REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj

Faculté *des Sciences et de la technologie*

Département d'électronique

Mémoire

Présenté pour obtenir

LE DIPLOME DE MASTER

FILIERE : ELECTRONIQUE

Spécialité : industries électroniques

Par

➢ **Mlle. MEHENNI AMIRA**

Intitulé

Etude et implémentation sur SoC-FPGA d'une méthode de cryptage symétrique

Soutenu le : 03 /07/2022.

Devant le Jury composé de :

Nom & Prénom	*Grade*	*Qualité*	*Etablissement*
Dr. F. KHALED	*MCB*	*Président*	*Univ-BBA*
Dr. S. AZOUG	*MCB*	*Encadreur*	*Univ-BBA*
Mme. F. HAMADACHE	*MAA*	*Examinateur*	*Univ-BBA*

Année Universitaire 2021/2022

Résumé

Dans ce mémoire, nous présentons l'étude et l'implémentation sur SoC-FPGA d'algorithmes de cryptage symétrique (DES, TDES et AES) en utilisant un langage de description matérielle HDL sur un SoC-FPGA Xilinx ZYNQ-7000. En programmant au plus bas niveau nous avons cherché à atteindre un bon compromis entre la vitesse et la surface occupée. Les programmes ont été simulés sur VIVADO et validés sous MATLAB. Les résultats d'implémentation de chaque algorithme ont été comparés les uns aux autres. Puis nous avons appliquer nos conceptions pour le chiffrement symétrique par bloc d'une image de tests de format standard niveau de gris (8 bits). Les résultats de l'implémentation et de comparaison en termes de performances ont été satisfaisantes.

Mots clés : Cryptage symétrique, DES, TDES, AES, Verilog, Soc-FPGA.

Abstract

In this work, we present the study and implementation on SoC-FPGA of symmetric encryption algorithms (DES, TDES and AES) using an HDL hardware description language on a Xilinx ZYNQ-7000 SoC-FPGA. By programming at the lowest level, we have sought to achieve a good compromise between speed and occupied surface. The programs were simulated on VIVADO and validated in MATLAB. The implementation results of each algorithm were compared to each other. Then we applied our designs for symmetric block encryption of a standard grayscale (8-bit) format test image. The results of the implementation and comparison in terms of performance were satisfactory.

Keywords: Symmetric cryptosystems, DES, TDES, AES, Verilog, Soc-FPGA.

ملخص

في هذا العمل ، نقدم دراسة لخوارزميات التشفير المتماثل (DES و TDES و AES) باستخدام لغة وصف أجهزة HDL على Xilinx ZYNQ-7000 SoC-FPGA. من خلال البرمجة على أدنى مستوى ، سعينا لتحقيق حل وسط جيد بين السرعة والسطح المشغول. تمت محاكاة البرامج على VIVADO والتحقق من صحتها في MATLAB. تمت مقارنة نتائج تنفيذ كل خوارزمية مع بعضها البعض. ثم طبقنا تصميماتنا لتشفير لصورة اختبار وكانت نتائج التنفيذ والمقارنة مرضية من حيث الأداء.

الكلمات المفتاحية: أنظمة التشفير المتماثل، DES، TDES، AES، Verilog، Soc-FPGA .

Remerciements

Avant tout, nous remercions Allah le tout puissant de nous avoir donné la patience, le courage et la volante pour pouvoir terminer ce travail.

Nous remercions chaleureusement notre encadreur Monsieur *Dr. AZOUG Seif Eddine* pour toutes les orientations et les conseille, il joué un rôle déterminant dans le progrès de notre travaille. Ce dernier est le résultat de sa collaboration qui a toujours été agréable et enrichissante. Qu'il trouve ici l'expression de notre profonde reconnaissance.

Ainsi que tous nos enseignent qui nous ont enseigné durant nos études au département de Electronique industriel. A nos familles *MEHENNI* et *NAIT HAMOUD*, notamment nos chers parents qui nous ont imbibés de soins, d'amour et d'affection et à nos frères *AREZKI* et *WALID* qui nous ont donnés un coup de main et ils ont contribué à ce que ce modeste travail. Nous tenons à remercier à *BOUKERCHE Zyad* et tous nos collègues d'étude,

Enfin, nous tenons aussi à exprimer notre profonde gratitude à toute personne de près ou de loin qui nous encourage.

Dédicaces

Je dédie ce modeste travail à :

A l'être le plus cher pour moi dans cette vie, ma mère AKILA et, à l'être que je respecte le plus dans ce monde, mon cher père AZIZ, pour tous les sacrifices et encouragement afin de finir ce travail. Je leur souhaite une vie pleine de joie, de santé et de bonheur. Que Dieu les garde et les entoure de ses bénédictions. A mes frères AREZKI et WALID et à toute ma famille élargie, à mes chers amis.

Ainsi qu'à toute personne qui m'est chère. Et à tous ceux qui m'ont aidé de près ou de loin à accomplir ce travail.

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

DES: Data Encryption Standard

AES: Advanced Encryption Standard

TDES: Triple Data Encryption Standard

ECB: Electronic Code Book

CBC: Cipher Block Chaining

CFB: Cipher Feedback

OFB: Output Feedback

XOR: Exclusive OR

IV: Initial Value

RC4: Rivest Cipher 4

PI: Permutation Initial

PF: Permutation Finale

IBM: International Business Machines

NIST: National Institute of Standard and Technology

S-Box: Substitution

PC1: Permuted Choice 1

PC2: Permuted Choice 2

FPGA: Field Programmable Gate Array

HDL: Hardware description language

CLB : Configurable Logic Bloc

IOB : Input Output Bloc

CPU : Central Processing Unit

ASIC: Application Specific Integrated Circuit

SoC : System on Chip

PL: Programmable Logic

PS: Processing System

FF: Flip flops

GPIO : General Purpose Input/ Output

DMA : Direct memory access

MMIO : Memory mapped input/output

LUT : Lookup-Table

IP : Intellectual Property

RTL : Register Transfer Level

RAM : Random Access Memory

SRAM : Static Random Access Memory

BRAM : Bloc Random Access Memory

ROM : Read Only Memory

Introduction Générale

Introduction Générale

Depuis l'invention de l'écriture, les humains ont exprimé le besoin de transmettre des informations de maniéré sécurisée, c'est-à-dire que même interceptées, les informations ne peuvent être comprises par l'ennemi. Cela a donné naissance à la cryptographie. La cryptographie est une science qui fournit un ensemble d'algorithmes pour assurer la confidentialité, l'authentification et l'intégrité des données. Pour assurer la confidentialité, on doit crypter ces données avec une ou plusieurs clés. Il existe deux types de chiffrement ou cryptage : chiffrement symétrique et chiffrement asymétrique.

En chiffrement ou cryptage symétrique une clé secrète (64 à 256 bits) est utilisée pour chiffrer et déchiffrer les données ce qui le rend rapide. D'autre part, en chiffrement asymétrique une paire de clé (1024 à 2048 bits) est utilisée : une clé publique pour le chiffrement et une clé privée pour le déchiffrement, ce qui peut entrainer des opérations arithmétiques longues.

Il existe plusieurs algorithmes symétriques parmi lesquels on retrouve l'algorithme de cryptage symétrique DES (Data Encryption Standard) qui utilise une clé secrète de 64 bits.et l'algorithme Triple DES, qui dépend des trois utilisations de l'algorithme DES (Chiffrement et déchiffrement) avec trois clés secrètes de 64 bits. On trouve également AES (Advanced Encryption Standard) qui utilise des clés secrètes de 128, 192 ou 256 bits.

En vue d'accélérer ces algorithmes, nous allons dans ce travail revoir en détail les opérations mathématiques utilisées par DES ,TDES et AES afin de pouvoir programmer une architecture de chiffrement /déchiffrement performante implantable sur FPGA.

Pour cela, nous avons utilisé une carte PYNQ-Z2 de Xilinx qui inclut un SoC-FPGA très performant de la famille ZYNQ7000 Ce dernier pourra supporter l'accélération du temps d'exécution des applications embarquées en réduisant le temps de chiffrement et de déchiffrement en maintenant un bon compromis entre la surface occupé sur le FPGA et le temps d'exécution. La vérification et validation des résultats a été effectué sur MATLAB.

Ce manuscrit est divisé en trois chapitres :

- **Chapitre 1 :** Traite les concepts de base à connaitre en cryptographie et en cryptage.
- **Chapitre 2 :** Etudie en détails les algorithmes de cryptage symétrique les plus connus (DES, Triple DES et AES) et leur principe de fonctionnement.
- **Chapitre 3:** Présente les résultats de simulation et d'implémentation des algorithmes AES, DES et Triple DES sur le SoC-FPGA de la carte PYNQ-Z2 avec une application en traitement d'image.

Chapitre 1 : Généralités sur le cryptage/décryptage

1.1. Introduction

L'information est un élément crucial. La confidentialité de l'information ne peut être assurée que par l'utilisation d'algorithmes de cryptage (ou de chiffrement) appartenant au domaine de la cryptographie.

Dans ce chapitre, nous allons revoir en premier le principe du cryptage/décryptage en cryptographie et les techniques de confusion/diffusion correspondantes. Ensuite, nous allons présenter en détail le concept du cryptage symétrique ainsi que les méthodes de cryptage par bloc sous ses différents modes.

1.2. Objectifs de la cryptographie

La cryptographie est un mot composé de deux mots de l'ancien grec « Kruptos » qui signifie « cacher » et « graphie » qui signifie « écrire ». Cela signifie littéralement, « cacher l'écriture » **[**1].

Les principaux objectifs de la cryptographie sont :

- **Confidentialité :** Protéger le contenu des données par l'utilisation d'algorithmes de cryptage.
- **Intégrité :** Assurer l'intégrité des données par l'utilisation d'algorithmes de hachage.
- **Authentification :** Garantir l'identité de l'émetteur/récepteur par l'échange de clés secrètes.
- **Non-répudiation :** Certifier l'échange des données par l'utilisation de signatures numériques.

Dans un article intitulé « Military Cipher » dans le journal de Science Militiers, Kerckhoff a énoncé certaines règles à respecter pour atteindre les objectifs souhaités : principes cryptographie, Son article comporte en réalité six principe. Connu depuis, sous le nom de principes de Kerckhoffs. On ici trois les plus utiles aujourd'hui [2] :

1. La sécurité repose sur le secret de la clé et non sur le secret de l'algorithme.
2. Le déchiffrement sans la clé doit être impossible (en temps raisonnable).
3. Trouver la clé à partir du clair et du chiffré est impossible (en temps raisonnable).

1.3. Principe du cryptage/décryptage

Le cryptage ou chiffrement (encryption en anglais) peut être défini comme une fonction mathématique $\boldsymbol{E_{Ke}(\ldots)}$ qui consiste à transformer un texte clair **T** en un texte crypté **C** à l'aide d'une clé secrète de cryptage **Ke** de sorte que :

$$\boldsymbol{E_{Ke}(T) = C.} \qquad (1.1)$$

Le décryptage ou déchiffrement (decryption en anglais) est une fonction mathématique inverse $\boldsymbol{D_{Kd}(\ldots)}$ qui permet de décrypter le texte crypté **C** et récupérer le texte en clair **T** original à l'aide d'une à l'aide d'une clé secrète de décryptage **Kd** [3] de sorte que :

$$\boldsymbol{D_{Kd}(C) = T.} \qquad (1.2)$$

Le but ici consiste à retrouver le texte en clair original à partir du texte chiffré seulement si **Kd = Ke** :

$$D_{Kd}(E_{Ke}(T)) = T \qquad (1.3)$$

Ce principe de cryptage et de décryptage est également illustré dans la figure 1.1.

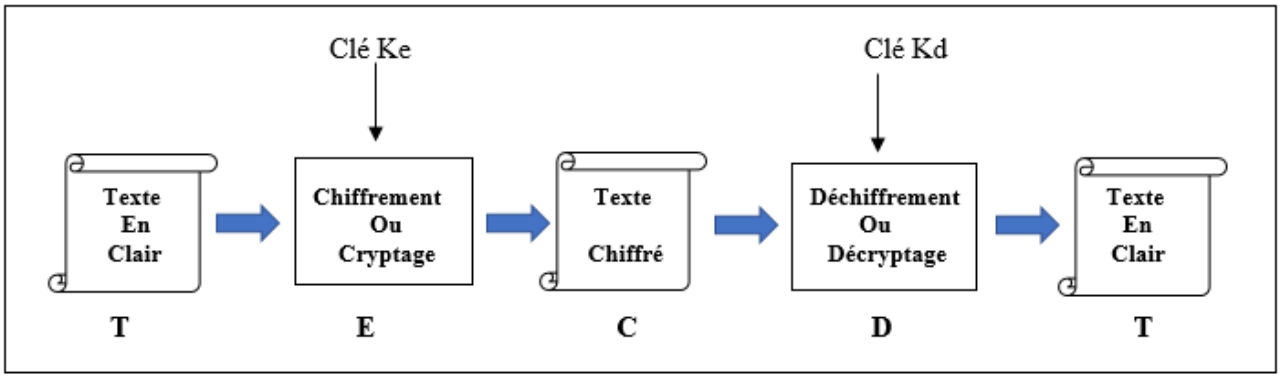

Figure 1.1. Principe du cryptage et décryptage

1.4. Concept de la confusion et de la diffusion

Selon le célèbre théoricien de l'information Claude Shannon, deux propriétés primitives, la confusion et la diffusion, permettent de rendre robuste la sécurité d'un algorithme de cryptage. Ces propriétés, lorsqu'elles sont présentes dans un algorithme de cryptage rendent difficile l'application d'attaques statistiques et d'autres méthode de cryptanalyse [2]. Ce concept n'est pas utilisé seulement en cryptage mais également dans la conception d'algorithmes de hachage et de générateurs de nombres pseudo-aléatoires où la décorrélation des valeurs générées est d'une importance primordiale [4].

1.4.1. Confusion (Substitution)

La confusion consiste à rendre complexe la relation clé de cryptage-texte chiffré par l'utilisation de techniques de substitution systématique. Le principe de la substitution consiste à remplacer dans un texte une ou plusieurs entités (généralement des lettres) par une ou plusieurs autres entités. [5]. En utilisant une fonction mathématique qui vise à rendre le texte aussi illisible que possible.

1.4.2. Diffusion (Permutation)

La diffusion consiste à rendre chaque élément du texte chiffre dépendant d'un nombre aussi grand que possible du reste des éléments du texte clair par l'utilisation de techniques de permutation. Le principe de la technique de permutation (aussi appelé transposition) consiste à change l'ordre des lettres du texte en clair de telle façon les rendre incompréhensibles. Avec le principe de permutation toutes les lettres du texte sont présentées, mais dans un ordre différent pour le rendre visuellement inutilisable, et c'est ce qu'on appelle la diffusion [1].

1.5. Méthodes de cryptage symétrique

Le cryptage symétrique ou cryptage a clé privée se base sur l'utilisation d'une seule clé privée pour le cryptage et pour le décryptage comme illustré sur la figure 1.2.

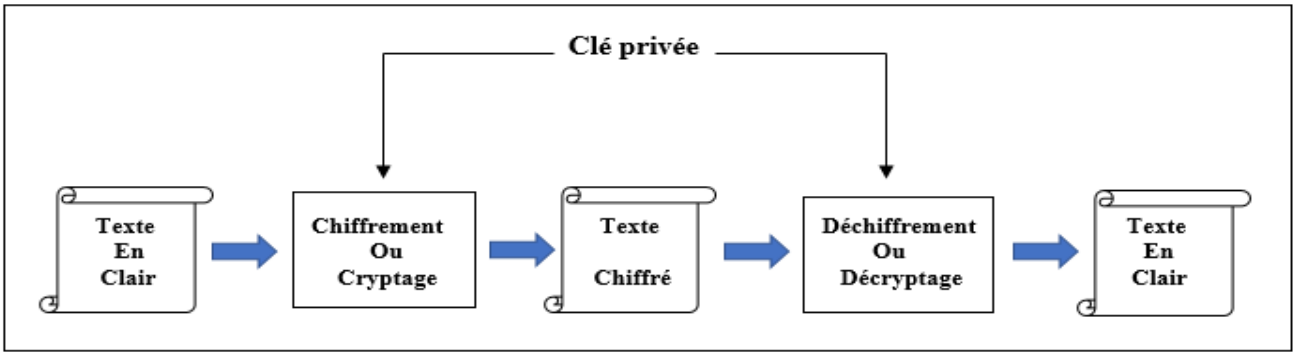

Figure 1.2. Principe du cryptage symétrique

Sans la clé privée, il est presque impossible de retrouver le texte original en clair (cela dépend du niveau de protection par cryptage utilisé ainsi que la complexité de la clé utilisée). L'émetteur doit transmettre la clé au récepteur via un canal sécurisé. Le principale désavantage des algorithmes de cryptage à clé symétrique est la sécurité du canal utilisé pour l'échange de clé. Cependant, les systèmes symétriques restent très efficaces, sécurisés, rapides et peuvent chiffrer et déchiffrer de grandes quantités de données en un temps record.

Les méthodes de cryptage symétrique peuvent être classé en deux catégories, le cryptage par bloc (bloc par bloc de bits) et le cryptage par flot (bit par bit) [6]. Les algorithmes de cryptage par bloc restent plus rapides que ceux par flot du fait de la possibilité d'appliquer un parallélisme aux blocs ce qui améliore grandement la vitesse d'exécution.

Parmi les algorithmes standards les plus recommandés en cryptage symétrique on trouve Triple DES (Data Encryption Standard) et AES (Advanced Encryption standard)**.** Ces algorithmes utilisent un cryptage symétrique par bloc et ils sont basés sur le concept de la confusion et de la diffusion en cryptographie en faisant appel à des fonctions de substitution et de permutation étudiées afin d'assurer la sécurité contre des attaques de cryptanalyse.

Cependant, AES offre un plus grand niveau de protection que DES (Triple DES) en raison de la taille des clés et de ses fonctions de substitution et de permutation.

1.6. Méthodes de cryptage asymétrique

En 1976, Whitfield Diffe et Martin E. Hellman décrit la possibilité de développer un algorithme de cryptage basé sur deux clés différentes [7].

Le cryptage à clé asymétrique ou cryptage a clé publique consiste deux clés différentes (Ke ≠ Kd) mais mathématiquement liées, l'une privée pour décrypter le message, elle est gardée secrète et l'autre publique pour crypter le message elle est publique connu par tout le monde comme le montre la figure 1.3.

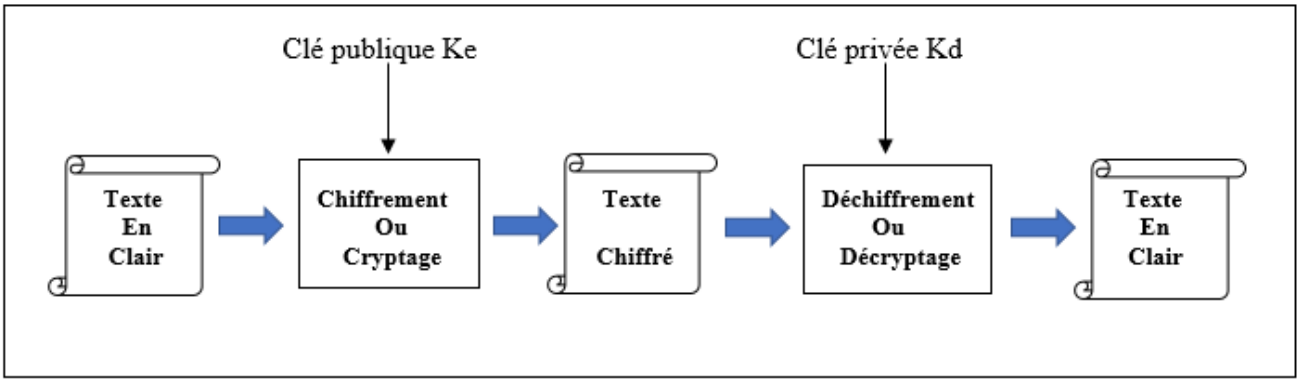

Figure 1.3. Principe du cryptage asymétrique

Pour bien comprendre le principe, on peut l'illustrer avec l'échange d'un message entre l'expéditeur et le destinateur, l'expéditeur possède deux clés, privé et publique. Il envoie son message contenant la clé publique au destinateur, le récepteur valide sa fiabilité et utilise la clé publique pour crypter son message, il envoie tout à l'expéditeur, l'expéditeur utilise sa clé privée pour décrypter le message [2].

Les cryptages asymétriques d'aujourd'hui sont beaucoup plus lents que les cryptages symétriques et nécessitent des clés plus longues. Leur principal avantage est la simplicité de la gestion des clés : les secrets ne sont pas partagés, les clés publiques peuvent être rendues publiques dans un annuaire, et lorsque plusieurs personnes veulent communiquer en privé, le nombre de clés est beaucoup plus réduit [1].

1.7. Cryptage symétrique par flot (Stream Cipher)

Les cryptages par flots fonctionnent généralement sur une dimension unitaire (bit par bit ou caractère par caractère). Le chiffrement par flux génère une série de bits sous forme de flux de clé à l'aide d'un générateur de nombre pseudo aléatoire (PRNG) comme le montre la figure 1.4.

D'après la figure 1.4, une clé secrète courte entraine une longue chaine de bits en combinant le flux clé-texte en clair. Habituellement, l'opération XOR bit à bit est choisie essentiellement pour sa simplicité à effectuer ce chiffrement.

