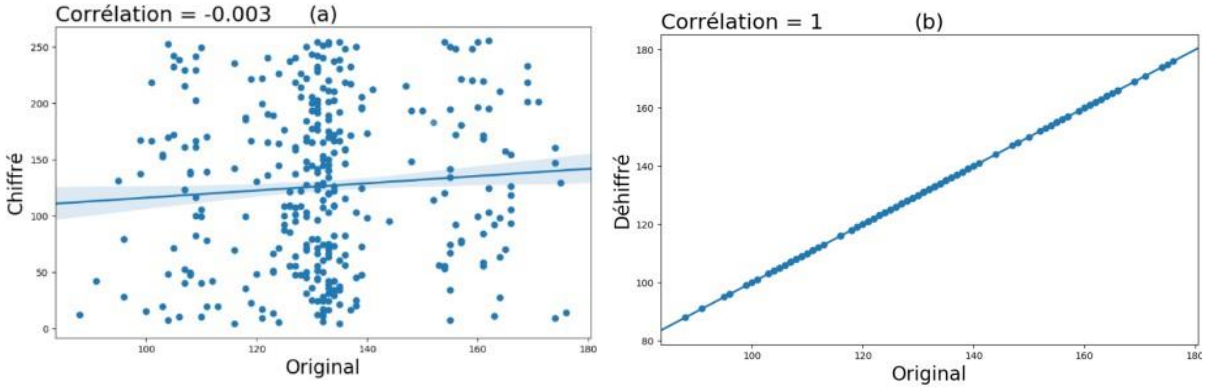

Figure 4.15 : Visualisation de la corrélation de l'implantation matérielle, (a) corrélation entre l'image originale et l'image déchiffrée et (b) corrélation entre l'image originale et l'image chiffrée.

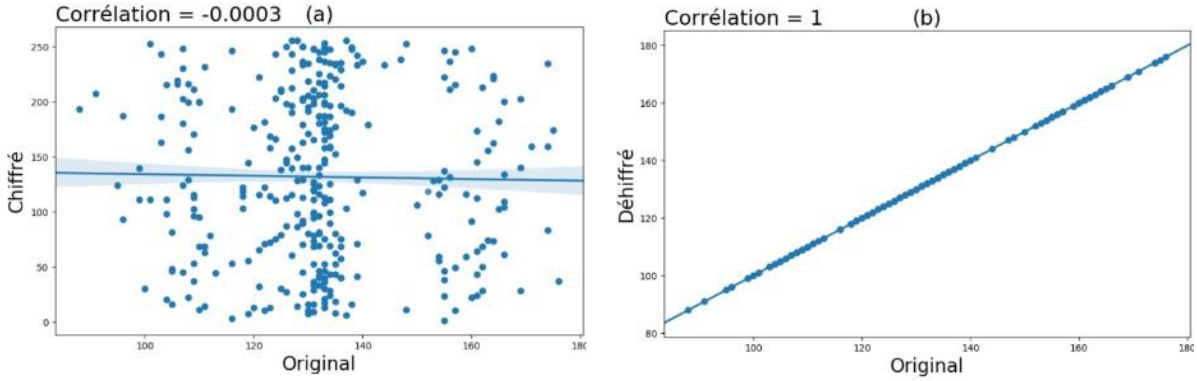

Figure 4.16 : Visualisation de la corrélation de l'implantation purement logicielle, (a) corrélation entre l'image originale et l'image déchiffrée et (b) corrélation entre l'image originale et l'image chiffré.

4.4.3 Résultats de temps d'exécution de chiffrement et de déchiffrement de données audio et image

Les implantations matérielle et logicielle sur la carte PYNQ-Z2 de l'algorithme de chiffrement/déchiffrement AES des deux types de données d'image et d'audio ont été exécutées sur Jupyter Notebook. De plus, les résultats du temps d'exécution de notre architecture matérielle comparée à l'implantation logicielle du signal audio et des deux images, soient celle en couleur et en niveau de gris sont démontrés dans le Tableau 4.2.