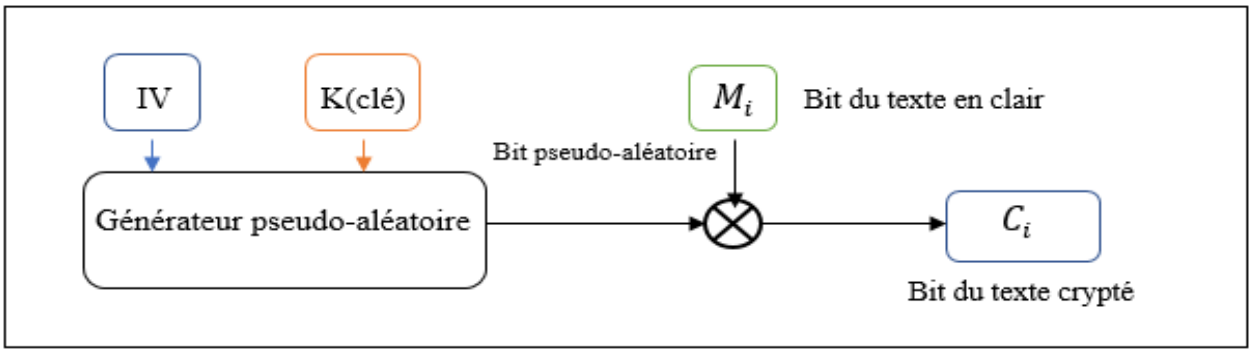

Figure 1.4. Principe de cryptage par flot

Leurs principaux avantages viennent du fait que la méthode de cryptage peut être changée à chaque symbole du texte en clair [2]. Quelques algorithmes de cryptage symétrique par flot [8] on trouve :

- **A5** dans les téléphones mobiles pour crypter les communications sans fil entre le téléphone mobile et l'antenne relais la plus proche [8].
- **RC4** largement déployé dans les protocoles de sécurité sans fil.

1.8. Cryptage symétrique par blocs (Block Cipher)

Contrairement aux chiffrements par flot qui fonctionnement bit par bit, dans le cryptage par bloc chaque texte en clair est découpé en blocs de longueur fixe (appelé la taille du bloc est de 64 ou 128) de mémé longueur et chiffré à l'aide d'une clé unique [2]. Ces algorithmes sont généralement construits sur un modèle itératif. Il utilise une fonction F qui prend une clé secrète k et un texte T de n bits. La fonction F est répétée un certain nombre de fois (appelé nombre de tours). A chaque tour, la clé k varie et le texte qui vient d'être obtenu de l'itération précédente est chiffré. Les différentes clés k[i] utilisées sont extraites de la clé secrète k [2]. Il existe quatre modes d'opération du cryptage par blocs :

- Mode ECB (Electronic Code Book)
- Mode CBC (Cipher Block Chaining)
- Mode OFB (Output Feedback)
- Mode CFB (Cipher Feedback)

1.8.1. Mode ECB

Le mode ECB (Electronic Code Book) ou dictionnaire de codes, est le mode opératoire le plus simple et rapide ce qui fait son principal avantage où chaque bloc de bits du message est chiffré indépendamment des autres et avec la même clé [8] comme le montre la figure 1.5.

L'inconvénient de ce mode est que si deux blocs du message sont identiques, cela se voit dans le texte chiffré. De plus, un attaquant actif peut chiffrer ou supprimer certains d'entre eux de telle sorte que le message modifié ait une signification différente de celle d'origine **[**3].

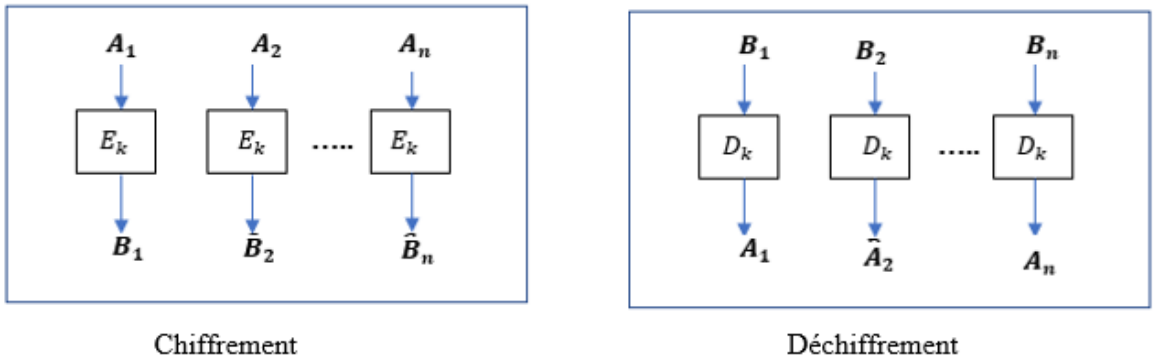

Figure 1.5. Principe du mode ECB

1.8.2. Mode CBC

Le mode CBC (Cipher block chaining) ou enchaînement des blocs. Comme le montre la figure 1.6, il consiste en un vecteur initial aléatoire (IV) qui est d'abord choisi pour être combiné avec le premier bloc (A_1) à l'aide d'une opération XOR avant le chiffrement. Une fois le premier bloc de texte en clair chiffrer par l'algorithme de chiffrement (E_K), le texte chiffré résultant (B_1) est stocké dans un registre de rétroaction afin qu'il puisse être combiné comme un vecteur initial avec le deuxième bloc de texte (A_2) avant le chiffrement [9]. Comme illustré sur la figure 1.6.

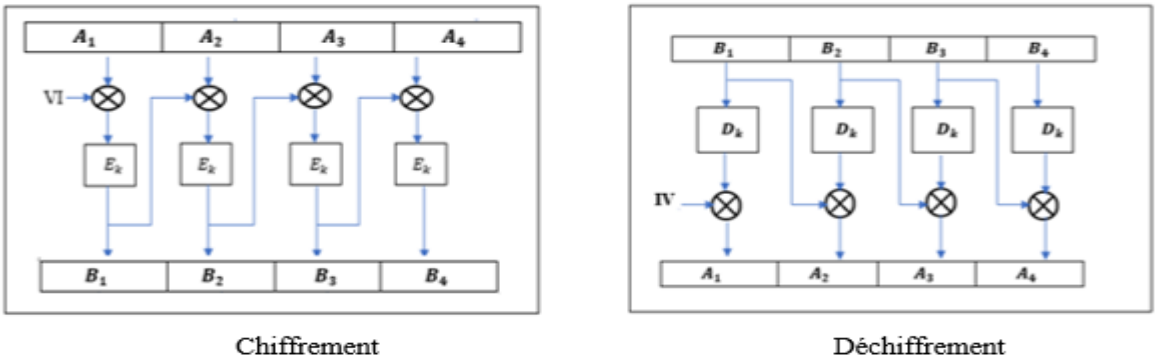

Figure 1.6. Principe du mode CBC

Pour déchiffrer un bloc en mode CBC, on applique l'algorithme de déchiffrement (C_K)au bloc chiffré (B_1), et puis on le combine par ou exclusif avec le bloc chiffré précédent, respectivement avec le vecteur initial.

1.8.3. Mode OFB

En mode OFB **(**Output Feedback**)** ou chiffrement à rétroaction de sortie est une variante du mode CFB [10]. Le mode OFB nécessite un IV, c'est-à-dire que le IV doit être unique pour chaque implémentation du mode avec une clé donnée. L'IV est chiffré par l'algorithme de chiffrement (E_K) pour produire le premier bloc de sortie (B_1). On applique sur le bloc de sortie (B_1) un Ou exclusif (XOR) avec le premier bloc clair (A_1), pour produire le premier bloc de texte chiffré. Ensuite, la fonction de chiffrement est invoquée sur le premier bloc de sortie pour produire le deuxième bloc de sortie. Ainsi de suite. Le processus de déchiffrement est le même que le processus de chiffrement [9]. Comme illustré sur la figure 1.7.

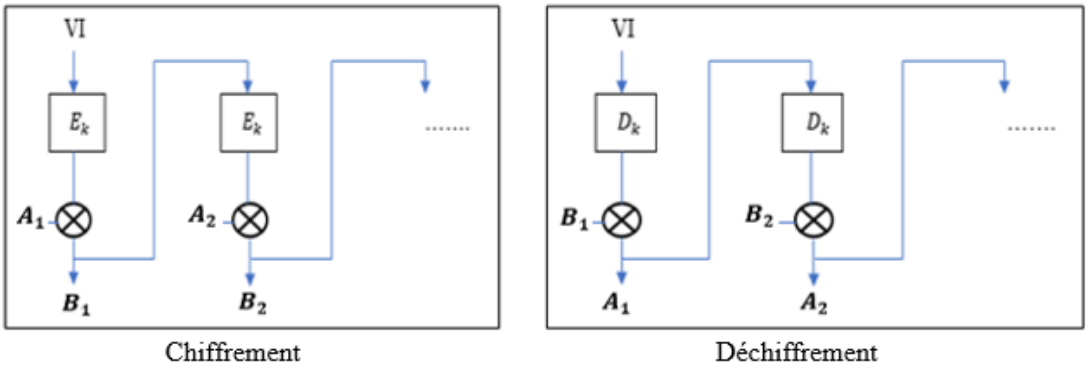

Figure 1.7. Principe du mode OFB

1.8.4. Mode CFB

Le mode de chiffrement CFB (Cipher Feedback) ou chiffrement à rétroaction, est proche du mode OFB mais le flot de bits pseudo-aléatoires dépend cette fois des blocs chiffrés [3].

Comme le montre la figure 1.8. Le premier bloc dans ce mode est un bloc avec une valeur initiale (VI) qui sera chiffré à l'aide de l'algorithme de chiffrement utilisé (E_K) et combiné par un ou exclusif avec le texte en clair (A_1) pour obtenir ainsi le texte chiffré (B_1) qu'on peut alors transmettre.

Le résultat de ce bloc sera réutilisé pour chiffrer le bloc suivant. Plus précisément, il est chiffré avec le même algorithme de chiffrement puis associé à la clé par un ou exclusif [9]. Comme le montre la figure 1.8.

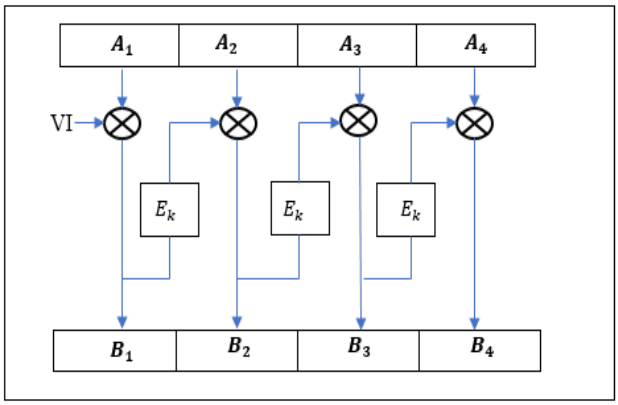

Figure1.8. Principe du mode CFB

1.9. Conclusion

Dans ce chapitre, nous avons présenté le principe du cryptage et du décryptage ainsi que le concept de confusion et de diffusion et leurs importances dans la conception d'algorithmes de cryptage sécurisés. Nous avons également vu en détail les différents modes d'opération en cryptage symétrique par bloc.

Dans le chapitre suivant nous allons décrire en détails les algorithmes standards de cryptage symétrique (DES, AES et Triple DES) qui font l'objet de notre travail.

Chapitre2 : Les algorithmes de cryptage symétrique

2.1. Introduction

Les méthodes de cryptage symétrique sont rapides et performantes car ils nécessitent des clés secrètes de petite taille et offrent plusieurs modes opératoires.

Dans ce chapitre, nous allons revoir en détail trois principaux algorithmes de chiffrement symétrique qui sont DES, TDES (Triple DES) et AES en vue d'une implémentation matérielle sur FPGA.

2.2. DES (Data Encryption Standard)

Le développement de l'algorithme de cryptage symétrique DES a été lancé en 1972. Le but était de concevoir un algorithme de cryptage standard pour l'industrie/l'administration américaines en visant une implémentation matérielle à bas cout mais suffisamment sur.

DES est un algorithme de chiffrement symétrique par blocs de 64bits. Il était très utilisé dans le domaine des cartes magnétiques et des cartes à puce [7]. Il permet de chiffrer un bloc de 64bits d'un texte en clair en un texte chiffré de même taille à l'aide d'une combinaison d'opérations binaires primitives tout en garantissant que ces opérations soient réversibles (chiffrement /déchiffrement). Le point fort du DES c'est qu'il soit facile à implémenter sur des plateformes matérielles car il n'utilise pas d'opérations qui nécessitent des ressources de calcul importantes.

Pour chiffrer un bloc (x) de 64bits, DES utilise différentes substitutions (f), permutations et rotations binaires pour comme le montre la figure 2.1. D'après la figure 2.1, on note que le bloc de 64bits en entrée passe par :

- Une permutation initiale (*IP*) en entrée et une permutation finale inverse (IP^{-1}) en sortie.
- Entre les deux permutations IP et IP^{-1}, les 64bits bits sont cryptés pendant 16 tours appelé aussi rondes ou round en anglais.
- Chaque tour comporte un réseau Feistel qui est une technique de cryptage symétrique par bloc ayant une fonction non linéaire (f) [7].
- Une clé (*Key k*) de taille 64 bits est utilisée comme une clé secrète dont 8 bits sont réservés pour la vérification de la parité (intégrité) de la clé. Cela signifie que réellement seulement les 56 bits restants sont utilisés à chaque tour de cryptage. Il peut donc y avoir 2^{56} combinaisons de clés différentes.
- Vu qu'il existe 16 tours, 16 sous-clés de 48 bits notés de *k1* à *k16* sont dérivées de la clé principale de 56 bits en utilisant une transformation significative.

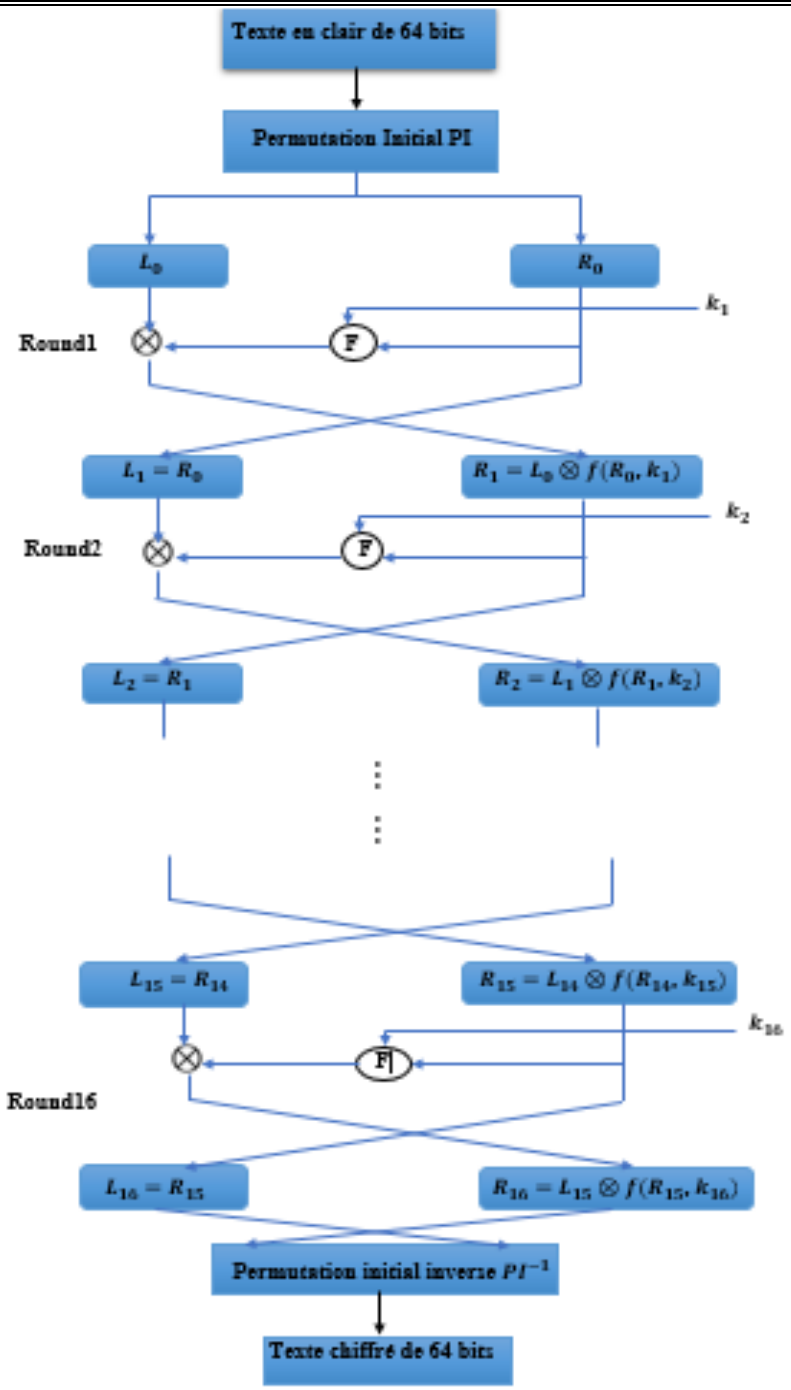

Figure 2.1. Chiffrement DES

2.2.1. Permutation initiale

Chaque bloc de 64bits du texte en clair subit une permutation, qui peut être représenté par la matrice de permutation initiale en fonction de la table de permutation présente dans le tableau 2.1. En parcourant la table de permutation de gauche à droite puis de haut en bas, on peut voir que le 58éme bit du bloc de texte 64bits est permuté avec le 1er bit du bloc, le 50éme bit avec 2éme et ainsi de suite [5].

IP	58	50	42	34	26	18	10	2
	60	52	44	36	28	20	12	4
	62	54	46	38	30	22	14	6
	64	56	48	40	32	24	16	8
	57	49	41	33	25	17	9	1
	59	51	43	35	27	19	11	3
	61	53	45	37	29	21	13	5
	63	55	47	39	31	23	15	7

Tableau 2.1. La table de permutation initiale en DES [11]

2.2.2. Réseau de Feistel

Après la permutation initiale (IP), le bloc avec les 64bits permutés est divisé en deux blocs 32 bits notés Li et Ri (L pour Left en anglais et R pour Right) ou i est le numéro du tour ou round. Comme on a en DES 16 tours ou round de cryptage, on note L0 et R0 l'état initiale (avant le tour 1) où L0 comprend les bits de position paire dans le texte en clair et R0 les bits de position impaire [5]. Chaque tour comprend un ensemble d'opérations (fonctions de permutation et de substitution S-box) appelé réseau de Feistel comme le montre la figure 2.2.

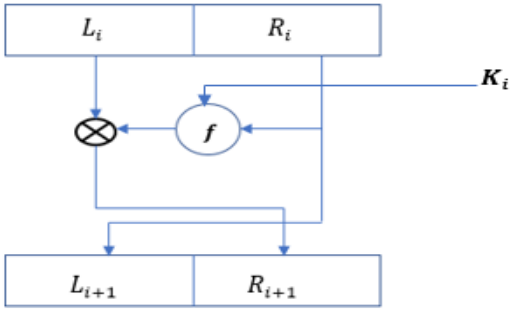

Figure 2.2. Réseau de Feistel en DES

Un réseau de Feistel a été conçu par le cryptologue d'IBM Horst Feistel pour la construction d'algorithmes de chiffrement symétrique par blocs [8]. Dans ce réseau, les 32 bits du bloc R_i sont transformées avec une fonction non linéaire f et une sous-clé k_i puis on applique un OU exclusif (XOR) avec le bloc des 32 bits.de gauche L_i. Après chaque tour i, les deux côtés du bloc de données sont inversés et cela continue de la même façon jusqu'au dernier tour (i = 16) de sorte que [7]:

$$L_{i+1} = R_i. \qquad \textbf{(2.1)}$$

$$R_{i+1}=L_i \oplus f_i(L_i). \qquad \textbf{(2.2)}$$

2.2.3. Fonction de Feistel

f_i est une fonction du réseau de Feistel vu comme le cœur du cryptage DES et qui peut comprendre des opérations linéaire (permutations) et des opérations non linéaire (substitution) comme le montre la figure 2.3 [11].

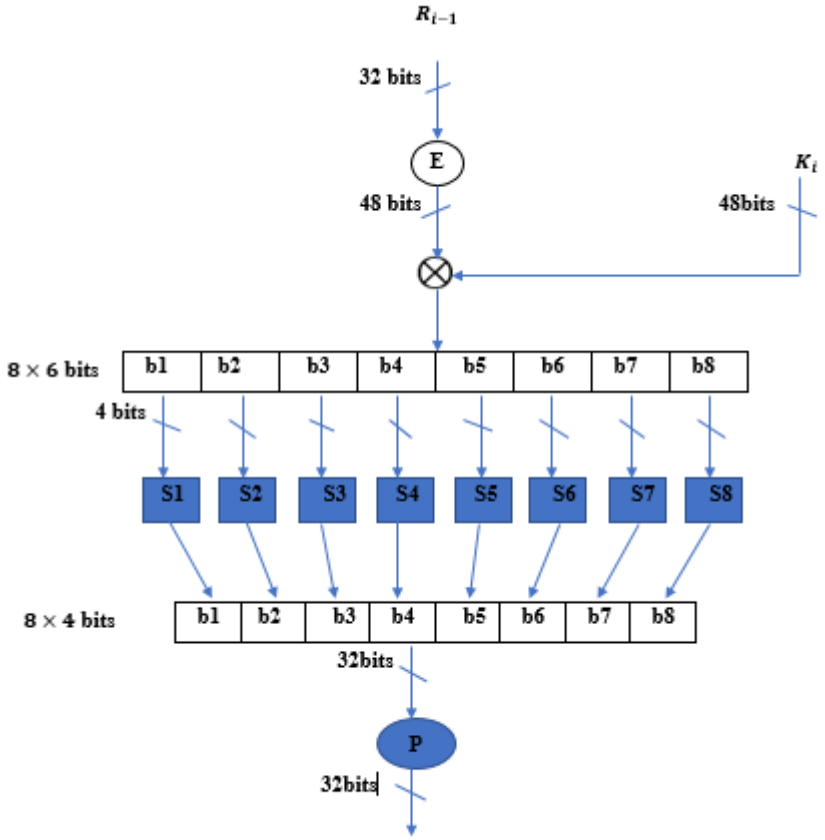

Figure 2.3. Fonction de Feistel

D'après la figure 2.3, un bloc (R_{i-1}) de 32 bits subit en premier une expansion (E) à 48 bits du nombre de bits. Cela est possible grâce à la table d'expansion décrite sur le tableau 2.2 [11].

E	32	1	2	3	4	5
	4	5	6	7	8	9
	8	9	10	11	12	13
	12	13	14	15	16	17
	16	17	18	19	20	21
	20	21	22	23	24	25
	24	25	26	27	28	29
	28	29	30	31	32	1

Tableau 2.2. Table d'expansion de la fonction Feistel

L'expansion est réalisée en redoublant l'utilisation de certains bits. Ensuite, le nouveau bloc de 48bits est crypté en appliquant un Ou exclusif avec la sous-clé k_i de 48bits correspondante au tour i. Les 48 bits résultants de l'expansion précédente sont répartis en 8 sous-blocs de 6 bits et introduits dans huit fonctions de substitutions différentes appelées S-box [5] que nous allons voir en détail dans la sous-section suivante. A la sortie des S-box on obtient un bloc de 32 bits. Ce bloc est soumis à une permutation P selon une table de permutation décrite dans le tableau 2.3[11].

P	16	7	20	21	29	12	28	17
	1	15	23	26	5	18	31	10
	2	8	24	14	32	27	3	9
	19	13	30	6	22	11	4	25

Tableau 2.3. Table de permutation P de la fonction Feistel

Le résultat de la permutation P est finalement soumis à un Ou exclusif avec le (L_{i-1}) de départ pour donner (R_i), tandis que le (R_{i-1}) donne (L_i).

2.2.4. Fonctions de substitution (S-box)

Les 48 bits résultants de l'expansion sont répartis en 8 sous-blocs de 6 bits et introduits dans huit fonctions de substitutions différentes appelées S-box (S1, S2,…,S8) [5][11]. Chacune des huit S-box mappe une entrée 6 bits à une sortie 4 bits selon une table de correspondance S. Chaque S-box contient 2^6 = 64 entrées généralement représentées par une table à 16 colonnes et 4 lignes. Toutes les boîtes S utilisés en DES sont répertoriées dans les tableaux 2.4 [11]**.**

La méthode à suivre pour utiliser ces S-box est comme suit :

- Le bit MSB et le bit LSB de chaque entrée 6 bits donnent le numéro de la ligne du tableau. Les 4 bits restants donnent le numéro de la colonne.
- Le nombre entier obtenu au croisement ligne-colonne donne une décimale sur 4 bits.

Prenons l'exemple illustré dans la figure 2.4.

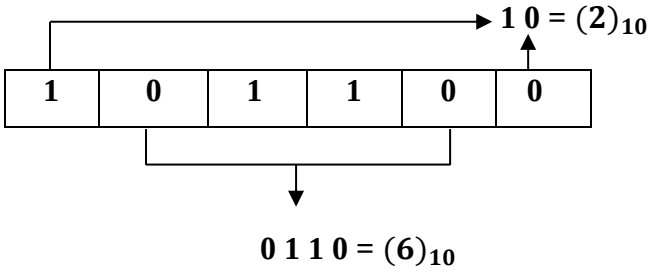

Figure 2.4. Exemple d'utilisation des S-box en DES

L/C		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S1	**0**	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
	1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
	2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
	3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13
S2	**0**	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
	1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
	2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
	3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09
S3	**0**	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
	1	13	07	00	09	03	04	06	10	10	08	05	14	12	11	15	01
	2	13	06	04	09	08	15	03	00	15	01	02	12	05	10	14	07
	3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12
S4	**0**	07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
	1	13	08	11	05	06	02	13	01	10	06	12	11	09	05	03	09
	2	10	06	09	00	12	06	02	11	15	12	09	07	03	10	05	04
	3	03	15	00	06	10	09	01	07	05	11	03	14	10	00	06	14
S5	**0**	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
	1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
	2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
	3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03
S6	**0**	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
	1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
	2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
	3	04	03	02	12	09	05	15	10	11	14	01	07	06	00	08	13
S7	**0**	04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
	1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
	2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
	3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12
S8	**0**	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
	1	01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
	2	07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
	3	02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

Tableau 2.4. Table des S-box en DES

Dans cet exemple, on a un nombre codé en binaire 101100. Son premier et dernier bit donne 10, soit 2 en décimal. Les bits 2,3,4 et 5 donnent 0110 soit en 6 décimales. En utilisant la table des S-box dans le Tableau 2.4 on trouve que cela correspond à la ligne n°2, dans la colonne n°6. Il s'agit de la valeur 02 dans la S-box S6, soit en binaire 0010 [11].

Chacun des 8 blocs de 6 bits est transmis à la fonction de substitution S1,S2,…, S8 correspondante ce qui donne en sortie 8 valeurs de 4 bits chacune. Ainsi chaque bloc 6 bits est remplacé par un bloc 4 bits. Ces bits sont regroupés pour former un bloc 32 bits à la sortie des S-Box.

2.2.5. Permutation finale

A la fin des 16 tours de cryptage par réseau de Feistel, les deux derniers blocs L_{16} et R_{16} sont collés ensemble pour former un seul bloc crypté de 64 bits et subissent une permutation finale (PI^{-1}) qui est l'inverse de la permutation initiale (PI). La table de permutation correspondante est donnée par le tableau 2.5. Cela signale la fin de toutes les étapes du cryptage DES avec en sortie un bloc 64bits crypté.

PI^{-1}	40	8	48	16	56	24	64	32
	39	7	47	15	55	23	63	31
	38	6	46	14	54	22	62	30
	37	5	45	13	53	21	61	29
	36	4	43	12	52	20	60	28
	35	3	42	11	51	19	59	27
	34	2	41	10	50	18	58	26
	33	1	40	9	49	17	57	25

Tableau 2.5. Table de la permutation finale PI^{-1}

2.2.6. Génération des clés

L'algorithme DES utilise 16 sous-clés différentes de 48 bits dérivés à partir de la clé initiale de 64 bits. Pour cela, avant le début du cryptage DES, ces sous-clés sont générées à l'aide d'opérations de permutations notés PC1 et PC2 et une rotation à gauche [5].

Comme le montre la figure 2.5, la clé initiale *K* de 64 bits est soumise aux premières opérations de permutation PC1 (réduction de 8 bits de parité) suivant une table de permutation PC1 sur le tableau 2.6 [5].

Ensuite, les 56 bits sont divisés en deux blocs de 28 bits (G0 et D0). A partir de ce moment, à chaque tour ou round, chaque bloc est traité séparément soumis à une rotation vers la gauche d'un ou de deux bits selon l'ordre du tour en fonction d'une table de décalage gauche LS (Left Shift) illustrée dans le tableau 2.7. [5].

Enfin, le bloc résultant passe encore par la deuxième permutation PC2 (une autre réduction de 8 bits) selon le tableau 2.8 [5] pour obtenir finalement une sous-clé de 48 bits.

Ces opérations sont répétées 15 fois pour générer les 15 clés (k2,…,k16). La différence réside dans l'incrémentation des paramètres de permutation.

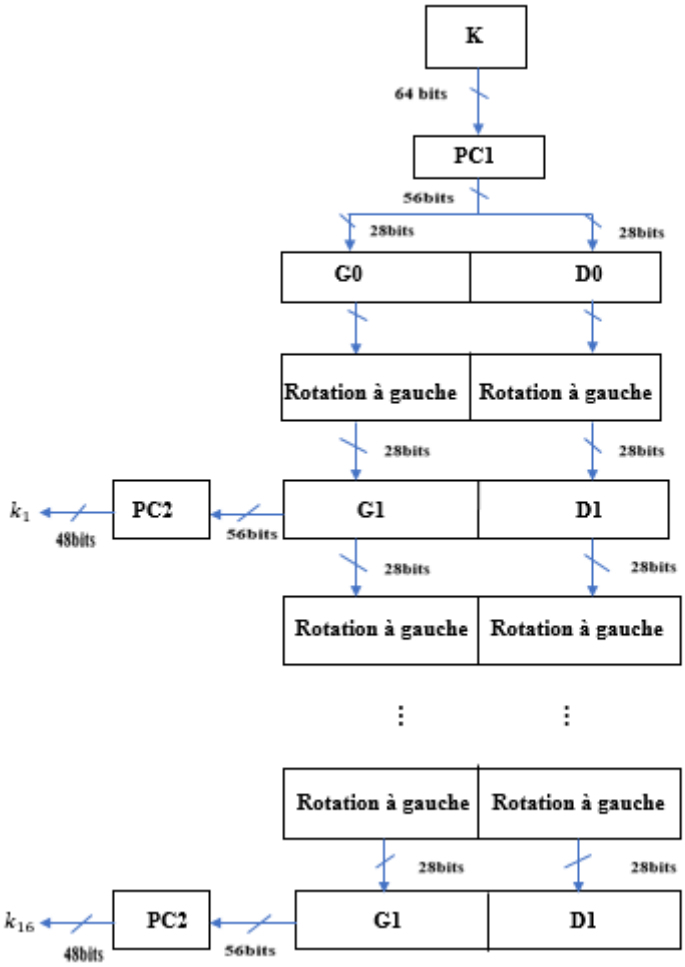

Figure 2.5. Génération des sous clés DES

PC1	57	49	41	33	25	17	9	1
	58	50	42	34	26	18	10	2
	59	51	43	35	27	19	11	3
	60	52	44	36	63	55	47	39
	31	23	15	7	62	54	46	38
	30	22	14	6	61	53	45	37
	29	21	13	5	60	52	44	36

Tableau 2.6. Table de permutation PC1

Itération	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Nombre de décalage	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tableau 2.7. Table des décalages gauche LS

PC2	14	17	11	24	1	5	3	28
	15	6	21	10	23	19	12	4
	26	8	16	7	27	20	13	2
	41	52	31	37	47	55	30	40
	51	45	33	48	44	49	39	56
	34	53	46	42	50	36	29	32

Tableau 2.8. Table de permutation PC2

2.2.7. Déchiffrement

L'un des avantages du DES est que le déchiffrement est effectué par le même algorithme, sauf que cette fois en inversant l'ordre d'application des clés, c'est-à-dire en utilisant k16 à la première itération et k15 à la seconde ainsi de suite jusqu'à k1 à la seizième [11][5]. Cela est dû au fait que la permutation finale est l'inverse de la permutation initiale.

$$R_{i-1} = L_i \quad \textbf{(2.3)}$$

$$L_{i-1} = R_i \oplus f(L_i, k_i) \quad \textbf{(2.4)}$$

2.3. Triple DES (Triple Data Encryption Standard)

En 1990, Eli Biham et Adi Shamir ont développé la cryptanalyse différentielle, il permet de craquer des versions restreintes du DES, dans lesquelles le nombre de tours de la boucle principale est réduit. Ainsi, un DES à 8 ou 10 tours pouvait facilement être cassé. d'autre part, un ordinateur appelé « DES cracker » qui contient 1536 puces et capable de rechercher 88 milliards de clés par seconde, a réussi à concurrencer les laboratoires RSA, craquant une clé DES en 56 heures [5]. De ce fait, l'algorithme de TDES a été créé par ce qu'il permet de renforcer la sécurité du DES.

Triple DES est un algorithme de cryptage symétrique par bloc, connue par les noms (TDES, 3-DES), ce nom explique que trois applications de DES à la suite de chaque bloc avec 3 clés de 56bits différentes (112) [7]. Le TDES consiste à appliquer plusieurs chiffrements et déchiffrement et chaque application DES utilise une clé diffèrent de 56 bits comme le montre la figure 2.6.

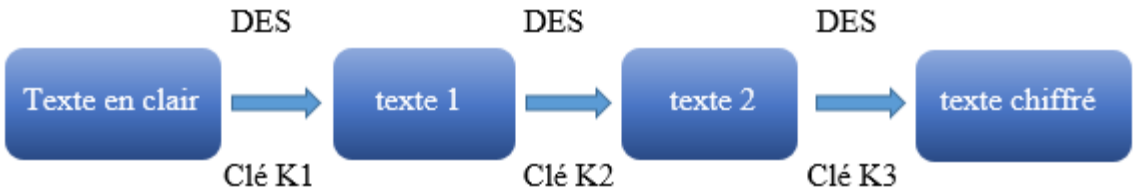

Figure 2.6. Algorithme du chiffrement Triple DES

2.4. AES (Advanced Encryption Standard)

L'algorithme AES est un algorithme de cryptage symétrique par bloc qui a remplacé DES. En septembre 1997. Le NIST a lancé un appel d'offre pour un algorithme de cryptage symétrique. La norme AES exige une taille de bloc de 128 bits et une clé de 128 bits,192 et 256 bits [12].

Selon le tableau 2.9, le nombre de tours *Nr* dépend de la taille des clés des 10 tours pour une clé de 128bits, 12 tours pour une clé de 192 bits et 14 tours pour une clé de 256. La taille du bloc est de 128 bits [8] et la longueur de clé est représenté par *Nk*= 4, 6 ou 8, qui reflète le nombre de mots de 32 bits (nombre de colonnes) dans la clé de chiffrement. La longueur du bloc d'entrée, du bloc de sortie et de l'état est de 128 bits. Ceci est représenté par Nb=4, qui reflète le nombre de mot de 32 bits (nombre de colonnes) [13].

	Nr	**Nk**	**Nb**
AES-128 bits	10	4	4
AES-192 bits	12	6	4
AES-256 bits	14	8	4

Tableau 2.9. Configurations possibles en chiffrement AES.

Dans ce travail, nous allons se concentrer sur l'AES-128 en vue d'une implémentation matérielle. Le reste des configurations est différente seulement dans la taille de la clé et le nombre de tours.

AES se compose de 10 tours, chaque tour se compose ce qu'on appelle des couches. Chaque couche manipule les 128 bits du chemin de données. Le chemin de données est également appelé l'état de l'algorithme. Il n'y a que trois types de calques différents à chaque tour, à l'exception du premier qui se compose de quatre couches (Subbytes, *Shiftrows, Mixcolumn,* et *AddRound*).

De plus, le dernier tour n'utilise pas la transformation MixColumn, ce qui rend le schéma de chiffrement et de déchiffrement symétrique. AES, quant à lui, crypte les 128 bits en une seule itération comme le montre la figure 2.7 [11].

En AES-128bits, chaque bloc 128 bits pris d'un texte en clair est divisé en 16 blocs 8bits (16 octets) dans une matrice 4×4 appelés (matrice state) [8]. Cette matrice est combinée avec la clé afin d'obtenir une nouvelle matrice à inclure dans le calcul des tours. C'est appelée également *AddRound*

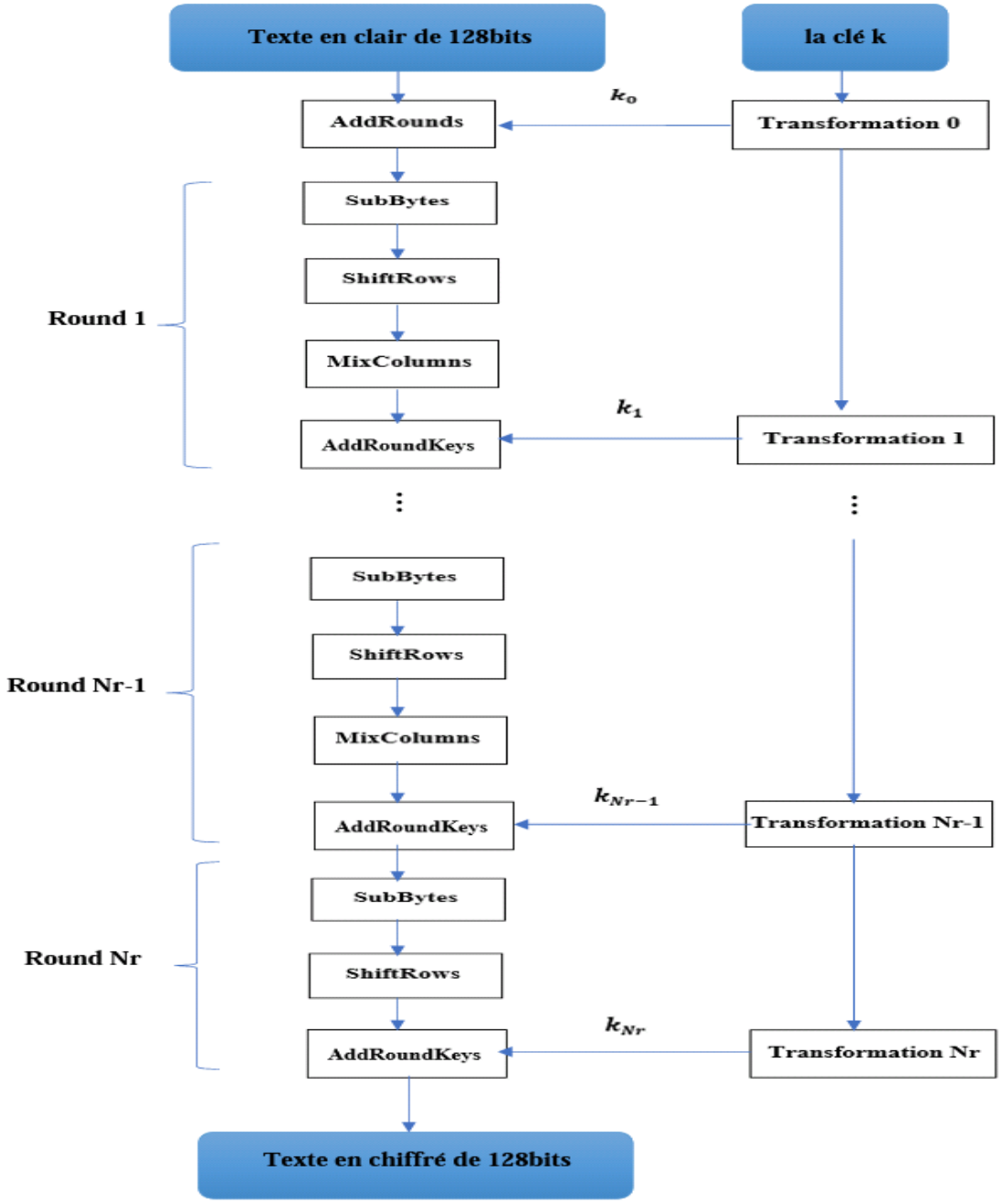

Figure2.7. Algorithme AES

2.4.1. Fonction tour (round)

Nous voyons que dans chaque tour ou tour AES, il y a quatre opérations différents (*Subbytes, ShiftRows, MixColumns, AddRoundKeys)* comme le montre la figure 2.8.

Le résultat de *AddRound* précédent divisée en 16 blocs de 8 bits (16 octets) A0 à A15 et où chaque octet est converti par une S-box. La sortie des 16 blocs de 8 bits résultants B0 à B15 est décalée périodiquement vers la gauche, cette étape est appelée lignes de transformation et elle est mélangée par la transformation *MixColumn* C0 à C15. Enfin, les sous-clés k_i de 128 bits subissent une opération XOR avec un résultat intermédiaire [11].

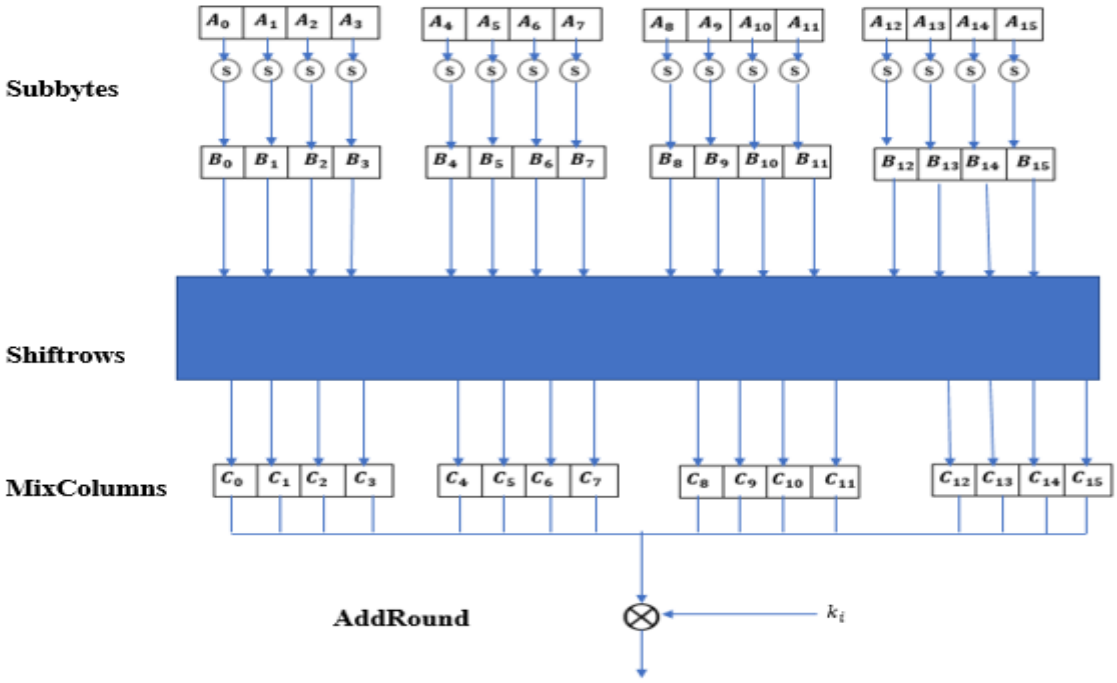

Figure2.8. La fonction Round de AES

2.4.2. Subbytes (S-box)

Comme le montre la précédente figure 2.8, la première couche est constituée de *subbytes* (bytes Substitution) qui se compose de 16 cases parallèles, chacune contenant un octet d'entrée et de sortie, où chaque octet A_i est remplacé par un autre octet B_i. De plus, toutes les S-box sont identiques, contrairement à l'algorithme DES où l'on utilise 8 S-box différents [11].

L'étape *Subbytes* est la seule transformation non-linéaire du chiffrement. Où chaque élément de la matrice State résultant de *AddRound* est permuté selon la table de substitution (S-box) du tableau 2.10 [12].