Le choix des types de données utilisées pour l'implantation de l'algorithme AES est basé sur le besoin de traiter les différents types de données tel que l'image et l'audio en les chiffrant et en les déchiffrant à l'aide des implantations matérielle et logicielle. Il convient de noter que nous avons utilisé la bibliothèque PIL (Python Imaging Library) pour convertir l'image couleur en une image en niveaux de gris.

Tableau 4.2 : Comparaison du temps d'exécution de chiffrement et de déchiffrement

Types de données	**Opérations**	**Temps d'exécution**	
		Jupyter Notebooks (Architecture proposée)	**Jupyter Notebooks (Software)**
Audio	Chiffrement	4.62 ms	13.60 ms
	Déchiffrement	4.87 ms	13.01 ms
Image niveau de gris (Gray)	Chiffrement	10.04 ms	28.79 ms
	Déchiffrement	10.09 ms	28.43 ms
Image couleur (RGB)	Chiffrement	35.82 ms	80.18 ms
	Déchiffrement	35.76 ms	80.20 ms

Le Tableau 4.2 montre donc que les temps d'exécution du chiffrement et du déchiffrement des données audio par l'architecture matérielle proposée sont réduits à 4.62 ms et 4.87 ms par rapport aux 13.60 ms et 13.01 ms du chiffrement et déchiffrement par

l'implantation logicielle. De même, l'architecture proposée est plus rapide que l'implantation logicielle de l'image en niveau de gris et de l'image en couleur en termes de temps d'exécution. Cela démontre que notre architecture est environ trois fois plus rapide et plus efficace que l'implantation logicielle de l'algorithme AES pour le chiffrement et déchiffrement du signal audio et des images.

Nous pouvons aussi voir que le temps de chiffrement d'une image en couleur prend entre trois à quatre fois le temps de chiffrement d'une image en niveau de gris dans les deux exécutions logicielle et matérielle. Cela est dû au fait que l'algorithme AES traite chaque pixel en trois couleurs (R, G, B) séparément. De ce fait, il faut plus de temps pour traiter une image en couleur qu'une image en niveau de gris. La taille de fichier est un autre paramètre qui influence le temps d'exécution. En effet, plus la taille de fichier est importante, plus le temps d'exécution est long.

Comme nous pouvons voir que le chiffrement d'une image en niveau de gris prend à peu près le double de temps que le chiffrement de l'audio, cela est dû au fait que la taille de l'image est beaucoup plus importante. De même, le chiffrement d'une image en couleur par rapport au chiffrement d'une image en niveau de gris prend plus de temps, car la taille d'une image en couleur est trois fois plus grande que la taille d'une image en niveau de gris.

4.5 COMPARAISON DE L'IMPLANTATION FPGA DU CHIFFREMENT AES

La fréquence maximale de notre implantation est calculée par le biais de la période d'horloge cible T, qui est de 10 ns et la pire marge négative de l'horloge cible WNS (Worst Negative Slack), qui est de 0.862 ns. Le débit de chiffrement et de déchiffrement AES est calculé à partir de la fréquence maximale de système et la taille du bloc B, qui est égale à 128 bits dans notre cas. La fréquence maximale et le débit sont définis de la manière suivante (Zhai *et al.*, 2017) :

$$F_{MAX} = \frac{1000}{T - WNS}\ MHz \tag{4.2}$$

$$Débit = B \times F_{MAX}\ Gbps \quad (4.3)$$

Les travaux publiés récemment dans la littérature et le présent travail diffèrent au niveau de la carte FPGA utilisée. Le résultat d'implantation du système proposé de chiffrement et de déchiffrement AES-128 bits, a montré que notre système consomme une puissance seulement de 43 mW et il occupe 3206 slices, ce qui conduit à une architecture efficace en termes de surface et de puissance. Il fonctionne à une fréquence maximale de 109.43 MHz, produisant un débit de 14 Gbps. L'architecture que nous proposons peut être utilisée efficacement dans les applications biomédicales embarquées à faible consommation.