Cette table comprend deux transformations :

- Dériver une fonction inverse dans le champ de Galois ou Galois Field GF [2^8]. g :
  - $A \rightarrow A^{-1}$. Cette fonction est connue pour ses bonnes propriétés non linéaires.
- Une S-box est construite en combinant sa fonction inverse avec une transformation affine inversible f [14].

$$S\text{-box}[A] = f\,(g\,(A)),\ \forall\ A \in GF\,[2^8] \qquad \textbf{(2.5)}$$

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

Tableau2.10. S-box AES

La transformation affine inversible est donnée par :

$$b = f(A) = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \bmod 2. \qquad \textbf{(2.6)}$$

L'opération *InvSubbytes* effectue la même structure que *Subbytes*, mais à partir de la table inverse de S-box créée en inversant la table S-box en utilisant la transformation affine inversible suivante :

$$b = f^{-1}(A) = \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} \equiv \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \bmod 2. \qquad \textbf{(2.7)}$$

Champ de Galois, nommé d'après ´Evariste Galois, également connu sous le nom de champ fini, se référant à un champ dans lequel il existe un nombre fini d'éléments. Il est particulièrement utile pour traduire des données informatiques car elles sont représentées sous des formes binaires.

Autrement dit, les données informatiques consistent en une combinaison de deux nombres, 0 et 1, qui sont les composants du champ de Galois dont le nombre d'éléments est de deux. La représentation des données sous forme de vecteur dans un champ de Galois permet aux opérations mathématiques de brouiller les données facilement et efficacement [15].

2.4.3. ShiftRows

Cette étape effectue une rotation circulaire de la matrice vers la gauche selon chaque ligne de la matrice $B_i$où comme le montre la figure 2.9, le nombre de décalage varie d'une ligne à l'autre [11] :

- La première ligne ne fait aucune rotation
- La deuxième ligne fait une rotation vers la gauche
- La troisième fait deux rotations vers la gauche
- La dernière fait trois rotation vers la gauche.

L'opération inverse de la fonction *ShiftRows* consiste à faire le mémé rotation pour toutes les lignes dans le sens inverse (rotation à droite).

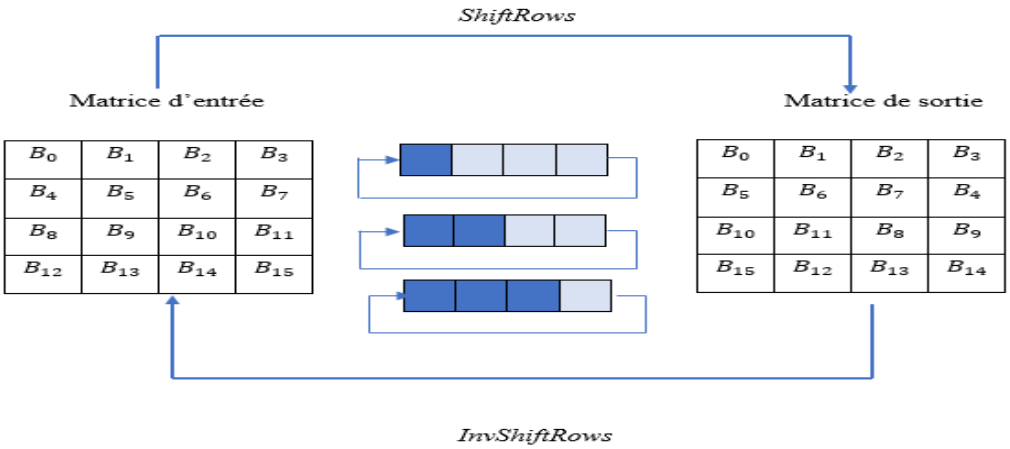

Figure 2.9. Principe du *ShiftRows* et *invShiftRows* en AES

2.4.4. MixColumns

Cette étape est une transformation linéaire qui mélange (mixe) chaque colonne de la matrice state car chaque octet d'entrée influe sur les quatre octets de sortie et chaque colonne de 4 octets est considérée comme un vecteur notée C multiplié par une matrice fixe de 4×4.

La matrice contient des entrées constantes. La multiplication et l'addition des coefficients se font dans un champ de Galois GF (2^8)[11].

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix} \quad \textbf{(2.8)}$$

L'opération inverse de la fonction *Microlumens* consiste à multiplier chaque colonne par une matrice inverse C^{-1}.

$$\begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} \quad \textbf{(2.9)}$$

Pour calculer la matrice de MixColumns, nous devons écrire chaque bit de sortie en fonction des bits d'entrées en utilisant des portes XORs.

Nous discutons maintenant les détails de la multiplication vecteur-matrice qui forme les opérations *MixColumns*. On rappelle que chaque octet d'état Ci et Bi est une valeur sur 8 bits représentant un élément de GF(2^8). Toute arithmétique impliquant les coefficients se fait dans ce corps de Galois. Pour les constantes de la matrice, une notation hexadécimale est utilisée : « 01 » fait référence au polynôme GF(2^8) avec les coefficients (00000001), c'est-à-dire qu'il s'agit de l'élément 1 du corps de Galois.

"02" fait référence au polynôme avec le vecteur de bit (00000010), c'est-à-dire au polynôme x ; et « 03 » fait référence au polynôme avec le vecteur binaire (00000011), c'est-à-dire l'élément de corps de Galois x+1 [11].

En prenant la première colonne de la matrice state après l'étape *Shiftrows* comme suit : Le produit matriciel entre le vecteur (B_0, B_1, B_2, B_3) et la matrice circulante nous donne le système d'équation suivant.

$$\begin{aligned} C_0 &= (\{02\} * B_0) \oplus (\{03\} * B_1) \oplus B_2 \oplus B_3. \\ C_1 &= B_0 \oplus (\{02\} * B_1) \oplus (\{03\} * B_2) \oplus B_3. \\ C_2 &= B_0 \oplus B_1 \oplus (\{02\} * B_2) \oplus (\{03\} * B_3). \\ C_3 &= (\{03\} * B_0) \oplus B_1 \oplus B_2 \oplus (\{02\} * B_3). \end{aligned} \quad \textbf{(2.10)}$$

- **La multiplication de B*01**

La multiplication par 01 est une multiplication par l'identité et n'implique aucune opération explicite [11].

- **La multiplication de B*02**

La multiplication par 02 peut également être implémenté comme une multiplication par x, qui est un décalage à gauche d'un bit, et une réduction modulaire avec P(x)= $x^8 + x^4 + x^3 + x + 1$ [11]. Pour calculer le produit, nous procidence comme suit :

$$b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0$$

$$\times \qquad\qquad x$$

$$b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x^1+b_7$$

$$\textbf{Mod} \qquad\qquad x^8+\ x^4+\ x^3+\ x+1$$

$$b_6x^7+b_5x^6+b_4x^5+(b_7+b_3)x^4+(b_7+b_2)x^3+b_1x^2+b_1x^2+(b_7+b_0)x+b_7$$

Nous obtenons le résultat suivant :

$c_0=b_7$, $c_1=b_7+b_0$, $c_2=b_1$, $c_3=b_7+b_2$, $c_4=b_7+b_3$, $c_5=b_4$, $c_6=b_5$, $c_7=b_6$.

- **La multiplication de B*03**

La multiplication par 03, qui représente le polynôme (x+1), peut être implémenté par un décalage à gauche d'un bit et addition de la valeur d'origine suivie d'une réduction modulaire avec P(x).

Nous obtenons le résultat suivant :

$c_0=b_0+b_7$, $c_1=b_0+b_1+b_7$, $c_2=b_1+b_2+b_7$, $c_3=b_2+b_3+b_7$, $c_4=b_3+b_4+b_7$, $c_5=b_4+b_5+b_7$, $c_6=b_5+b_6$, $c_7=b_6+b_7$.

De la même structure pour *InvMixcolumns*, nous devons écrire chaque bit de sortie en fonction des bits d'entrées en utilisant des portes XORs.

En prenant la première colonne de la matrice state après l'étape *AddRound* comme suit : Le produit matriciel entre le vecteur (B_0, B_1, B_2, B_3) et la matrice circulante nous donne le système d'équation suivant.

$$\begin{aligned} C_0 &= (\{0E\} * B_0) \oplus (\{0B\} * B_1) \oplus (\{0D\} * B_2) \oplus (\{09\} * B_3). \\ C_1 &= (\{09\} * B_0) \oplus (\{0E\} * B_1) \oplus (\{0B\} * B_2) \oplus (\{0D\} * B_3). \\ C_2 &= (\{0D\} * B_0) \oplus (\{09\} * B_1) \oplus (\{0E\} * B_2) \oplus (\{0B\} * B_3). \\ C_3 &= (\{0B\} * B_0) \oplus (\{0D\} * B_1) \oplus (\{09\} * B_2) \oplus (\{0E\} * B_3). \end{aligned} \qquad \textbf{(2.11)}$$

- **La multiplication de B*09**

La multiplication par 09, qui représente le polynôme (x^3+1), peut être implémenté par un décalage à gauche d'un bit et addition de la valeur d'origine suivie d'une réduction modulaire avec P(x) [11].

Nous obtenons le résultat suivant :

$c_0=b_5+b_6$, $c_1=b_5+b_7$, $c_2=b_0+b_6$, $c_3=b_0+b_1+b_5+b_6+b_7$, $c_4=b_1+b_2+b_5+b_7$, $c_5=b_2+b_3+b_6$, $c_6=b_3+b_4+b_7$, $c_7=b_4+b_5$.

- **La multiplication de B*0B**

La multiplication par 0B, qui représente le polynôme $(x^3 + x + 1)$, peut être implémenté par un décalage à gauche d'un bit et addition de la valeur d'origine suivie d'une réduction modulaire avec P(x). [11].

Nous obtenons le résultat suivant :

$c_0=b_0 + b_5 + b_7$, $c_1=b_0 + b_1 + b_5 + b_7$, $c_2=b_1 + b_2 + b_6 + b_7$, $c_3=b_0 + b_2 + b_3 + b_5$, $c_4=b_1 + b_3 + b_4 + b_5 + b_6 + b_7$, $c_5=b_2 + b_4 + b_5 + b_6 + b_7$, $c_6=b_3 + b_5 + b_6 + b_7$, $c_7=b_4 + b_6 + b_7$.

- **La multiplication de B*0D**

La multiplication par 0D, qui représente le polynôme $(x^3 + x^2 + 1)$, peut être implémenté par un décalage à gauche d'un bit et addition de la valeur d'origine suivie d'une réduction modulaire avec P(x) [11].

Nous obtenons le résultat suivant :

$c_0=b_0 + b_5 + b_6$, $c_1=b_1 + b_5 + b_7$, $c_2=b_0 + b_2 + b_6$, $c_3=b_0 + b_1 + b_3 + b_5 + b_6 + b_7$, $c_4=b_1 + b_2 + b_4 + b_5 + b_7$, $c_5=b_2 + b_3 + b_5 + b_6$, $c_6=b_3 + b_4 + b_6 + b_7$, $c_7=b_4 + b_5 + b_7$.

- **La multiplication de B*0E**

La multiplication par 0E, qui représente le polynôme $(x^3 + x^2 + x)$, peut être implémenté par un décalage à gauche d'un bit et addition de la valeur d'origine suivie d'une réduction modulaire avec P(x) [11].

Nous obtenons le résultat suivant :

$c_0=b_5 + b_6 + b_7$, $c_1=b_0 + b_5$, $c_2=b_0 + b_1 + b_6$, $c_3=b_0 + b_1 + b_2 + b_5 + b_6$, $c_4=b_1 + b_2 + b_3 + b_5$, $c_5=b_2 + b_3 + b_4 + b_6$, $c_6=b_3 + b_4 + b_5 + b_7$, $c_7=b_4 + b_5 + b_6$.

2.4.5. AddRoundKey

Dans une transformation *AddRoundKey*, il y a deux entrées, la matrice state et la sous clé sont combinées par une simple opération XOR entre chaque octet [NIST. Fips.197].

La taille de la clé de ronde est donc la même que celle de la matrice state. A noter que l'opération XOR est égale à l'addition dans le champ de Galois GF $[2^8]$[11].

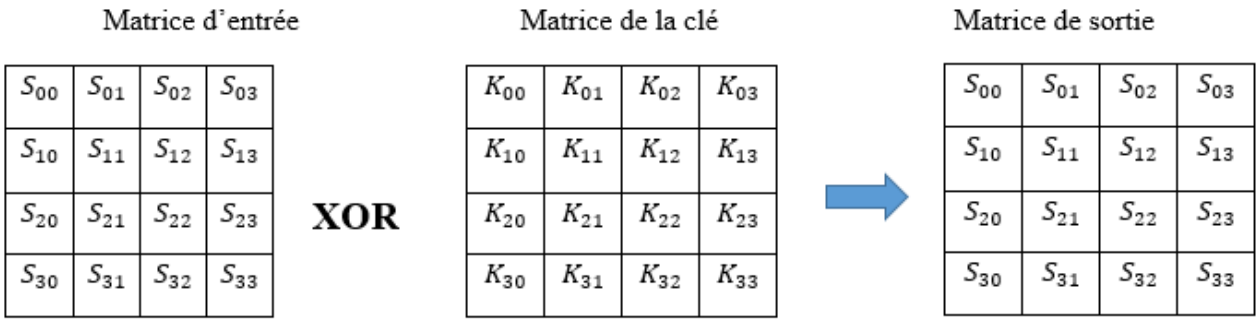

Figure 2.10. Principe du *AddRoundKey* en AES.

2.4.6. Expansion de la clé

L'algorithme AES prend la clé de chiffrement secrète originale K et exécute une routine d'expansion de clé pour créer une table de clés. Notez qu'un XOR avec les sous-clés est utilisé sur les entrées et les sorties AES (*AddRoundKeys*). Ce processus est parfois appelé blanchiment des clés. Le nombre de sous-clés est égal au nombre de tours plus un. Pour une clé 128 bits, le nombre de cycles est de Nr=10, soit 11 sous-clés de 128 bits chacune Les sous-clés sont calculées de manière récursive, et la sous-clé précédente doit être connue afin d'extraire la clé suivante [11].

Ainsi, L'algorithme AES prend en entrée une clé de quatre mots (16 octets) et produit un tableau linéaire de 44 mots (176 octets). Cela est suffisant pour fournir une clé de quatre mots par round pour l'étape initiale *AddRoundKey* et chacun des 10 tours restants [11]. Les sous-clés sont stockées dans un tableau d'extension clé avec comme éléments W [0], ..., W [43]. Les sous-clés sont calculées comme illustré à la figure 2.11.

Les éléments K0, ..., K15 désignent les octets de la clé AES d'origine. Tout d'abord, nous notons que la première sous-clé k0 est la clé AES d'origine, c'est-à-dire que la clé est copiée dans les quatre premiers éléments du tableau de clés W. Les autres éléments de la matrice sont calculés comme suit :

- La clé est copiée dans les quatre premiers mots de la clé développée.
- Le reste de la clé développée est composé de quatre mots à la fois. Chaque mot ajouté w[i] dépend du mot immédiatement précédent, w [i - 1] et du mot quatre positions en arrière, w [i - 4].
- Dans trois cas sur quatre, un simple XOR est utilisé. Pour un mot dont la position dans le tableau w est un multiple de 4, une fonction plus complexe est utilisée.

La figure 2.11 illustre la génération de la clé développée, en utilisant le symbole g pour représenter cette fonction complexe [16].

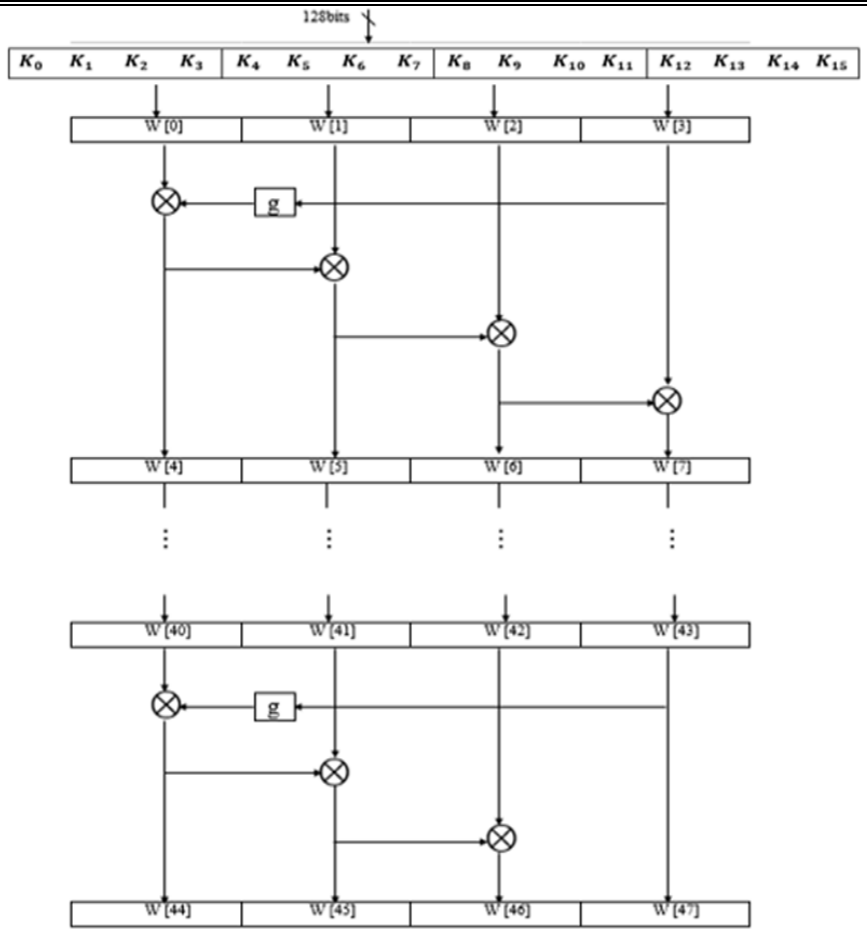

Figure2.11. La génération de la clé de AES

La fonction g () est une transformation non linéaire de quatre octets à l'entrée et la sortie qui consiste une rotation vers le haut d'un octet, chaque octet effectue à une substitution S-box, et lui ajoute un coefficient rond RC.

Le coefficient d'arrondi est un élément du corps de Galois GF [2^8]. C'est-à-dire une valeur sur 8 bits. Il n'est ajouté qu'à l'octet le plus à gauche dans la fonction g () comme le montre la figure2.12 [11]. Ce coefficient de tour varie d'un tour à l'autre selon la règle suivante :

$$\text{RC [1]} = x^0 = (00000001)_2,$$

$$\text{RC [2]} = x^1 = (00000010)_2,$$

$$\text{RC [3]} = x^2 = (00000100)_2,$$

$$\vdots$$

$$\text{RC [10]} = x^9 = (00110110)_2.$$

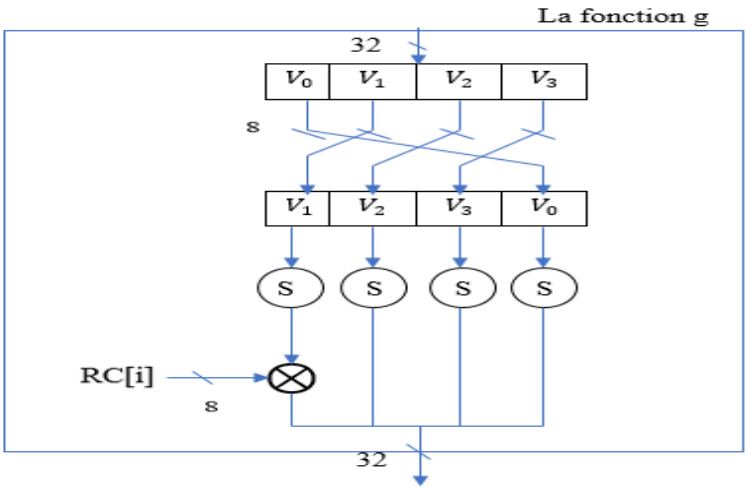

Figure2.12. La fonction g en AES

2.4.7. Déchiffrement

Le déchiffrement de l'algorithme AES-128 est l'inverse de l'algorithme de cryptage AES vu à la figure 2.5 s'il est implémenté dans l'ordre inverse en utilisant bien sûr des transformations inverses tels que : *InvMixColumns*, *InvShiftRows* et *InvSubbytes*. Chacune d'eux a été expliqué précédemment.

2.5. Conclusion

Dans ce chapitre, nous avons présenté le principe de chaque algorithme de cryptage symétrique DES, AES et TDES avec une explication détaillée ; nous allons voir, dans le chapitre suivant, l'étude de circuit FPGA cible ainsi que la méthode utilisée pour l'implémentation d'un tel système de cryptage et les résultats obtenus.

Chapitre3 : Implémentation sur SoC-FPGA

3.1. Introduction

Les microprocesseurs modernes restent limités en termes de performances comparés aux circuits programmable FPGA (Field Programmable Gate Array) malgré qu'ils puissent effectuer des traitements en parallèle en plus du traitement séquentiel classique. En effet, un FPGA reste l'accélérateur matériel favori des algorithmes car contrairement aux microprocesseurs classiques le FPGA utilise un langage de description matériel et une implantation spatiale des taches (parallélisme) ce qui apporte un plus grand gain en performances [17].

Dans ce chapitre, nous allons présenter les résultats obtenus lors des différentes simulations et implémentations sur FPGA des algorithmes AES, DES, TDES vu dans le chapitre précédent. Nous avons utilisé comme plateforme cible de développement et d'implémentation une carte Xilinx PYNQ-Z2 avec comme cœur un système sur puce SoC (System on Chip) de la famille Zynq-7000 qui comprend un microprocesseur et un FPGA. Une discussion des différents résultats est présentée à la fin du chapitre.

3.2. Rappels sur les FPGA

Les FPGA sont des dispositifs qui peuvent être programmés électriquement pour devenir presque n'importe quel type de circuit ou de système numérique [18]. La figure 3.1 montre l'architecture de base d'un FPGA moderne basée sur des cellules SRAM (Static Random Access Memory) pour le routage des circuits logiques et blocs logiques.