Nous pouvons observer dans le Tableau 4.3 que les architectures précédentes de Zodpe & Sapkal (2020) et Murugan *et al*. (2020) ont un débit par Slice et une fréquence de fonctionnement plus élevés par rapport à l'architecture AES proposée, mais leurs consommations d'énergie sont plus élevées, ce qui les rend moins adaptées aux applications biomédicales embarquées à faible consommation.

Tableau 4.3 : Résultats obtenus avec d'autres implantations matérielles

Travaux	Dispositif	Largeur de bloc (bits)	Nombre de Slice	Fréquence maximale (MHz)	Débit (Gbps)	Mbps/slice	Puissance (mW)
(Zodpe & Sapkal, 2020)	Spartan-6 XC6SLX150	128	5566	237.45	30.3	5.4	/
(Murugan et al., 2020)	Spartan-3 XC300s	128	1132	67.75	8.672	7.66	/
(Vliegen *et al*., 2017)	Virtex-7 X485T	128	38241	119	15.24	0.398	/
(Murugan et al., 2020)	Virtex-4 XC4VLX200	128	1120	112.37	14.38	12.84	261
(Zhai *et al*., 2017)	XC7Z020	128	372	50	5.1 Mbps	13.73 Kbps	107
Architecture proposée	**XC7Z020-CLG400-1**	**128**	**3206**	**109.43**	**14**	**4.36**	**43**

CONCLUSION GÉNÉRALE

Dans ce mémoire, nous avons vu que la sécurité des données constitue une préoccupation de plus en plus importante dans le domaine de la santé. Pour assurer cette sécurité, il est important de choisir les bonnes techniques de cryptographie et de les implanter correctement. De plus, nous avons passé en revue les différents travaux qui ont été menés dans ce domaine et les différentes techniques qui ont été mises en place pour améliorer la sécurité des données. Nous avons également étudié les circuits FPGA et leurs principales caractéristiques, ce qui nous démontre qu'ils peuvent être une solution efficace pour l'optimisation de l'implantation de système de chiffrement avancé (AES).

L'AES est l'algorithme de chiffrement le plus utilisé dans le monde. Il est largement reconnu pour sa fiabilité et sa sécurité. L'utilisation des circuits FPGA pour son implantation offre de nombreux avantages, notamment en termes de vitesse et de consommation électrique. Cependant, les circuits FPGA ont aussi quelques inconvénients, comme leur prix élevé et leur complexité de programmation. Malgré ces inconvénients, l'utilisation des circuits FPGA pour l'implantation AES est une solution intéressante à prendre en considération pour assurer la sécurité des données afin de garantir la confidentialité et l'intégrité de celle-ci.

Nous avons présenté une architecture FPGA pour le chiffrement et le déchiffrement AES-128 bits qui utilise le mode compteur (CTR). Elle est capable de chiffrer des données telles que le signal ECG, l'audio et l'image, et ne peut être déchiffrée qu'avec la clé secrète appropriée. Ce système a été implanté sur une carte FPGA PYNQ-Z2, avec deux approches de différents niveaux d'abstraction (bas-niveau et haut-niveau).

Pour l'implantation des processus de chiffrement et de déchiffrement des données de 128 bits, les résultats présentés ont démontré que l'architecture proposée ne nécessite que 13% des ressources matérielles du circuit ZYNQ de la carte PYNQ-Z2 et 46 mW en termes de consommation de puissance, avec une fréquence maximale de 109.43 MHz produisant un débit de 14 Gbps. Au niveau du temps d'exécution, nos différents tests ont démontré que le

chiffrement de signal ECG (ElectroCardioGram) en format (*.csv) est deux fois plus rapide que le chiffrement de signal ECG en format (*.txt), avec un temps de chiffrement et de déchiffrement respectivement de 3.18 ms et 3.26 ms. Ce qui implique que notre architecture proposée nécessite des ressources et du temps de calcul minimal, ce qui les rend idéales pour les systèmes embarqués dont les ressources et la puissance sont limitées. De plus, notre système prendra en charge d'autres types de données numériques tels que les signaux audio et les images, avec un temps de traitement très faible par rapport à l'implantation logicielle. Ce qui démontre l'efficacité et la performance de notre architecture proposée.