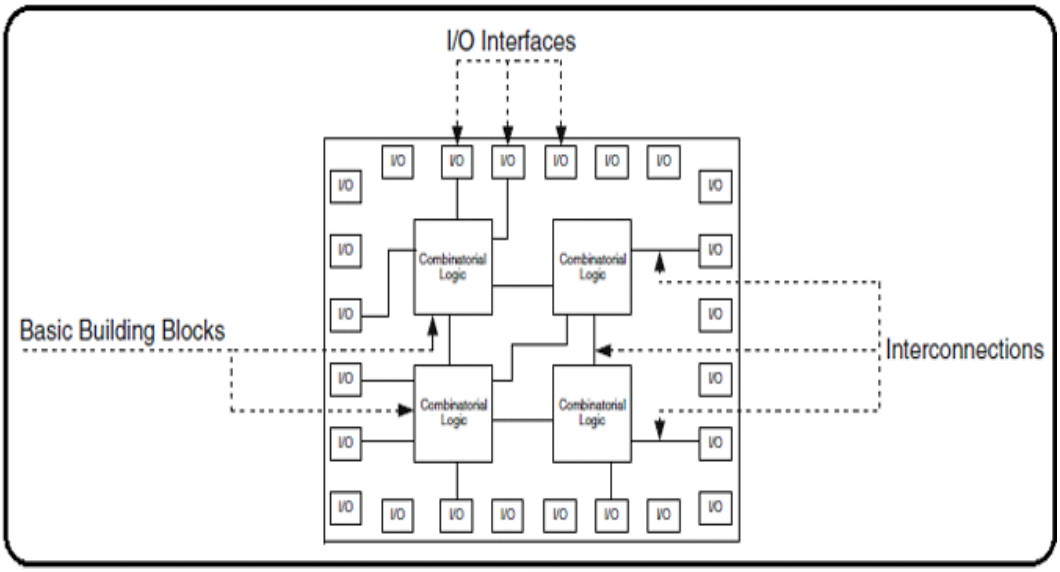

Figure 3.1. Concept architectural de base des FPGA [19]

- IOB (Input Output Blocks) : Ces unités d'entrée-sortie I/O sont utilisées pour interfacer le FPGA avec son environnement externe (périphériques.etc...) [20].

- CLB (Configurable Logic Block) : Les blocs logiques FPGA sont appelés blocs logiques configurables (CLB). Un CLB comprend deux cellules élémentaires appelés SLICE : SLICEM (ROM, logique, RAM et registres à décalage) et SLICEL (ROM, logique) et une matrice dite matrice SWITCH [21]. Comme illustré à la figure 3.2.

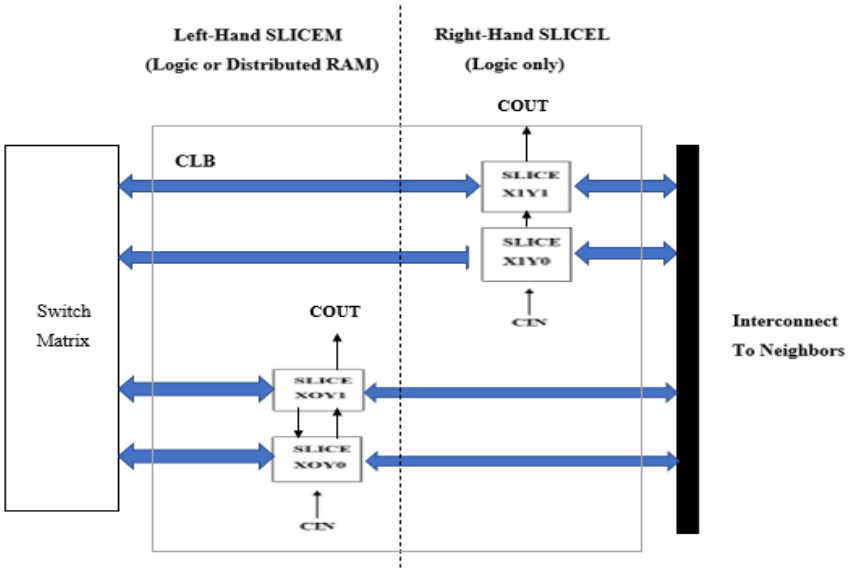

Figure 3.2. Structure d'un CLB de base

- Un CLB comprend:
  - Une matrice SWITCH : Fournit une connectivité avec l'intérieure et l'extérieur du CLB [21].
  - Chaque SLICE (tranche) contient 4 LUT à 6 entrées et 4 bascules flips flops (FF) [21].
  - Les LUT (Look up Table) ou table de correspondance est une table comme une table de vérité avec une seule sortie et peuvent implémenter une seule fonction logique [21].
  - Multiplexeurs où chaque multiplexeur reçoit des signaux d'entrée et de sortie d'un CLB adjacent [21].

Les constructeurs de FPGA ont également ajouté au fil des générations de nombreux éléments tels que :

- Les blocs mémoire RAM : Certains fabricants de FPGA ont inséré à l'intérieur des FPGA, de 10 à 90 mégabits de mémoire selon les fabricants [20].
- Les multiplieurs : la logique des FPGA permet d'effectuer toutes sortes d'opérations arithmétiques [20].
- Les blocs processeur : certains circuits FPGA comprennes des cœurs de processeur tels que les SoC-FPGA de Xilinx (SoC ZYNQ 7000 disponible en 2012). SoC ou System on Chip veut dire

un système sur puce qui est un système intégrant un ou plusieurs cœur ARM avec un FPGA le tout sur une seule puce électronique [20].

3.3. Carte de développement PYNQ-Z2

La figure 3.3 montre la carte XILINX PYNQ-Z2 utilisé dans ce projet.

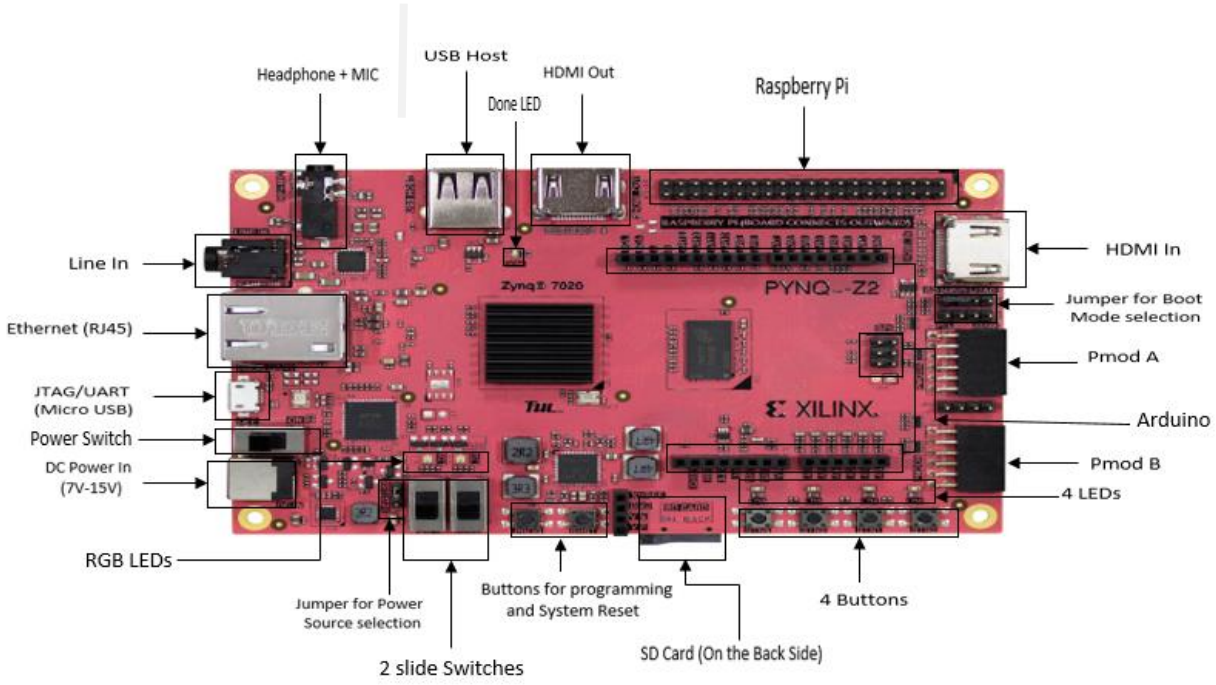

Figure 3.3. Carte cible PYNQ-Z2 [22]

La carte PYNQ-Z2 comprend :

- Un SoC-FPGA XILINX ZYNQ XC7Z020-1CLG400C:
  - 32 Ko d'instructions, 32 Ko de données par processeur Cache L1.
  - Cache L2 unifié de 512 Ko.
  - 256 Ko de mémoire.
  - 85000 cellules logiques programmable
    - 13300 SLICE logiques, chacune avec 4 LUT à 6 entrées et 8 FF
  - Blocs RAM (BRAM) de 630 Ko
  - 20 SLICE DSP.
- Vitesses d'horloge interne supérieure à 450 MHz.
- 2x 12 bit, 1 MSPS On-chip analog-to-digital converter (XADC).
- 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x32b entrées sorties à usage général GPIO.
- 2x USB 2.0, 2x Ports Gigabit Ethernet, 2x périphériques de cartes mémoires SD.

La carte PYNQ-Z2 peut étre programmé en Python et/ou avec un langage HDL d'où le nom PYNQ (**PY**thon pour Zy**NQ**). En effet, PYNQ est un projet XILINX open source qui facilite la conception de

systèmes embarqués avec le SoC Zynq 7000. Cela permet d'exploiter les avantages de la logique programmable ainsi que les avantages des microprocesseurs ARM dans un seul SoC Zynq-7000 [22].

Les SoC Zynq-7000 sont un SoC-FPGA basé sur une architecture Xilinx appelée All programmable SoC. Comme le montre la figure 3.4, cette architecture comporte une partie (PS : Processing System) ARM Cortex A9 double cœur et une logique programmable FPGA (PL : Programmable Logic) dans une seule puce [20].

Le SoC ZYNQ dispose de 9 interfaces AXI entre le PS et le PL. du coté PL, il y a 4x ports AXI Master HP (haute performance), 2x ports AXI GP (General purpose), 2x ports AXI Slave GP et 1x port AXI Master ACP. Il existe également des contrôleurs GPIO dans le PS qui sont connectés au PL.

Il existe aussi quatre classes PYNQ qui sont utilisées pour transférer des données entre les interfaces Aynq PS (y compris la DRAM PS) et PL.

- GPIO : Entrée/ sortie à usage général.
- DMA : Accès direct en mémoire.
- Xlnk : Allocation de mémoire.
- MMIO : Entrée/ sortie mappées en mémoire

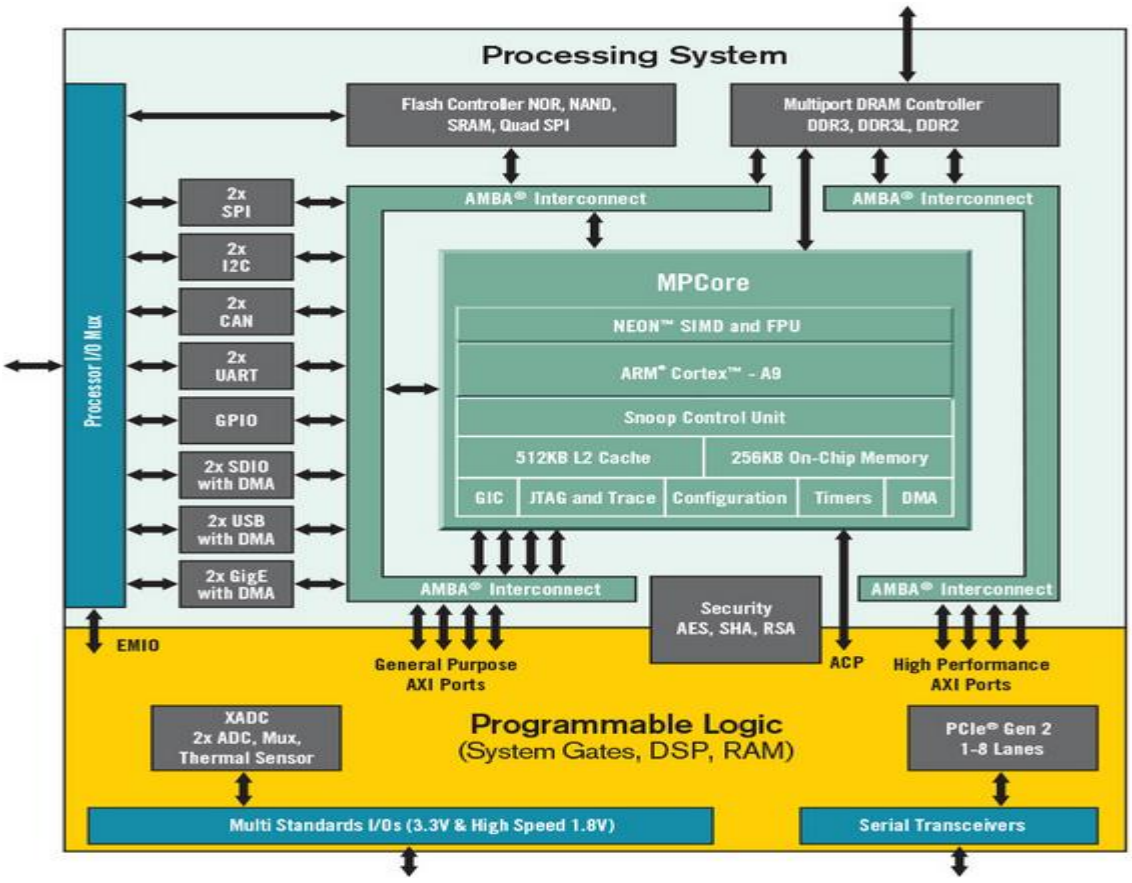

Figure 3.4. Architecture de base du SoC-FPGA Zynq 7000 [22]

3.4. Outils de développement

Nous avons utilisé Vivado Design Suite pour le développement et l'implémentation des algorithmes DES, TDES et AES sur la carte PYNQ-Z2. Vivado Design Suite est une suite logicielle produite par Xilinx pour la synthèse et l'analyse de conceptions HDL (Hardware Description Langage) avec des capacités de développement sur SoC. Vivado offre un environnement qui permet la conception d'une architecture logique en VHDL ou Verilog jusqu'à son implémentation sur la carte cible passant par la simulation et l'analyse. Il fournit un flux de conception centré sur IP (Intellectual property). Un IP est un composant virtuel dont on peut distinguer deux types : le premier correspond à un cœur de processeur, le deuxième correspond à une fonction ou à une opération spécifique [20].

Sur Vivado, on doit passer par trois étapes primordiales pour l'implémentation d'une conception logique quelconque sur SoC-FPGA de la carte cible PYNQ-Z2 :

- **Simulation :** Simuler le circuit en utilisant des vecteurs de test dans le but de vérifier si le code s'exécute correctement.

- **Synthèse :** Lors de la synthèse, le compilateur traduit le premier code en un autre équivalent mais à un niveau d'abstraction plus bas. A ce niveau de synthèse, le compilateur traduit le code de haut niveau en portes logiques ceci en fonction du circuit logique programmable ciblé [19].

- **Implémentation :** Une fois la synthèse terminée, le circuit est implémenté sur le composant en spécifiant les références exactes : circuit cible, fréquence fonctionnement, et autres restrictions matérielles.

Vivado permet également de finaliser ces étapes par plusieurs rapports (Erreurs, I/O, estimation de la consommation en Watts, estimation du temps d'exécution, ressources utilisés …etc…) ce qui permet de savoir si la conception est appropriée pour l'application ciblée [19].

Concernant le langage de développement matérielle choisi pour la conception des architectures logiques des algorithmes de chiffrement symétrique DES, TDES et AES. Nous avons choisi le langage Verilog comme langage HDL (Hardware Description langage). Il a été inventé par Gatway Design Automation Inc en 1984. C'est un langage propriétaire conçu à l'origine pour être utilisé dans leurs simulateurs logiques, la popularité croissante du VHDL (un autre langage HDL) a incité les concepteurs à faire de Verilog un nouveau standard ouvert IEEE 1364. La syntaxe de Verilog est fortement inspirée du langage C [19] ce qui justifie notre choix car ayant appris le langage C auparavant.

3.5. Description de la conception matérielle

Dans cette section, nous allons décrire notre conception des modules programmés Verilog qui vont implémenter les algorithmes DES, TDES et AES sur la carte cible PYNQ-Z2. Ces modules que ce soit en AES ou en DES sont divisés en général sur deux parties, la première partie correspond au design global de l'algorithme et la seconde correspond à la fonction de génération des clés.

3.5.1. Conception DES

La conception DES comprend deux modules : un module global **DES_mode** et un module secondaire **KeySchedule**.

DES_mode c'est le module cœur du design. Il comprend plusieurs fonctions :

- La fonction **IP** : effectue la permutation initiale.
- La fonction **PF** : effectue la permutation inverse.
- Ensemble des fonctions ronde (Expansion, permutation P, S-box, Fonction f)

Comme vu au chapitre précédent, les fonctions de permutation sont seulement une question de permutation des bits d'entrée (texte en clair). On peut donc dire que cet élément ne nécessite pas de ressources logiques. La fonction ronde est répétée 16 fois nous avons donc besoin d'une boucle for. C'est en fait une conception structurelle constitué de plusieurs composants :

- **Xp** (expansion).
- **Si (**S-box) et **P (**permutation).
- **Xor1** (XOR de la sous clé**)** et **Xor2** (XOR de P).

KeySchedule ce module sert à la génération et la synchronisation des clés comme vu au chapitre précédent. Il contient 3 fonctions : PC1 (Permutation Choice-1), PC2 (Permutation Choice-2) et Rotation à gauche (Shift left).

Ce module conduit à générer toutes les sous-clés sur une seule ligne. Ainsi, le composant clé (et ses sous-composants) n'utilise que des ressources filaires car il se compose uniquement d'opération de permutation et de rotation. Cette partie s'exécutera donc très rapidement et aucune optimisation n'aura d'effet.

3.5.2. Conception AES

Le code de la conception AES comprend deux modules : un module global **AES_mode** et un module secondaire **KeySchedule**.

AES_mode c'est le module cœur du design AES. Il comprend plusieurs fonctions essentielles :

- *AddRound*..
- *SubBytes/InvSubBytes* : chaque élément résultant est permuté selon la table de substitution.
- *ShiftRows/InvShiftRows* : chaque élément d'une ligne subi une rotation vers la gauche.
- *Mixcolumns/InvMixcolumns :* un produit matriciel.

Dans ce module, comme vu au chapitre précédent une opération XOR est effectuée en premier entre la matrice « STATE » avec la clé de chiffrement « KEY », le résultat intermédiaire sera la même transformation avec quatre blocs de 32 bits effectuée en parallèle.

KeySchedule ce module sert à la génération et la synchronisation des clés comme vu au chapitre précédent. Il contient trois fonctions : C'est le composant de génération de la clé, il contient 3 fonctions :

- Matrice d'extension W (W [0] ,.,.,W[43]).
- Fonction g.
- XOR

Comme vu au chapitre précédent, ce module doit pouvoir stockées dans la matrice d'expansion de clé W les 11 sous-clés (10 sous-clés plus la clé d'origine) qui se composent des mots (W [1] ,.,.,W[44]). Les 4 W (W [1], W [2], W [3], W [4]) contient la clé d'origine key, et W [5] à W [44] celui qui entre dans une boucle for ou nous calculons chaque 4 W en utilisant XOR et la fonction g.

Chaque W[4i] , entre dans la fonction g() qui est en fait une conception structurelle constitué de plusieurs composants :

- Une rotation vers le haut d'un octet (8bits) (permutation).
- Une substitution S-box.
- Coefficient rond RC : c'est une constante qui xorée les dernière 8bits de Substitution.

3.6. Résultats de l'implémentation DES

Nous avons conçu sur Vivado les modules DES précédents en incluant tous les modes de chiffrements : ECB, CBC, OFB et CFB.

Pour chaque mode, nous avons utilisés :

- Une clé de chiffrement « key » constante : $(\text{F33C77F99BBCFFF1})_{16}$
- Un texte en clair « plaintext » constant : $(\text{0123456789ABCDEF})_{16}$

Nous avons effectué une simulation comportementale pour chaque mode afin de vérifier si l'exécution du chiffrement est correcte en comparant le texte crypté « ciphertext » obtenu en simulation Vivado avec celui obtenu en simulation MATLAB et sur d'autres outils de chiffrement tiers.

Enfin, pour chaque mode de chiffrement nous avons pu synthétiser et implémenter et générer le bitstream de notre conception DES et le charger sur la carte cible PYNQ-Z2 en programmant un banc d'essai qui comprend un bouton de lancement du chiffrement symétrique DES et une LED témoin qui indique l'état du chiffrement.

3.6.1. Schéma RTL

Nous avons généré l'analyse RTL (Register Transfer Level) sur Vivado afin de visualiser le schéma de représentation du circuit de génération des sous clés DES commun à tous les modes de chiffrement comme le montre la figure 3.5.

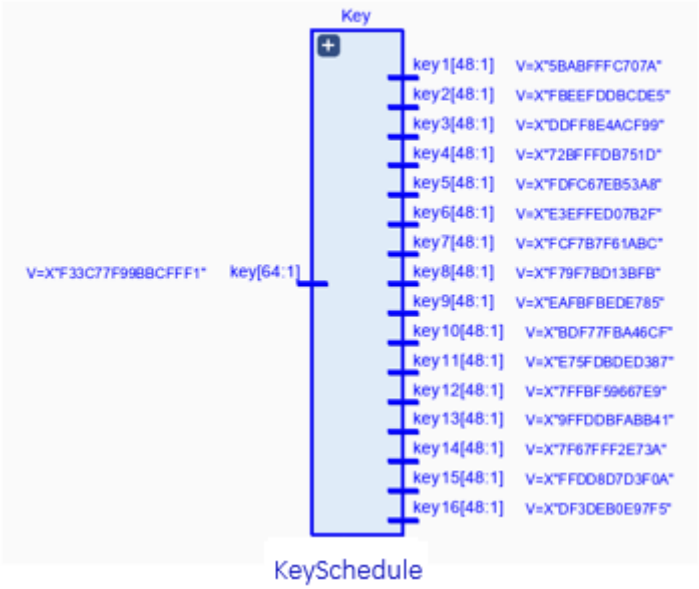

Figure3.5. Schéma RTL du circuit de génération des sous clés DES

En parcourant le schéma RTL, on remarque que la boite noire implémentée par notre algorithme ne représente que le nombre d'entrées/sorties (1 entrée « key » et 16 sorties « key1, …, key16 »). Comme le montre la figure 3.5. Après la vérification des résultats, nous avons remarqué que les sous clés de l'algorithme DES sur le bloc VIVADO sont les même avec les sous clés de l'algorithme DES dans les simulations MATLAB donc corrects.