Nos résultats ont aussi démontré que l'approche bas-niveau (langage C sous Vitis) est relativement plus performante que l'approche haut-niveau (langage Python sous Jupyter). Cependant, la plateforme Jupyter notebook est gratuite (open source), elle fournit une meilleure interface graphique que Vitis IDE. Elle ne nécessite aucune installation de l'outil Xilinx et permet une mise en application assez simple. Elle nous donne aussi la possibilité de basculer entre plusieurs applications différentes. De plus, lors d'une modification et l'exécution d'un algorithme, il n'est pas nécessaire de reconstruite à chaque reprise le boot image et le copier sur la carte SD. Ce qui le rend beaucoup moins complexe que Vitis IDE. En revanche, il est important de noter que la plateforme Vitis a des fonctionnalités de débogage plus avancées que Jupyter notebook. Ce qui rend Vitis IDE plus adapté pour des projets plus complexes et professionnels.

Comme perspective, il est possible d'ajouter un système d'identification de données à notre architecture et d'implanter d'autres modes de chiffrement AES tels que le mode CBC avec différentes langueurs de clé, soit 128 bits, 192 bits et 256 bits. De plus, il est possible de tester cette architecture avec d'autres cartes FPGA comme la carte ZYNQ-UltraScale MPSoc qui dispose de processeurs plus performants. Enfin, il est possible de tester cette architecture avec d'autres types de données numériques telle que la vidéo.

RÉFÉRENCES BIBLIOGRAPHIQUES

Aleisa, N. (2015). A comparison of the 3DES and AES encryption standards A Comparison of the 3DES and AES Encryption Standards. *International Journal of Security and Its Applications*, *Vol.9*(February), pp.241-246. https://doi.org/10.14257/ijsia.2015.9.7.21

Alharam, A. K., & El-Madany, W. (2017). Complexity of cyber security architecture for IoT healthcare industry: A comparative study. *Proceedings - 2017 5th International Conference on Future Internet of Things and Cloud Workshops, W-FiCloud 2017*, *2017-Janua*, 246–250. https://doi.org/10.1109/FiCloudW.2017.100

Altaf, M., & Mane, P. P. B. (2017). Watermarking and Cryptography Based Image Authentication on Watermarking and Cryptography Based Image Authentication on Reconfigurable Platform. *Bulletin of Electrical Engineering and Informatics*, *Vol. 6*(June), 181–187. https://doi.org/10.11591/eei.v6i2.651

Amira, A., Saghir, M. A. R., Ramzan, N., Grecos, C., & Scherb, F. (2013). A reconfigurable wireless environment for ecg monitoring and encryption. *International Journal of Embedded and Real-Time Communication Systems*, *Vol. 4*(3), 72–87. https://doi.org/10.4018/ijertcs.2013070104

Arundhati, J., Dakhole, P. K., & Thatere, A. (2015). Implementation of S-Box for Advanced Encryption Standard. *2015 IEEE International Conference on Engineering and Technology (ICETECH), 20th March 2015, Coimbatore, TN, India.*, *March*. https://doi.org/10.1109/ICETECH.2015.7275043

Bailey, D. G. (2011). Design for Embedded Image Processing on FPGAs. In *IEEE* (1st editio). Wiley-IEEE Press. https://doi.org/10.1002/9780470828519