3.6.2. Mode DES-ECB

Les résultats de simulation en mode DES-ECB sont illustrés sur les figures 3.6 et 3.7. « clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé.

Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.7 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

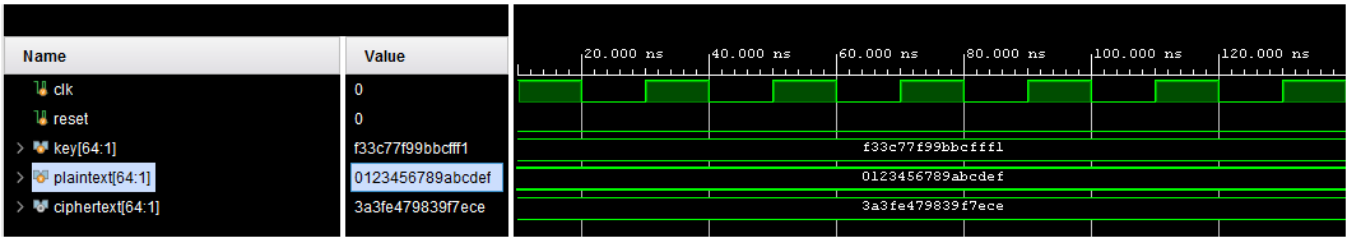

Figure.3.6. Résultat de simulation du chiffrement DES-ECB

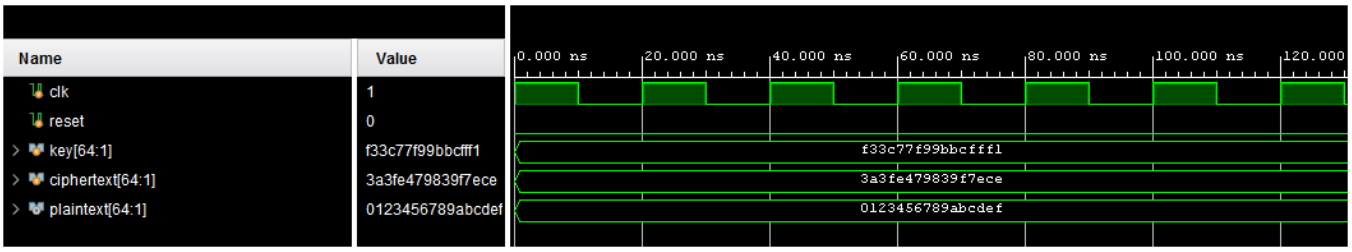

Figure.3.7. Résultat de simulation du déchiffrement DES-ECB

Après la simulation comportementale, nous présentons sur le tableau 3.1 les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement DES-ECB			Déchiffrement DES-ECB		
	Utilisation	Disponible	%	Utilisation	Disponible	%
#LUT	1166	53200	2.2%	1167	53200	2.2%
#Slice Occupées	316	13300	2.3%	326	13300	2.3%
SLICEM	162			167		
SLICEL	154			159		
#BRAM	0	140	0%	0	140	0%
#DSP	0	220		0	220	
#Flip Flops	0	106400		0	106400	
Puissance (W)	0.863			0.863		
Temps Execution (ns)	1010			1010		

Tableau3.1. Ressources utilisées pour l'implémentation du DES-ECB

3.6.3. Mode DES-CBC

Les résultats de simulation en mode DES-CBC sont illustrés sur les figures 3.8 et 3.9

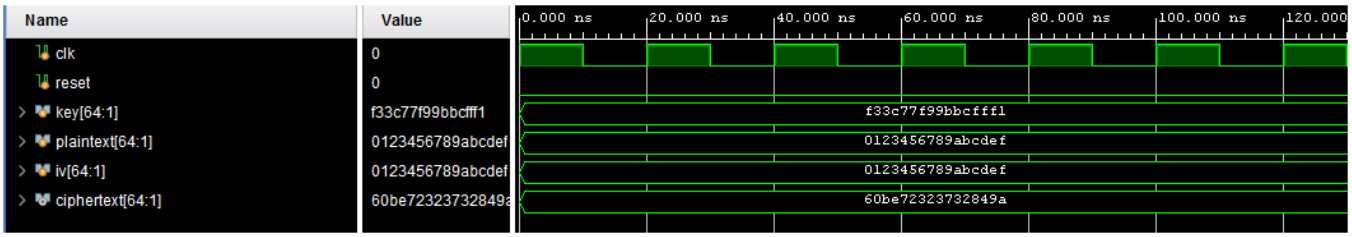

Figure.3.8. Résultat de simulation du chiffrement DES-CBC

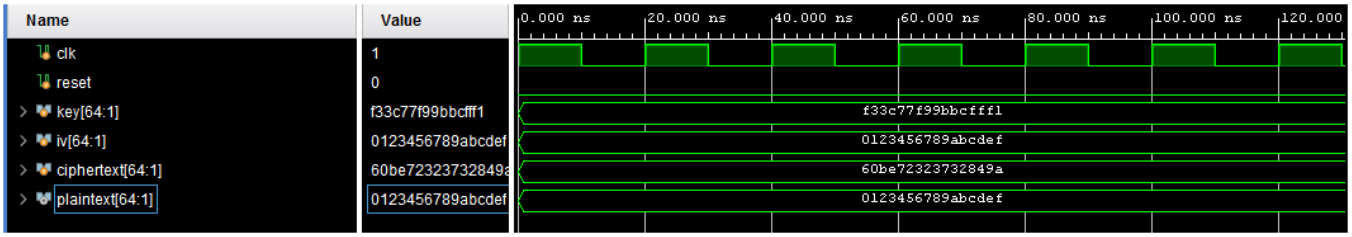

Figure.3.9. Résultat de simulation du déchiffrement DES-CBC

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.9. est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

Après la simulation comportementale, nous présentons sur le tableau 3.2. les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement DES-CBC			Déchiffrement DES-CBC		
	Utilisation	Disponible	%	Utilisation	Disponible	%
#LUT	1184	53200	2.2%	1200	53200	2.2%
#Slice Occupées	328	13300	2.3%	330	13300	2.3%
SLICEM	165			167		
SLICEL	163			163		
#BRAM	0	140	0%	0	140	0%
#DSP	0	220		0	220	
#Flip Flops	0	106400		0	106400	
Puissance (W)	0.863			0.863		
Temps Execution (ns)	1010			1010		

Tableau3.2. Ressources utilisées pour l'implémentation du DES-CBC

A partir de l'analyse des résultats de figures précédant (Figure3.6 à Figure3.9) et les tableaux, nous avons observé que :

- Le texte chiffré est différent malgré que la clé est la même dans les deux modes (ECB et CBC) parce qu'il y a une opération redondante en mode CBC qui est XOR.

- Une augmentation dans les ressources matérielles (slices (SLICEM et SLICEL) et les Lut) dans le mode CBC par rapports le mode ECB.

3.6.4. Mode DES-CFB

Les résultats de simulation en mode DES-CFB sont illustrés sur les figures 3.10 et 3.11.

Figure.3.10. Résultat de simulation du chiffrement DES-CFB

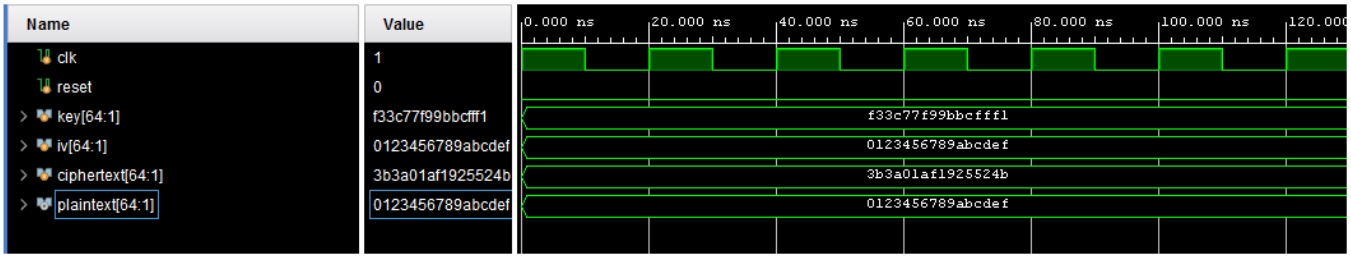

Figure.3.11. Résultat de simulation du déchiffrement DES-CFB

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.11. est identique à celui en chiffrement ce qui confirme l'exécution avec succés des opérations de chiffrement et de déchiffrement.

Après la simulation comportementale, nous présentons sur le tableau 3.3. Les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement DES-CFB			Déchiffrement DES-CFB		
	Utilisation	Disponible	%	Utilisation	Disponible	%
#LUT	8340	53200	16%	8324	53200	16%
#Slice Occupées	2285	13300	17%	2304	13300	17%
SLICEM	575			687		
SLICEL	1710			1617		
#BRAM	0	140	0%	0	140	0%
#DSP	0	220		0	220	
#Flip Flops	0	106400		0	106400	
Puissance (W)	0.863			8324	53200	16%
Temps Execution (ns)	1010			2304	13300	17%

Tableau3.3. Ressources utilisées pour l'implémentation du DES-CFB.

A partir de l'analyse des résultats nous avons observé que :

- Malgré que la clé est la même dans les trois modes (ECB, CBC et CFB) mais le texte chiffré est toujours différent, parce qu'il y a des opération redondante dans les modes CBC et CFB qui est XOR et le nombre d'algorithme DES.
- Une augmentation dans les ressources matérielles (slices (SLICEM et SLICEL) et les Lut), puis la puissance dans le mode CFB par rapports aux modes ECB et CBC.

3.6.5. Mode DES-OFB

Les résultats de simulation en mode DES-OFB sont illustrés sur les figures 3.12 et 3.13.

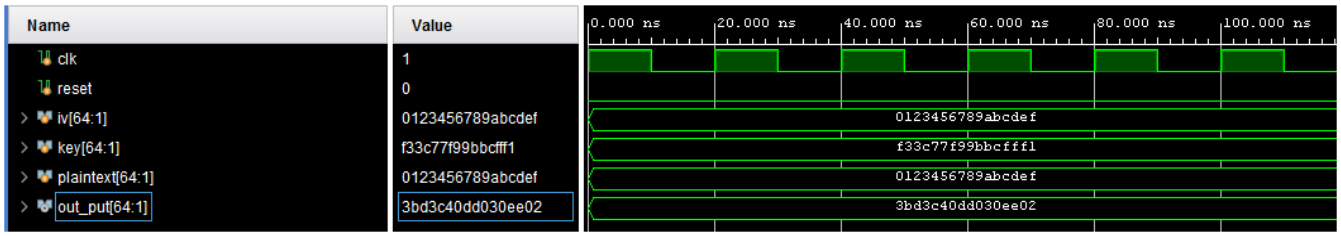

Figure.3.12. Résultat de simulation du chiffrement DES-OFB

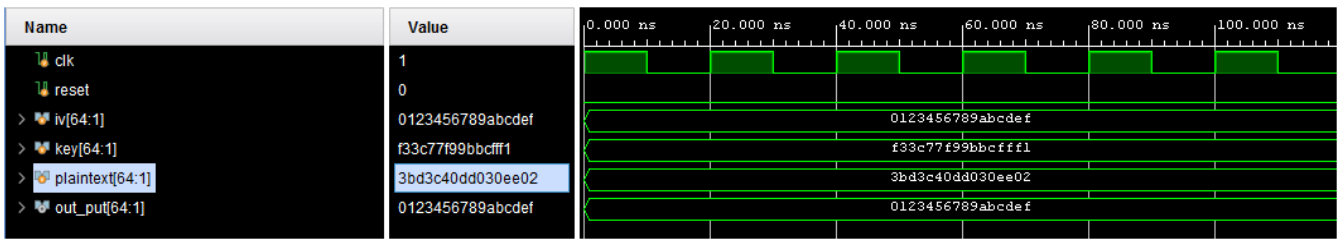

Figure.3.13. Résultat de simulation du déchiffrement DES-OFB

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous

remarquons que le texte déchiffrée « plaintext » sur la figure 3.13 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

Après la simulation comportementale, nous présentons sur le tableau 3.4 les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement/Déchiffrement DES-OFB		
	Utilisation	Disponible	%
#LUT	8663	53200	16%
#Slice Occupées	2380	13300	18%
SLICEM	522		
SLICEL	1858		
#BRAM	0	140	0%
#DSP	0	220	
#Flip Flops	0	106400	
Puissance (W)	0.863		
Temps Execution (ns)	1010		

Tableau3.4. Ressources utilisées pour l'implémentation du DES-OFB

Les résultats d'implémentation de L'algorithme DES en mode OFB de chiffrement et de déchiffrement sont identique. C'est pourquoi nous l'avons représenté avec un seul tableau représentant à la fois le chiffrement et le déchiffrement.

A partir de l'analyse des résultats, nous avons observé que :

- La clé est la même dans les quatre modes (ECB, CBC, CFB et OFB) mais que le texte chiffré est différent, parce qu'il y a des opérations redondantes dans les modes CBC, CFB et OFB qui est XOR et le nombre d'utilisation des algorithmes DES.
- Augmentation dans les ressources matérielles (slices (SLICEM et SLICEL) et les Lut).

3.7. Résultats de l'implémentation TDES

Nous avons conçu sur Vivado les modules TDES précédents en incluant tous les modes de chiffrements : ECB, CBC, OFB et CFB. Pour chaque mode, nous avons utilisés :

- Une clé de chiffrement « KEY1 » constante : $(F33C77F99BBCFFF1)_{16}$
- Une clé de chiffrement « KEY2 » constante : $(F33C77F99BBCFFF1)_{16}$
- Une clé de chiffrement « KEY3 » constante : $(F33C77F99BBCFFF1)_{16}$
- Un texte en clair « plaintext » constant : $(0123456789ABCDEF)_{16}$

Nous avons effectué une simulation comportementale pour chaque mode afin de vérifier si l'exécution du chiffrement est correcte en comparant le texte crypté « ciphertext » obtenu en simulation Vivado avec celui obtenu en simulation MATLAB et sur d'autres outils de chiffrement tiers.

Enfin, pour chaque mode de chiffrement nous avons pu synthétiser et implémenter et générer le bitstream de notre conception DES et le charger sur la carte cible PYNQ-Z2 en programmant un banc d'essai qui comprend un bouton de lancement du chiffrement symétrique TDES et une LED témoin qui indique l'état du chiffrement.

3.7.1. Schéma RTL

Nous avons généré l'analyse RTL (Register Transfer Level) sur Vivado afin de visualiser le schéma de représentation du circuit de génération des sous clés TDES commun à tous les modes de chiffrement comme le montre la figure 3.14.

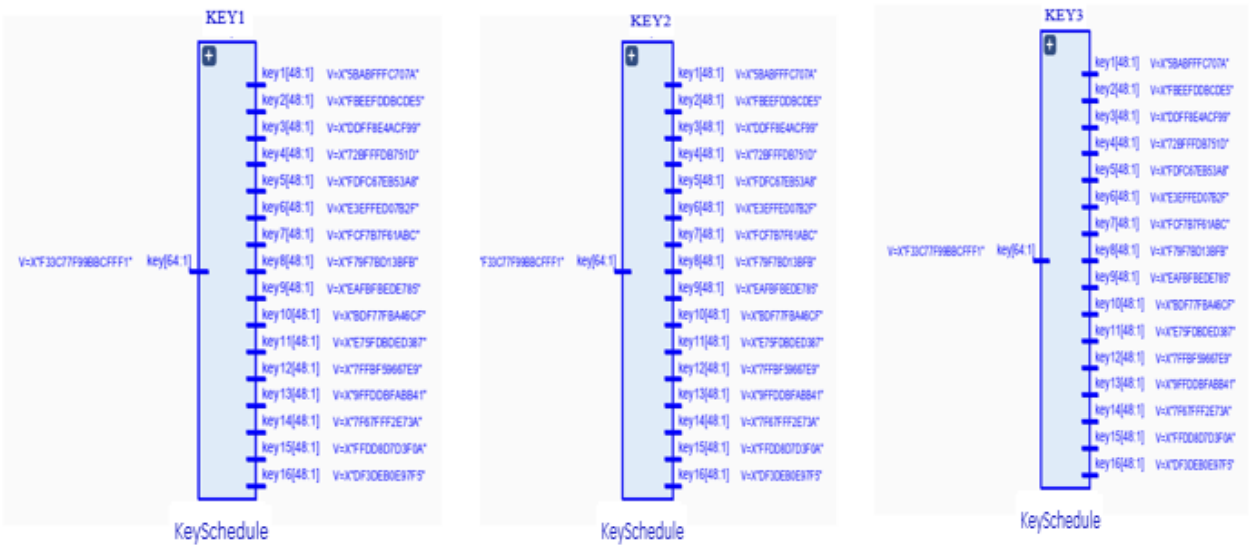

Figure 3.14. Schéma RTL du circuit de génération des sous clés TDES

3.7.2. Mode TDES-ECB

Les résultats de simulation en mode TDES-ECB sont illustrés sur les figures 3.15 et 3.16.

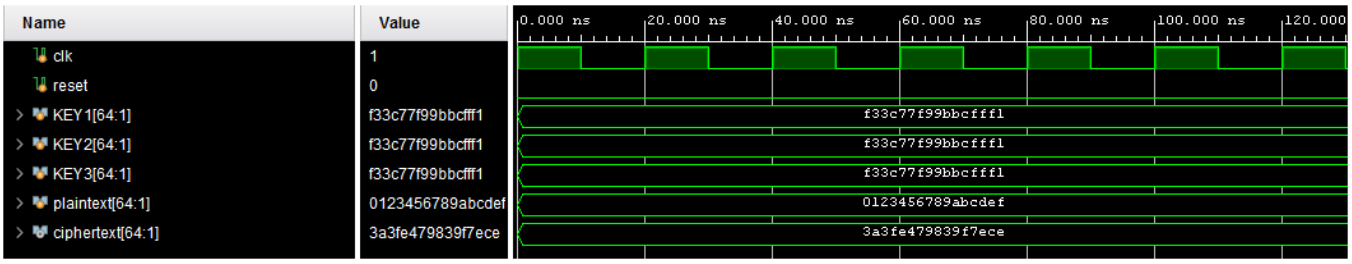

Figure.3.15. Résultat de simulation du chiffrement TDES-ECB

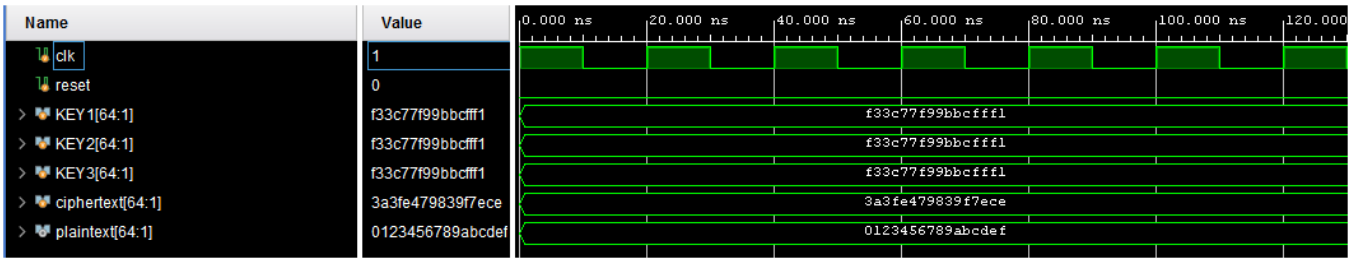

Figure.3.16. Résultat de simulation du chiffrement TDES-ECB

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.16 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

Après la simulation comportementale, nous présentons sur le tableau 3.5 les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement TDES-ECB			Déchiffrement TDES-ECB		
	Utilisation	Disponible	%	Utilisation	Disponible	%
#LUT	3588	53200	7%	3589	53200	7%
#Slice Occupées	982	13300	7%	993	13300	7%
SLICEM	336			370		
SLICEL	646			623		
#BRAM	0	140	0%	0	140	0%
#DSP	0	220		0	220	
#Flip Flops	0	106400		0	106400	
Puissance (W)	0.863			0.863		
Temps Execution (ns)	1010			1010		

Tableau3.5. Ressources utilisées pour l'implémentation du TDES-ECB

3.7.3. Mode TDES-CBC

Les résultats de simulation en mode TDES-CBC sont illustrés sur les figures 3.17 et 3.18.