Bartz, G., & Markey, B. (2012). *La cryptographie à travers l'histoire : objet mathématique et arme politique Cryptography through history: mathematical object and political weapon*. https://docplayer.fr/25581482-La-cryptographie-a-travers-l-histoire-objet-

mathematique-et-arme-politique-cryptography-through-history-mathematical-object-and-political-weapon.html

Bénony, V. (2006). *Étude et conception de systèmes de chiffrement à flot dans le contexte d'architectures matérielles fortement contraintes* (Vol. 1) [Université de Lile (France)]. http://www.theses.fr/2006LIL10040/document

Bergeron, F., & Goupil, A. (2014). *La cryptographie de l ' Antiquit à l ' Internet*. http://bergeron.math.uqam.ca/wp-content/uploads/2014/04/crypto.pdf

Bhushan, S., Kumar, M., Kumar, P., Ravi, R. V., & Kumar Singh, A. (2022). *Holistic Approach to Quantum Cryptography in Cyber Security* (1st ed.). CRC Press. https://doi.org/10.1201/9781003296034

Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D., & Shamir, A. (2010). Key Recovery Attacks of Practical Complexity on AES-256 Variants with up to 10 Rounds. In H. Gilbert (Ed.), *29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010, Proceedings* (pp. 299–319). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-13190-5_15

Blanc, J., & De Georges, A. (2004). *Techniques de cryptographie*. http://deptinfo.unice.fr/twiki/pub/Linfo/PlanningDesSoutenances20032004/blanc-degeorges.pdf

Blazhevski, D., Bozhinovski, A., Stojcevska, B., & Pachovski, V. (2013). Modes of operation of the aes algorithm. *The 10th Conference for Informatics and Information Technology (CIIT 2013), April*. https://www.researchgate.net/publication/236656798%0AMODES

Boura, C. (2012). Analyse de fonctions de hachage cryptographiques [University Pierre et Marie Curie-Paris VI (France)]. In *Thèse de doctorat*. https://theses.hal.science/tel-00767028

Brisson, R., & Théberge, F. (2000). *Un aperçu de l ' histoire de la cryptologie*. 1–20. https://www.apprendre-en-ligne.net/crypto/bibliotheque/PDF/brisson.pdf

Crockett, L. H., Elliot, R. A., Enderwitz, M. A., & Stewart, R. W. (2014). *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc* (1st ed.). Strathclyde Academic Media. https://is.muni.cz/el/1433/jaro2015/PV191/um/The_Zynq_Book_ebook.pdf

Daemen, J., & Rijmen, V. (2000). The Block Cipher Rijndael. In *Springer* (Vol. 1820). Springer, Berlin, Heidelberg. https://doi.org/10.1007/10721064_26

Daemen, J., & Rijmen, V. (2020). The advanced encryption standard process. In *Springer*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-60769-5_1

Daoud, L., Hussein, F., & Rafla, N. (2019). Optimization of advanced encryption standard (AES) using vivado high level synthesis (HLS). *Proceedings of 34th International Conference on Computers and Their Applications, CATA 2019*, *58*, 36–44. https://doi.org/10.29007/x3tx

Deschamps, J. P., Bioul, G. J. A., & Sutter, G. D. (2006). *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*. A JOHN WILEY & SONS, INC. Hoboken, New Jersey. https://doi.org/10.1002/0471741426

DFRobot. (2022). *PYNQ-Z2 Development Board*. https://www.mouser.ca/new/dfrobot/dfrobot-pynqz2-dev-board/#RelatedContent-6

Dombkowski, K. E., & Kocan, K. F. (2004). FPGA technology to minimize extended life-cycle development. *Bell Labs Technical Journal*, *9*(1), 191–195. https://doi.org/10.1002/bltj.20013

Dumas, J., Roch, J., Tannier, E., & Varrette, S. (2007). *Théorie des codes : compression, cryptage, correction* (1ère éditi). Dunod, Paris.

Dworkin, M., Barker, E., Nechvatal, J., Foti, J., Bassham, L., Roback, E., & Dray, J. (2001).