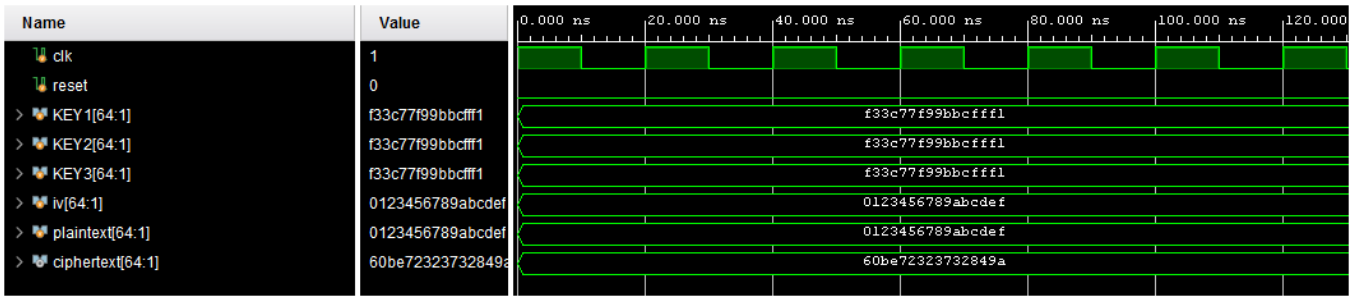

Figure.3.17. Résultat de simulation du chiffrement TDES-CBC

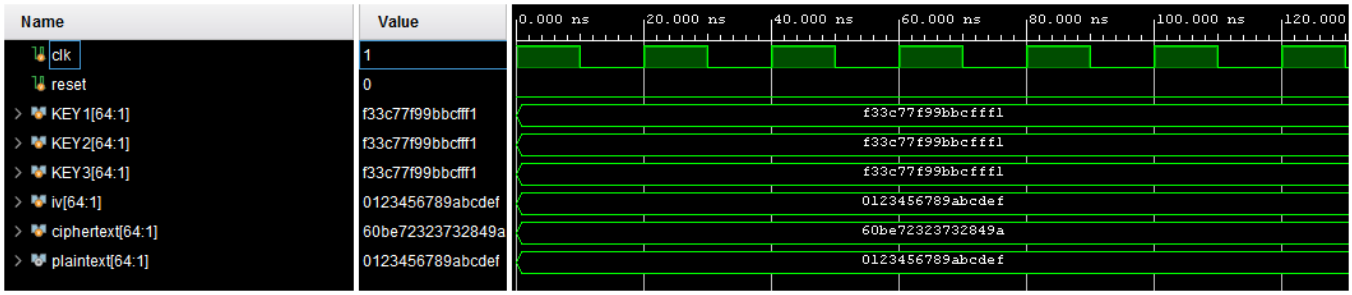

Figure.3.18. Résultat de simulation du déchiffrement TDES-CBC

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.18 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

Après la simulation comportementale, nous présentons sur le tableau 3.6 les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement TDES-CBC			Déchiffrement TDES-CBC		
	Utilisation	Disponible	%	Utilisation	Disponible	%
#LUT	3598	53200	7%	3561	53200	7%
#Slice Occupées	993	13300	7%	975	13300	7%
SLICEM	333			354		
SLICEL	660			621		
#BRAM	0	140	0%	0	140	0%
#DSP	0	220		0	220	
#Flip Flops	0	106400		0	106400	
Puissance (W)	0.863			0.863		
Temps Execution (ns)	1010			1010		

Tableau3.6. Ressources utilisées pour l'implémentation du TDES-CBC

A partir de l'analyse des résultats nous avons observé que :

- Malgré que la clé est la même dans les deux modes (ECB et CBC) mais que le texte chiffré est différent, parce qu'il y a une opération redondante en mode CBC qui est XOR.
- Augmentation dans les ressources matérielles (slices (SLICEM et SLICEL) et les Lut) dans le mode CBC par rapports le mode ECB.

3.7.4. Mode TDES-CFB

Les résultats de simulation en mode TDES-CFB sont illustrés sur les figures 3.19 et 3.20.

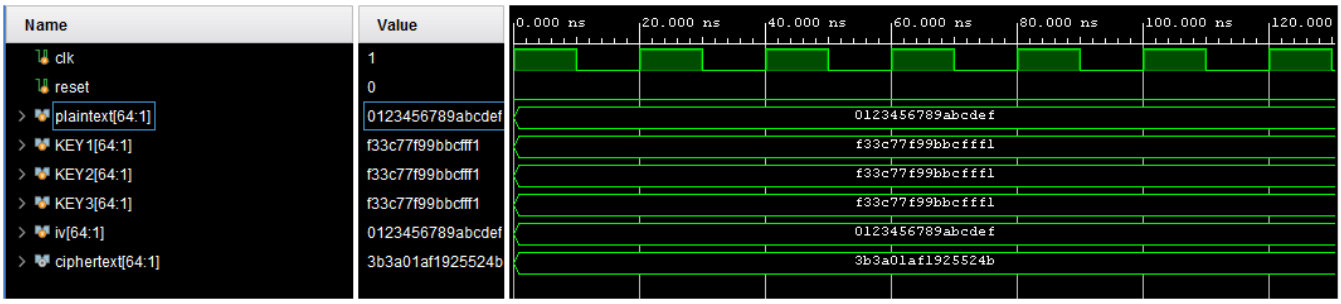

Figure.3.19. Résultat de simulation du chiffrement TDES-CFB

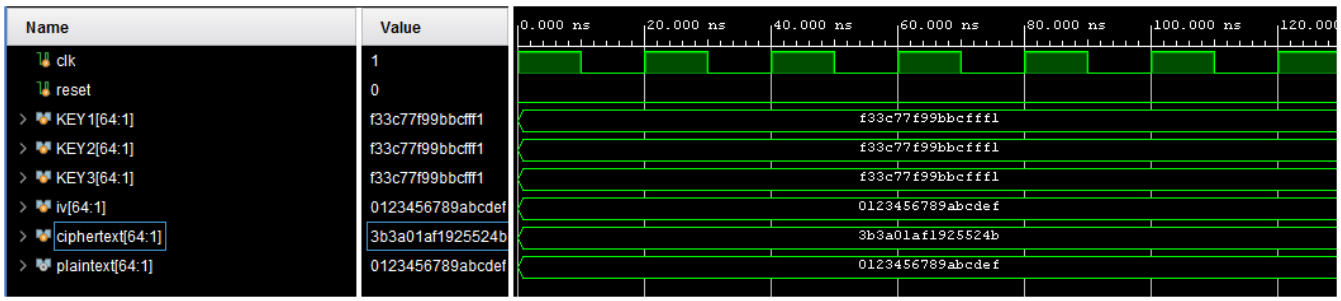

Figure.3.20. Résultat de simulation du déchiffrement TDES-CFB

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.20 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

Après la simulation comportementale, nous présentons sur le tableau 3.7 Les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement TDES-CFB			Déchiffrement TDES-CFB		
	Utilisation	Disponible	%	Utilisation	Disponible	%
#LUT	24995	53200	47%	24926	53200	47%
#Slice Occupées	6740	13300	51%	6730	13300	51%
SLICEM	2213			2211		
SLICEL	4527			4519		
#BRAM	0	140	0%	0	140	0%
#DSP	0	220		0	220	
#Flip Flops	0	106400		0	106400	
Puissance (W)	0.863			0.863		
Temps Execution (ns)	1010			1010		

Tableau3.7. Ressources utilisées pour l'implémentation du TDES-CFB.

A partir de l'analyse des résultats de figures précédant (Figure3.15 à Figure3.20) et les tableaux (tableau3.5 à tableau3.7), nous avons observé que :

- La clé est la même dans les trois modes (ECB, CBC et CFB) mais le texte chiffré est différent, parce qu'il y a des opérations redondantes dans les modes CBC et CFB qui est XOR et le nombre d'utilisation de l'algorithme TDES.
- Augmentation dans les ressources matérielles (slices (SLICEM et SLICEL) et les Lut.

3.7.5. Mode TDES-OFB

Les résultats de simulation en mode TDES-OFB sont illustrés sur les figures 3.21 et 3.22.

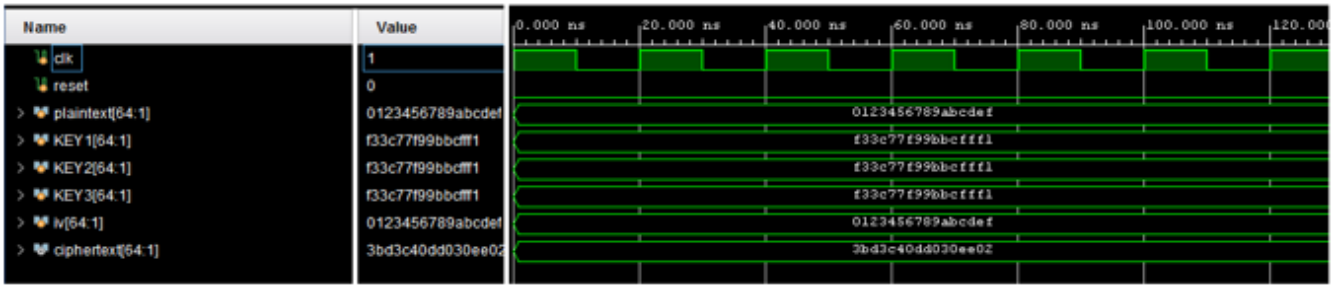

Figure.3.21. Résultat de simulation du chiffrement TDES-OFB

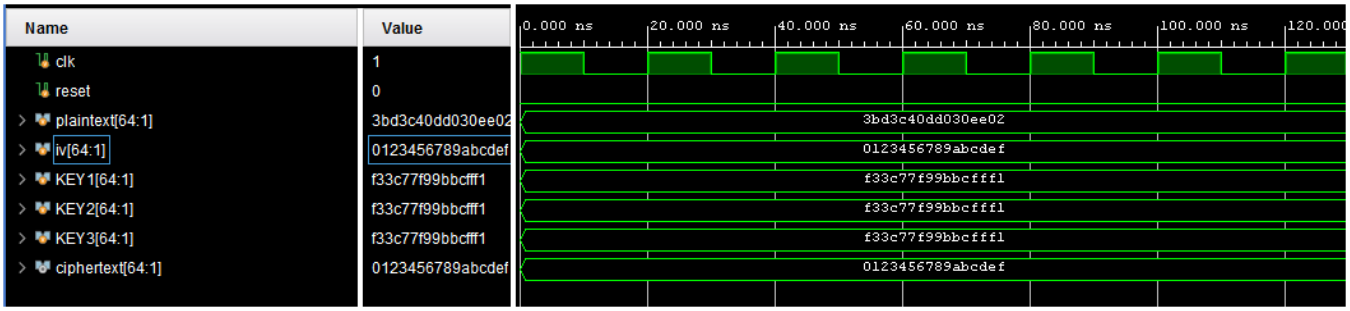

Figure.3.22. Résultat de simulation du déchiffrement TDES-OFB

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.22 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

Après la simulation comportementale, nous présentons sur le tableau 3.8 les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement/Déchiffrement TDES-OFB		
	Utilisation	Disponible	%
#LUT	25806	53200	49%
#Slice Occupées	7028	13300	53%
SLICEM	2278		
SLICEL	4750		
#BRAM	0	140	0%
#DSP	0	220	
#Flip Flops	0	106400	
Puissance (W)	0.863		
Temps Execution (ns)	1010		

Tableau3.8. Ressources utilisées pour l'implémentation du TDES-OFB.

Les résultats d'implémentation de L'algorithme TDES en mode OFB de chiffrement et de déchiffrement sont identique. C'est pourquoi nous l'avons représenté avec un seul tableau représentant à la fois le chiffrement et le déchiffrement.

A partir de l'analyse des résultats nous avons observé que :

- La clé est la même dans les trois modes (ECB, CBC et CFB) mais que le texte chiffré est différent, parce qu'il y a des opérations redondantes dans les modes CBC et CFB qui est XOR et le nombre d'utilisation de l'algorithme TDES.
- Une augmentation dans les ressources matérielles (slices (SLICEM et SLICEL) et les Lut).

3.8. Résultats de l'implémentation AES

Nous avons conçu sur Vivado les modules AES précédents en incluant tous les modes de chiffrements : ECB, CBC, OFB et CFB. Pour chaque mode, nous avons utilisés :

- Une clé de chiffrement « key » constante : $(0000000000000000F33C77F99BBCFFF1)_{16}$
- Un texte en clair « plaintext » constant : $(00000000000000000123456789ABCDEF)_{16}$

Nous avons effectué une simulation comportementale pour chaque mode afin de vérifier si l'exécution du chiffrement est correcte en comparant le texte chiffré « ciphertext » obtenu en simulation Vivado avec celui obtenu en simulation MATLAB et sur d'autres outils de chiffrement tiers.

Enfin, pour chaque mode de chiffrement nous avons pu synthétiser et implémenter et générer le bitstream de notre conception AES et le charger sur la carte cible PYNQ-Z2 en programmant un banc d'essai qui comprend un bouton de lancement du chiffrement symétrique AES et une LED témoin qui indique l'état du chiffrement.

3.8.1. Schéma RTL

Nous avons généré l'analyse RTL sur Vivado afin de visualiser le schéma de représentation du circuit de génération des sous clés AES commun à tous les modes de chiffrement comme le montre la figure 3.14. En parcourant le schéma RTL, on remarque que la boite noire implémentée par notre algorithme ne représente que le nombre d'entrées/sorties (1 entrée « key » et 44 sorties « w1, …, w44 » chaque 4 w représenté une clé donc il y a 11 clés, une clé d'origine et 10 clés). Comme montre la figure 3.23. Après la vérification des résultats, nous avons remarqué que les sous clés de l'algorithme DES dans VIVADO sont les même sous clés obtenus dans les simulations MATLAB donc complètement corrects.

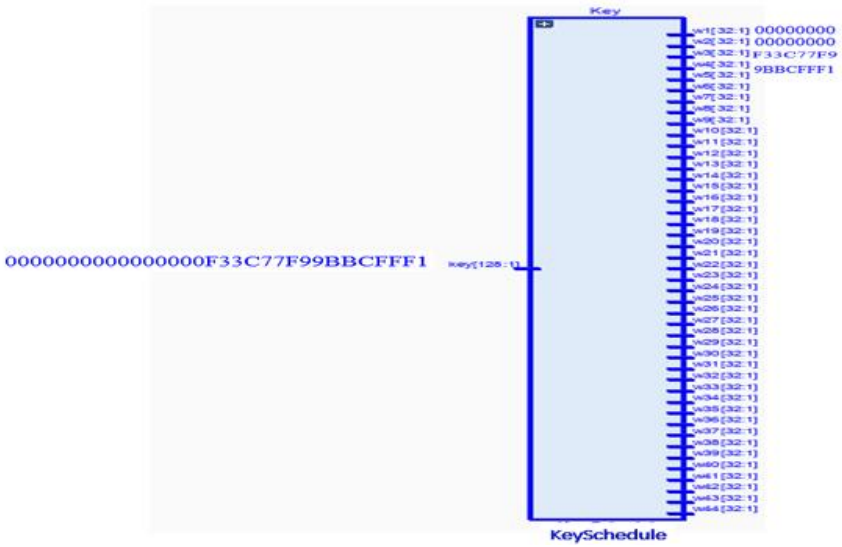

Figure3.23. Schéma RTL du circuit de génération des sous clés AES

3.8.2. Mode AES-ECB

Les résultats de simulation en mode AES-ECB sont illustrés sur les figures 3.24 et 3.25.

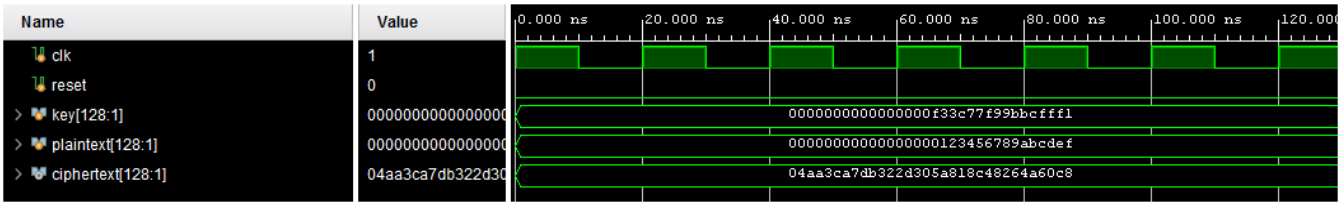

Figure.3.24. Résultat de simulation du chiffrement AES-ECB

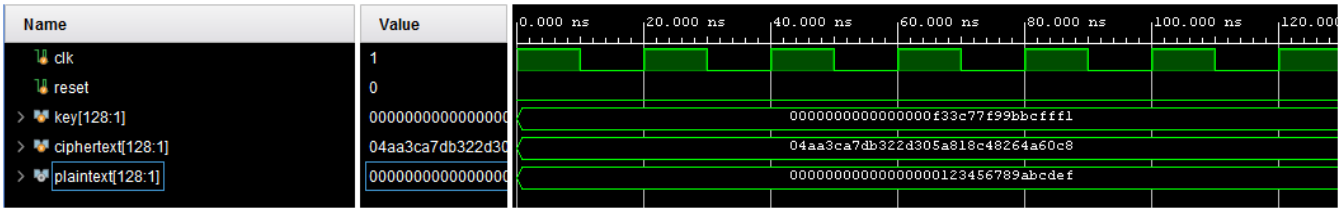

Figure.3.25. Résultat de simulation du déchiffrement AES-ECB

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.25 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

Après la simulation comportementale, nous présentons sur le tableau 3.9 les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement AES-ECB			Déchiffrement AES-ECB		
	Utilisation	Disponible	%	Utilisation	Disponible	%
#LUT	10243	53200	19%	12375	53200	23%
#Slice Occupées	2944	13300	22%	3456	13300	26%
SLICEM	721			929		
SLICEL	2223			2527		
#BRAM	0	140	0%	0	140	0%
#DSP	0	220		0	220	
#Flip Flops	0	106400		0	106400	
Puissance (W)	0.863			0.863		
Temps Execution (ns)	1010			1010		

Tableau3.9. Ressources utilisées pour l'implémentation du AES-ECB

3.8.3. Mode AES-CBC

Les résultats de simulation en mode AES-CBC sont illustrés sur les figures 3.26 et 3.27.

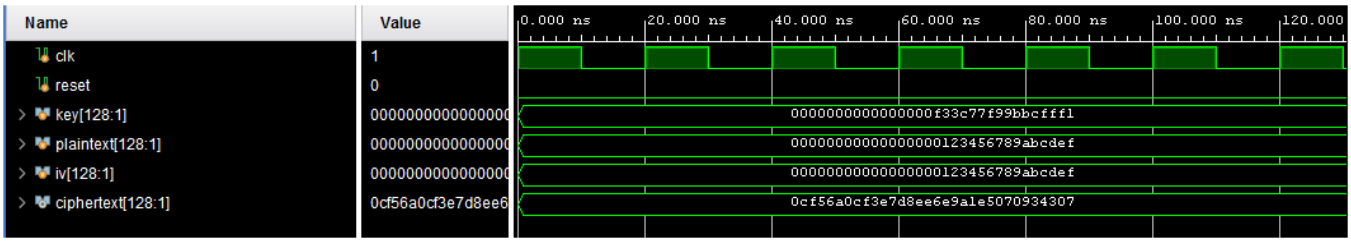

Figure.3.26. Résultat de simulation du chiffrement AES-CBC

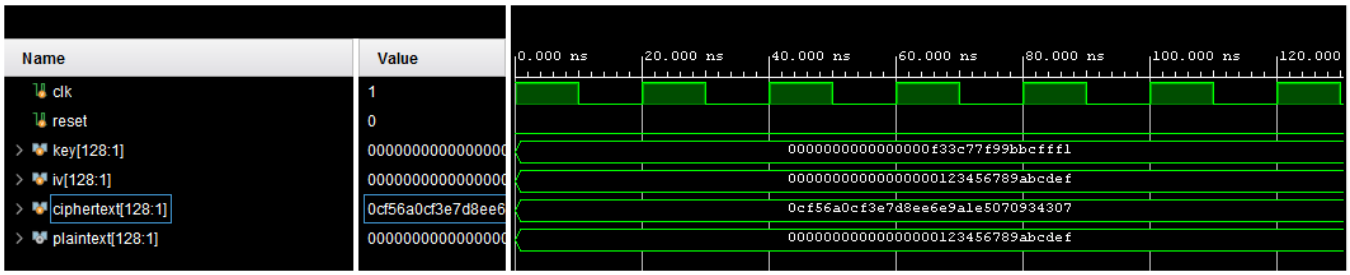

Figure.3.27. Résultat de simulation du déchiffrement AES-CBC

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.27 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

Après la simulation, nous présentons sur le tableau 3.10 Les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement AES-CBC			Déchiffrement AES-CBC		
	Utilisation	Disponible	%	Utilisation	Disponible	%
#LUT	10235	53200	19%	11856	53200	22%
#Slice Occupées	2891	13300	22%	3372	13300	25%
SLICEM	762			919		
SLICEL	2129			2453		
#BRAM	0	140	0%	0	140	0%
#DSP	0	220		0	220	
#Flip Flops	0	106400		0	106400	
Puissance (W)	0.863			0.863		
Temps Execution (ns)	1010			1010		

Tableau3.10. Ressources utilisées pour l'implémentation du AES-CBC.

A partir de l'analyse des résultats nous avons observé que :

- Dans la simulation du programme AES, nous remarquons malgré que la clé est la même dans les deux modes (ECB et CBC) mais que le texte chiffré est différent, parce qu'il y a des opération redondante en mode CBC qui est le nombre des XOR et le nombre d'utilisation de l'algorithme AES.
- Après l'analyse des tableaux nous remarquons qu'il y a une augmentation dans les ressources matérielles (slices (SLICEM et SLICEL) et les Lut) dans le mode CBC par rapports le mode ECB.

3.8.4. Mode AES-CFB

Les résultats de simulation en mode AES-CFB sont illustrés sur les figures 3.28 et 3.29.