Advanced Encryption Standard (AES). Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD. https://doi.org/https://doi.org/10.6028/NIST.FIPS.197

Fan, C., & Hwang, J. (2007). Implementations of High Throughput Sequential and Fully Pipelined AES Processors on FPGA. *Proceedings of 2007 International Symposium on Intelligent Signal Processing and Communication Systems Nov.28-Dec.1, 2007 Xiamen, China Implementations*, 353–356. https://doi.org/10.1109/ISPACS.2007.4445896

Gabrick, M., Nicholson, R., Winters, F., Young, B., & Patton, J. (2006). FPGA Considerations for Automotive Applications. *SAE 2006 World Congress & Exhibition*. https://doi.org/10.4271/2006-01-0368

Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet. *Circulation*, *101*(23), e215–e220. https://doi.org/10.1161/01.CIR.101.23.e215

Good, T., & Benaissa, M. (2007). Pipelined AES on FPGA with support for feedback modes (in a multi-channel environment). *IEEE*, *1*, 1–10. https://doi.org/10.1049/iet-ifs

Hameed, M. E., Ibrahim, M. M., & Manap, N. A. (2019). Compression and encryption for ECG biomedical signal in healthcare system. *Telkomnika (Telecommunication Computing Electronics and Control)*, *17*(6), 2826–2833. https://doi.org/10.12928/TELKOMNIKA.v17i6.13240

Hameed, M. E., Ibrahim, M. M., Manap, N. A., & Mohammed, A. A. (2020). An enhanced lossless compression with cryptography hybrid mechanism for ECG biomedical signal monitoring. *International Journal of Electrical and Computer Engineering*, *10*(3), 3235–3243. https://doi.org/10.11591/ijece.v10i3.pp3235-3243

Harvey, M. (2016). *Fast AES-128 CTR mode encryption/decryption*. http://www.markharvey.info/des/aes128_40G/aes128_40G.html

Henry-Labordère, A. (2021). *Cryptographie classique - De la préparation du concours Alkindi jusqu'aux épreuves du Bac* (Ellipses (Ed.)).

Kee, H., Ly, T., Aziz, A., & Spasojevic, P. (2017). FPGA-Based Channel Coding Architectures for 5G Wireless Using High-Level Synthesis. *International Journal of Reconfigurable Computing*, 23. https://doi.org/https://doi.org/10.1155/2017/3689308

Lai, X., & Massey, J. L. (1991). A proposal for a new block encryption standard. *Springer*, *473 LNCS*, 389–404. https://doi.org/10.1007/3-540-46877-3_35

Li, H., & Li, J. (2005). A High Performance Sub-Pipelined Architecture for AES. *International Conference on Computer Design*, 491–496. https://doi.org/10.1109/ICCD.2005.10

Liu, B., & Baas, B. M. (2013). Parallel AES Encryption Engines for Many-Core Processor Arrays. *In IEEE Transactions on Computers*, *62*(3), 536–547. https://doi.org/10.1109/TC.2011.251.

Manoj, K. T., & Karthigaikumar, P. (2018). FPGA implementation of an optimized key expansion module of AES algorithm for secure transmission of personal ECG signals. *Springer*, *22*(1–2), 13–24. https://doi.org/10.1007/s10617-017-9189-5

Maqsood, F., Ahmed, M., Mumtaz, M., & Ali, M. (2017). Cryptography: A Comparative Analysis for Modern Techniques. *International Journal of Advanced Computer Science and Applications*, *8*(6), 442–448. https://doi.org/10.14569/ijacsa.2017.080659

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Crytography. In *CRC Press, Boca Raton* (1st editio, Vol. 106, Issue 1).

Miao, F., Jiang, L., Li, Y., & Zhang, Y. T. (2009). A novel biometrics based security solution for body sensor networks. *Proceedings of the 2009 2nd International Conference on Biomedical Engineering and Informatics, BMEI 2009*. https://doi.org/10.1109/BMEI.2009.5304950

Miaou, S. G., Chen, S. T., & Lin, C. L. (2002). An integration design of compression and encryption for biomedical signals. *Journal of Medical and Biological Engineering*, *22*(4), 183–192.

Ministère du Travail de l'Emploi et de la Solidarité Sociale. (2022). *M-9, r. 17 - Code de déontologie des médecins*. https://www.legisquebec.gouv.qc.ca/fr/document/rc/M-9, r. 17

Mohan, H. S., Reddy, A., & Manjunath, T. N. (2011). Improving the Diffusion power of AES Rijndael with key multiplication. *International Journal of Computer Applications*, *30*, 39–44. https://doi.org/10.5120/3635-5076

Mohanty, S. P. (2009). A secure digital camera architecture for integrated real-time digital rights management. *Journal of Systems Architecture*, *55*(10–12), 468–480. https://doi.org/10.1016/j.sysarc.2009.09.005

Mondal, R., Ngo, H., Shey, J., Rakvic, R., Walker, O., & Brown, D. (2020). Efficient Architecture Design for the AES-128 Algorithm on Embedded Systems. *Proceedings of the 17th ACM International Conference on Computing Frontiers*, 89–97. https://doi.org/10.1145/3387902.3392624

Monmasson, E., & Cirstea, M. N. (2007). FPGA design methodology for industrial control systems - A review. *IEEE Transactions on Industrial Electronics*, *54*(4), 1824–1842. https://doi.org/10.1109/TIE.2007.898281

Moody, G. B., & Mark, R. G. (2001). The impact of the MIT-BIH arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, *20*(3), 45–50. https://doi.org/10.1109/51.932724

Murugan, C. A., Karthigaikumar, P., & Sathya Priya, S. (2020). FPGA implementation of hardware architecture with AES encryptor using sub-pipelined S-box techniques for compact applications. *Journal for Control, Measurement, Electronics, Computing and Communications*, *61*(4), 682–693. https://doi.org/10.1080/00051144.2020.1816388

Naya-Plasencia, M. (2009). *Chiffrements à flot et fonctions de hachage : conception et cryptanalyse.* Thèse de doctorat. (Vol. 1) [EDITE de Paris]. http://www.theses.fr/2009PA066526

OMS, O. M. de la S. (2020). *L'OMS lève le voile sur les principales causes de mortalité et d'incapacité dans le monde : 2000-2019*. https://www.who.int/fr/news/item/09-12-2020-who-reveals-leading-causes-of-death-and-disability-worldwide-2000-2019#:~:text=Les maladies cardiaques sont restées,millions de décès en 2019.

PhysioNet, T. (2017). *Frequently Asked Questions about PhysioNet*. PhysioNet. https://archive.physionet.org/faq.shtml#signal-default-names

Priya, S. S. S., Karthigaikumar, P., Sivamangai, N. M., & Rejula, V. (2017). High Throughput AES Algorithm Using Parallel Subbytes and MixColumn. *Springer*, *95*(2), 1433–1449. https://doi.org/10.1007/s11277-016-3858-8

Priya, S. S. S., Karthigaikumar, P., & Teja, N. R. (2021). FPGA implementation of AES algorithm for high speed applications. *Springer*, *0123456789*. https://doi.org/10.1007/s10470-021-01959-z

Rahimunnisa, K., Karthigaikumar, P., Christy, N. A., Kumar, S. S., & Jayakumar, J. (2013). PSP : Parallel sub-pipelined architecture for high throughput AES on FPGA and ASIC. *Central European Journal of Computer Science*, *3*(4), 173–186. https://doi.org/10.2478/s13537-013-0112-2

Rahimunnisa, K., Kumar, K., Rasheed, S., Jayaraj, J., & Sureshkumar, S. (2014). FPGA implementation of AES algorithm for high throughput using folded parallel architecture. *Wiley Online Library*, *October 2012*, 2225–2236. https://doi.org/10.1002/sec.651

Rakanovic, D., & Struharik, R. (2017). IP core for AES256 and TDES algorithms with AXI interface. *24th Telecommunications Forum, TELFOR 2016*. https://doi.org/10.1109/TELFOR.2016.7818860

Reddy, A. (2012). Revised aes and its modes of operation. *International Journal of Information Technology & Knowledge Management*, *5*(June), 31.

Ricci, F. (2002). *Conception d'une plateforme de prototypage rapide basée sur un FPGA pour le développement d'algorithmes de contrôle de moteurs.* Thèse (M. Sc.) Université Laval.

Rivest, R. L., Shamir, A., & Adleman, L. M. (1978). A method for obtaining digital signatures and public key cryptosystems. *Secure Communications and Asymmetric Cryptosystems*, 217–239.

RkBlog. (2020). *Overview of PYNQ project offering FPGA capabilities to Python and data engineers.* RkBlog. https://rk.edu.pl/en/overview-pynq-project-offering-fpga-capabilities-python-and-data-engineers/

Rodriguez-Andina, J. J., Moure, M. J., & Valdes, M. D. (2007). Features, design tools, and application domains of FPGAs. *IEEE Transactions on Industrial Electronics*, *54*(4), 1810–1823. https://doi.org/10.1109/TIE.2007.898279

Soltani, A., & Sharifian, S. (2015). Microprocessors and Microsystems An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA. *Elsevier*, *39*(7), 480–493. https://doi.org/10.1016/j.micpro.2015.07.005

Thawte. (2013). *Histoire du chiffrement et de ses méthodes*. https://www.thawte.fr/assets/documents/guides/history-cryptography.pdf

Visconti, P., Velazquez, R., Capoccia, S., & Fazio, R. De. (2021). High-performance AES-128 algorithm implementation by FPGA-based SoC for 5G communications. *International Journal of Electrical and Computer Engineering (IJECE)*, *11*(5), 4221–4232. https://doi.org/10.11591/ijece.v11i5.pp4221-4232

Vliegen, J., Reparaz, O., & Mentens, N. (2017). Maximizing the throughput of threshold-protected AES-GCM implementations on FPGA. *IEEE*, 140–145.

https://doi.org/10.1109/IVSW.2017.8031559

Xilinx. (2018). Vivado Design Suite User Guide. *Ug903*, *4*, 1–173. http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug903-vivado-using-constraints.pdf

Xilinx. (2020). Vitis High-Level Synthesis User Guide. *Ug1399*, *2*, 1–657.

Xilinx. (2021). PYNQ - Python productivity for Zynq - Board. *Pynq.Io*, 368. http://www.pynq.io/board.html

Xilinx. (2022a). Vitis Unified Software Platform Documentation Embedded Software Development. *Xilinx Technical Documentation*, *UG1400*, 667.

Xilinx. (2022b). Vitis Unified Software Platform Documentation Embedded Software Development. *Xilinx Technical Documentation*, *UG1393*, 667.

Xilinx Inc. (2018). Zynq-7000 SoC Data Sheet: Overview. *Xilinx*, *DS190*, 1–21. https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

Yang, X., Yi, X., Khalil, I., Fengling, H., & Tari, Z. (2016). Securing body sensor network with ECG. *ACM International Conference Proceeding Series*, 298–306. https://doi.org/10.1145/3007120.3007121

Zhai, X., Ait Si Ali, A., Amira, A., & Bensaali, F. (2017). ECG encryption and identification based security solution on the Zynq SoC for connected health systems. *Elsevier*, *106*, 143–152. https://doi.org/10.1016/j.jpdc.2016.12.016

Zodpe, H., & Sapkal, A. (2020). An efficient AES implementation using FPGA with enhanced security features. *Journal of King Saud University - Engineering Sciences*, *32*(2), 115–122. https://doi.org/10.1016/j.jksues.2018.07.002