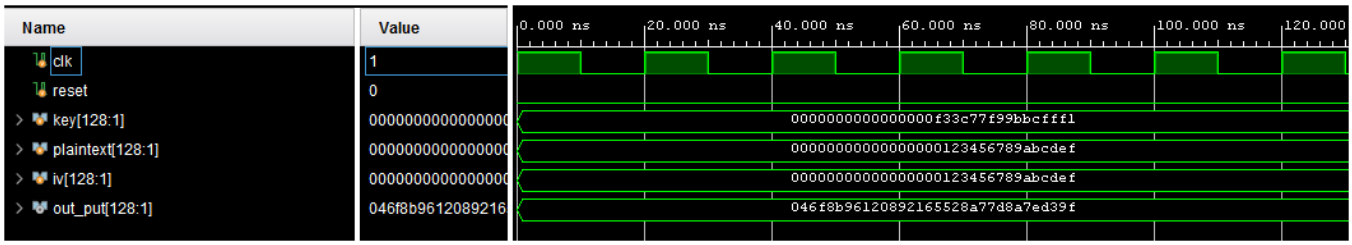

Figure.3.28. Résultat de simulation du chiffrement AES-CFB

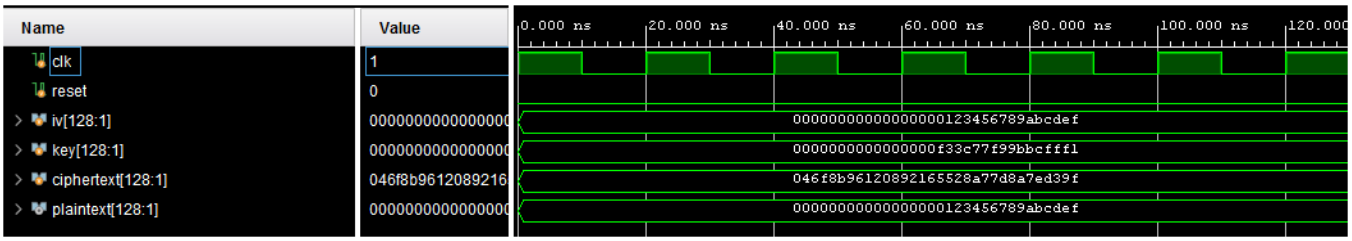

Figure.3.29. Résultat de simulation du déchiffrement AES-CFB

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. La clé (key). Le banc fournit un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.29 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement

Après la simulation comportementale, nous présentons sur le tableau 3.11 les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

#Ressources	Chiffrement/Déchiffrement AES-CFB		
	Utilisation	Disponible	%
#LUT	10273	53200	19%
#Slice Occupées	2887	13300	22%
SLICEM	733		
SLICEL	2154		
#BRAM	0	140	0%
#DSP	0	220	
#Flip Flops	0	106400	
Puissance (W)	0.863		
Temps Execution (ns)	1010		

Tableau3.11. Ressources utilisées pour l'implémentation du AES-CFB.

Les résultats d'implémentation de L'algorithme AES en mode CFB de chiffrement et de déchiffrement sur 128 bits sont identique. C'est pourquoi nous l'avons représenté avec un seul tableau représentant à la fois le chiffrement et le déchiffrement. A partir de l'analyse des résultats nous avons observé que :

- Malgré que la clé est la même dans les trois modes (ECB, CBC et CFB) mais que le texte chiffré est différent, parce qu'il y a des opération redondante dans les modes CBC et CFB qui est XOR et le nombre d'utilisation de l'algorithme AES.
- Augmentation dans les ressources matérielles (slices (SLICEM et SLICEL) et les Lut).

3.8.5. Mode AES-OFB

Les résultats de simulation en mode AES-OFB sont illustrés sur les figures 3.30 et 3.31.

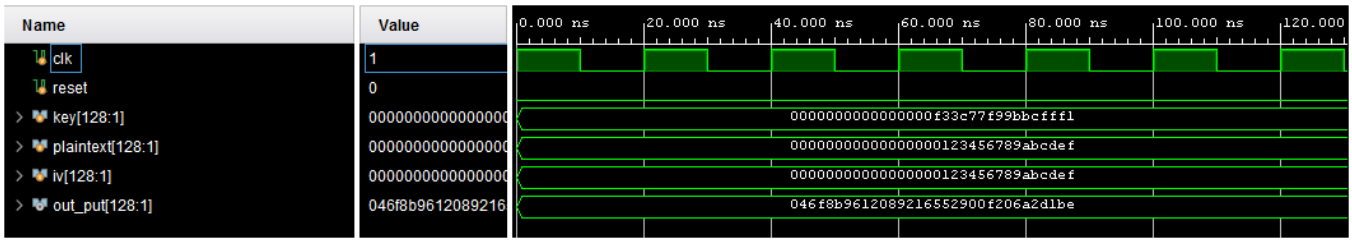

Figure.3.30. Résultat de simulation du chiffrement AES-OFB

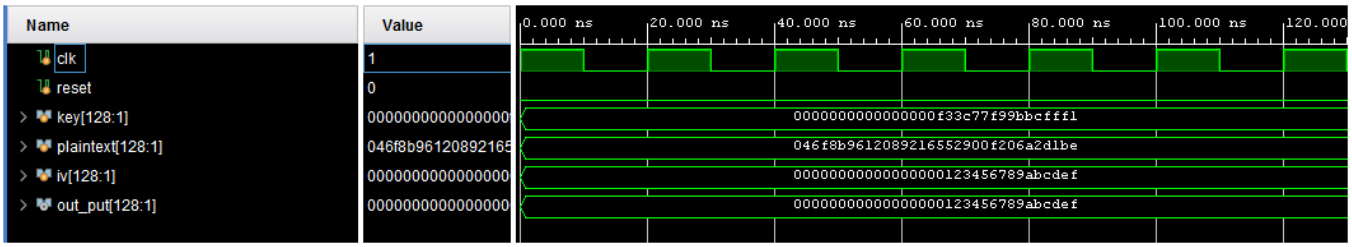

Figure.3.31. Résultat de simulation du déchiffrement AES-OFB

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 101 cycles d'horloge, et compare la sortie finale avec le chiffrement associé. Nous remarquons que le texte déchiffrée « plaintext » sur la figure 3.31 est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

Après la simulation comportementale, nous présentons sur le tableau 3.12 Les résultats des rapports de synthèse et d'implémentation de notre IP en donnant une estimation du nombre (#) de ressources utilisées.

Les résultats d'implémentation de L'algorithme AES en mode CFB de chiffrement et de déchiffrement sur 128 bits sont identique. C'est pourquoi nous l'avons représenté avec un seul tableau représentant à la fois le chiffrement et le déchiffrement.

#Ressources	Chiffrement/Déchiffrement AES-OFB		
	Utilisation	Disponible	%
#LUT	10273	53200	19%
#Slice Occupées	2887	13300	22%
SLICEM	733		
SLICEL	2154		
#BRAM	0	140	0%
#DSP	0	220	
#Flip Flops	0	106400	
Puissance (W)	0.863		
Temps Execution (ns)	1010		

Tableau3.12. Ressources utilisées pour l'implémentation du AES-OFB.

A partir de l'analyse des résultats, nous avons observé que :

- La clé est la même dans les trois modes (ECB, CBC, CFB et OFB) mais que le texte chiffré est différent, parce qu'il y a des opérations redondantes dans les modes CBC et CFB qui est XOR et le nombre d'utilisation de l'algorithme AES.
- Augmentation dans les ressources matérielles (slices (SLICEM et SLICEL) et les Lut).

3.9. Application en chiffrement d'images

A partir de l'analyse des résultats de simulation, synthèses et l'implémentation des algorithmes (DES, AES et TDES) , nous avons observé que :

- Par l'implémentation sur les algorithmes de chiffrement par bloc (DES, TDES et AES) on trouve que la premier algorithme (DES) consomme moins de ressources matérielles et la deuxième (AES) est le plus rapide possible que le premier algorithme sur FPGA
- Nous constatons également que la valeur de la puissance est presque constante pour tous les.

De ce fait, nous avons décidé d'appliquer notre conception pour chiffrer une image de test standard CAMERAMAN 32×32 de niveau de gris 8 bits et de format PNG en utilisant les trois conceptions précédentes (DES, AES et TDES). Nous avons converti l'image en un fichier binaire en utilisant un script MATLAB ensuite nous avons utilisé ce fichier comme entrée à notre application sur VIVADO pour une implémentation sur le SoC-FPGA. Enfin, nous utilisant MATLAB pour afficher l'image chiffrée/déchiffrée sur le FPGA.

Pour faciliter la simulation de la conception, nous avons conçu sur VIVADO un banc de test utilisant une couche supérieure (aes_ecb ou des_ecb) qui reçoit des vecteurs de simulation prédéfinis (img, plaintext, key et horloge) et vérifie que la sortie (ciphertext) correspond bien à la sortie attendue. Le banc fournit en fait un nouveau texte en clair et une nouvelle clé tous les 2 cycles d'horloges.

3.9.1. Résultats de simulation en DES

La figure 3.32 montre les résultats de simulation du chiffrement d'une image de niveau de gris 32x32 en utilisant un chiffrement symétrique DES-ECB.

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. « img» représente l'image chiffrée d'une taille (64× 128), à l'origine d'une taille (32×32) ou 1024 x 8bits.

L'algorithme DES ne chiffre que 64 bits, ce qui veut dire qu'on va utiliser 128 lignes de 64 bits. La figure3.32 montre les résultats que nous attendions.

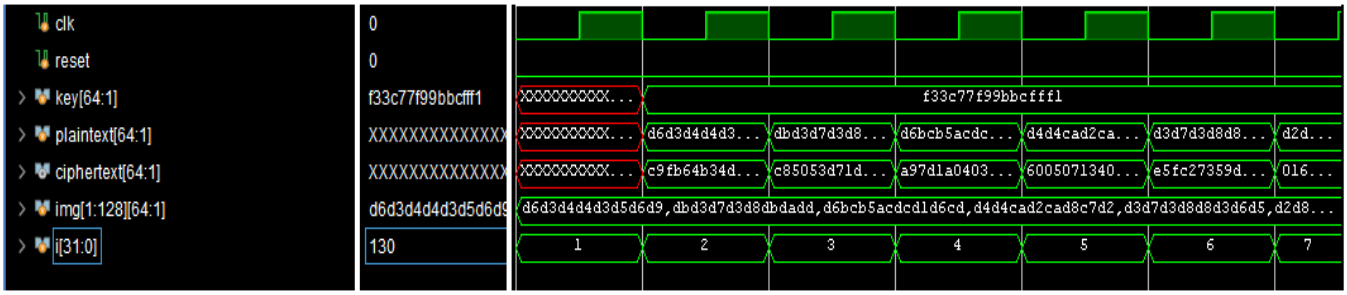

Figure.3.32. Résultat de simulation du chiffrement DES-ECB

Après le chiffrement de l'algorithme DES en mode ECB, nous présentons la transformation inverse du chiffrement et la même architecture que celle du chiffrement a été appliquée.

La figure3.33 montre que le texte déchiffrée « plaintext » est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

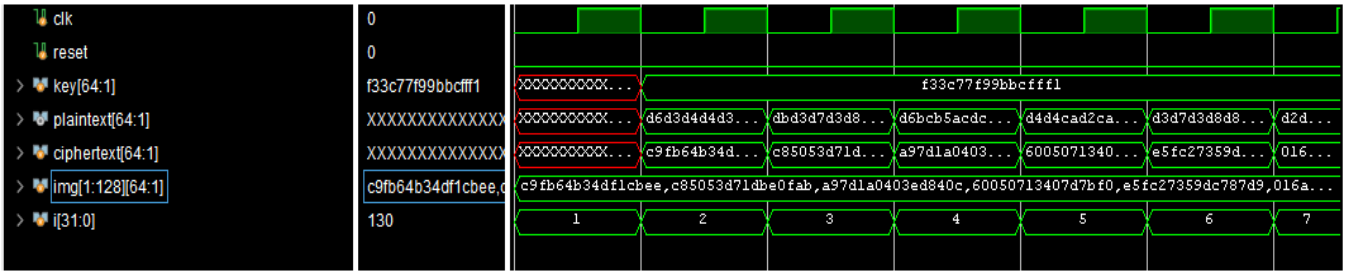

Figure.3.33. Résultat de simulation de déchiffrement DES-ECB

Pour montrer la sécurité et l'efficacité de l'algorithme DES, nous avons changé l'octet MSB de la clé de déchiffrement (F3 → 9E), les résultats de simulation sont illustrés dans la figure 3. 34. D'après la figure, nous remarquons qu'avec la clé modifiée nous obtenons une image déchiffrée « plaintext » complètement différente de l'image correctement déchiffrée « plaintext » sur la figure 3.32 précédente.

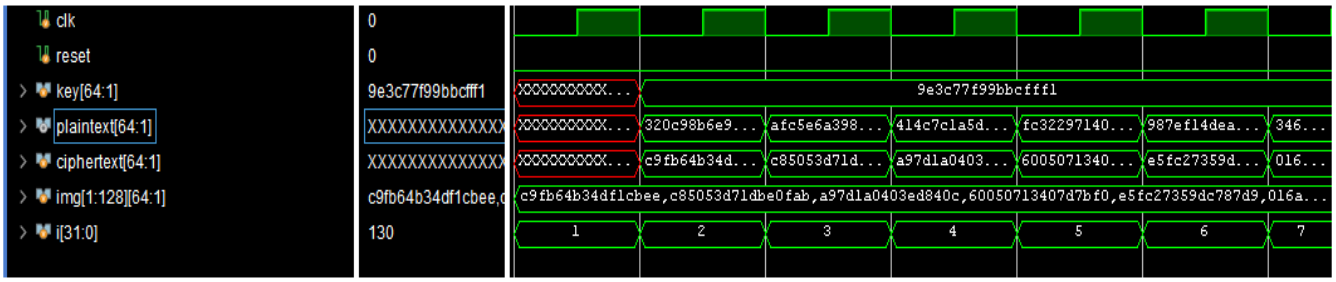

Figure.3.34. Résultat de simulation du déchiffrement avec une erreur en DES-ECB

3.9.2. Résultats de simulation en TDES

La figure 3.35 montre les résultats de simulation du chiffrement d'une image de niveau de gris 32x32 en utilisant un chiffrement symétrique TDES-ECB.

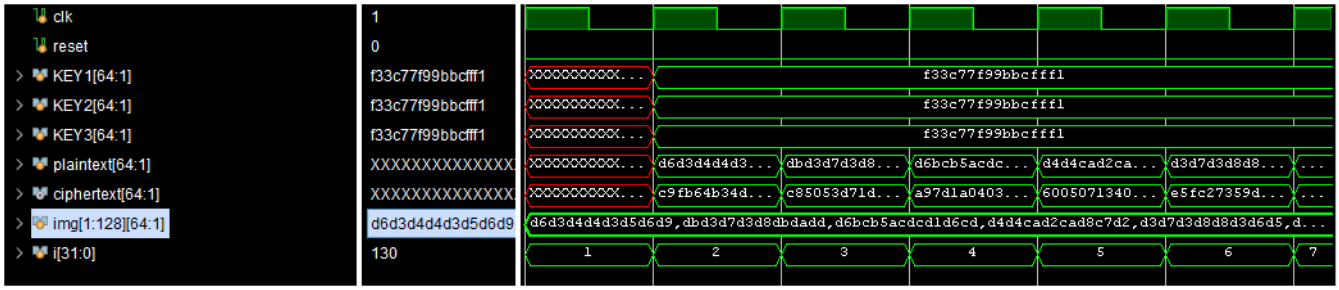

Figure.3.35. Résultat de simulation du chiffrement TDES-ECB

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. « Img» représente l'image chiffrée d'une taille (64× 128), à l'origine d'une taille (32×32) ou 1024 x 8bits.

L'algorithme DES ne chiffre que 64 bits, ce qui veut dire qu'on va utiliser 128 lignes de 64 bits. La figure3.35. Montre que les résultats Ce chiffre correspond à ce que nous attendions

Après le chiffrement de l'algorithme TDES en mode ECB, nous présentons la transformation inverse du chiffrement et la même architecture que celle du chiffrement a été appliquée. La figure3.36 montre que le texte déchiffrée « plaintext » est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

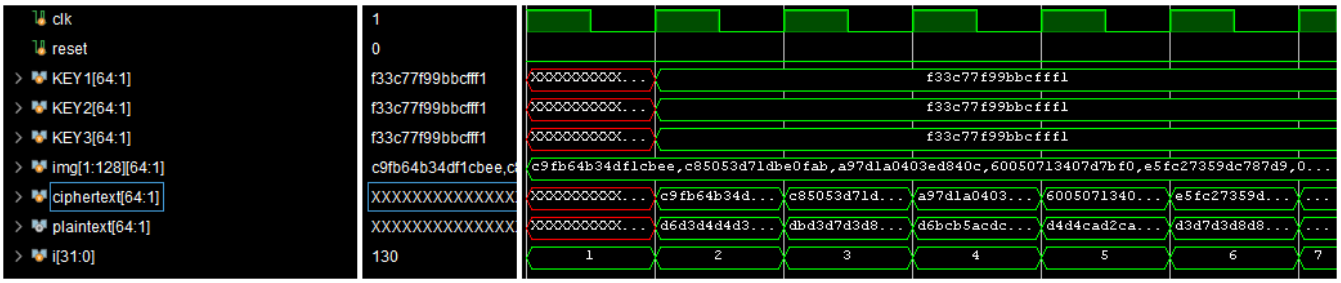

Figure.3.36. Résultat de simulation de déchiffrement TDES-ECB

Pour montrer la sécurité et l'efficacité de l'algorithme TDES, nous changeons les 8 bits (1octet) MSB de la clé de déchiffrement (F3 → 9E), les résultats de simulation sont illustrés dans la figure 3. 37. D'après la figure, nous remarquons qu'avec la clé modifiée nous obtenons une image déchiffrée « plaintext » complètement différente de l'image correctement déchiffrée « plaintext » sur la figure 3.35 précédente.

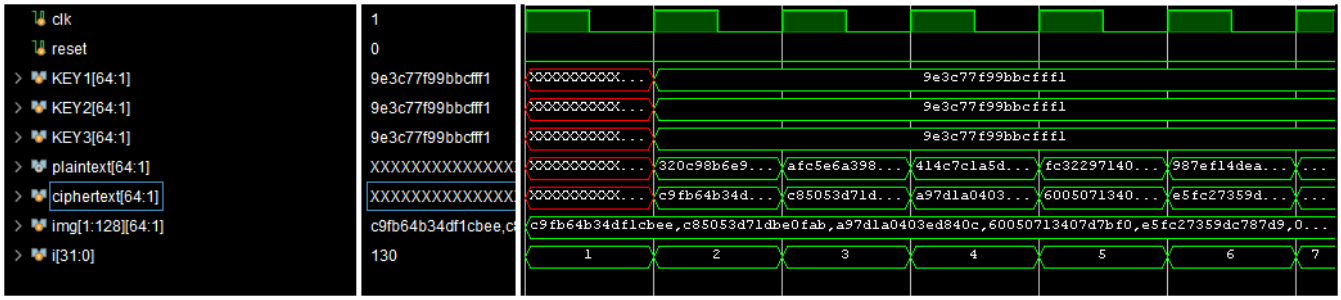

Figure.3.37. Résultat de simulation de déchiffrement avec une erreur en TDES-ECB

3.9.3. Résultats de simulation en AES

La figure 3.32 montre les résultats de simulation du chiffrement d'une image de niveau de gris 32x32 en utilisant un chiffrement symétrique AES-ECB.

« clk » représente le signal horloge, « reset » le signal de remise à zéro, « key » la clé secrète de chiffrement et de déchiffrement. « img» représente l'image chiffrer d'une taille (64× 128), à l'origine d'une taille (32×32) ou 1024 bits.

L'algorithme AES ne chiffre que 128 bits, ce qui veut dire qu'on va utiliser 64 lignes de 128 bits. La figure3.38 Montre que les résultats Ce chiffre correspond à ce que nous attendions.

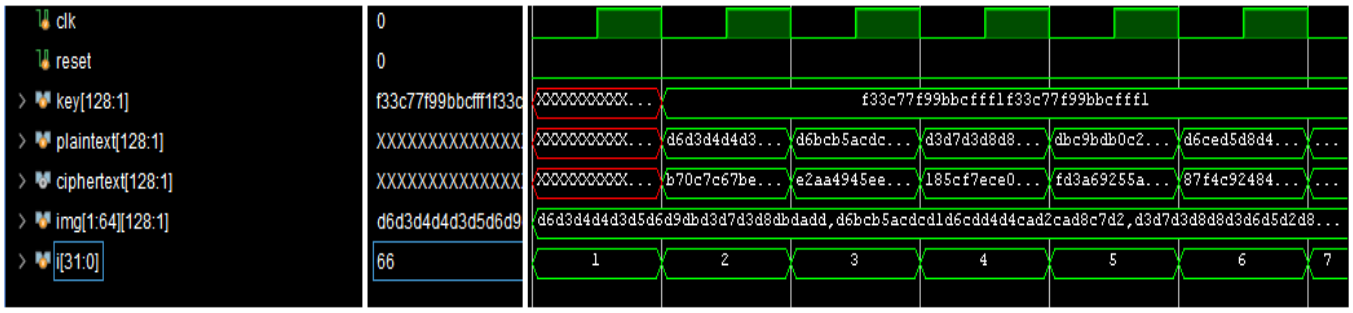

Figure.3.38. Résultat de simulation de chiffrement AES-ECB

Après le chiffrement de l'algorithme AES en mode ECB, nous présentons la transformation inverse du chiffrement et la même architecture que celle du chiffrement a été appliquée. La figure3.39 montre que le texte déchiffrée « plaintext » est identique à celui en chiffrement ce qui confirme l'exécution avec succès des opérations de chiffrement et de déchiffrement.

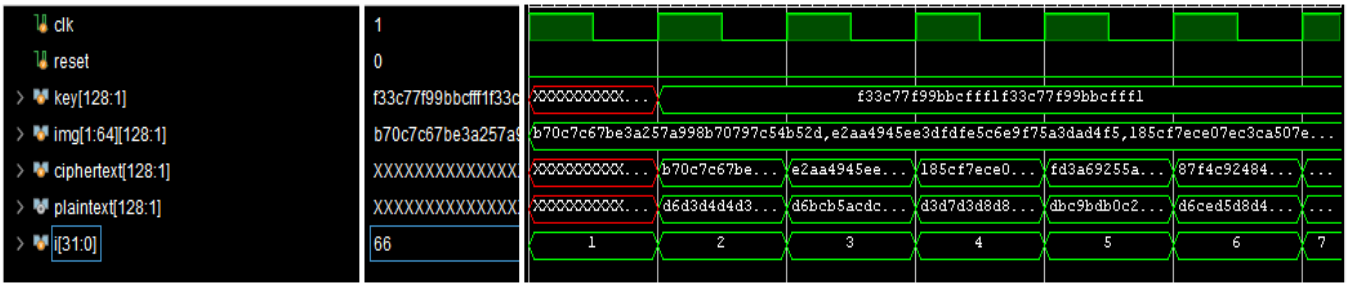

Figure.3.39. Résultat de simulation de déchiffrement AES-ECB

Pour montrer la sécurité et l'efficacité de l'algorithme AES, nous changeons les 8 bits (1octet) de la clé de déchiffrement (F3 → 9E), les résultats de simulation sont illustrés dans la figure 3. 40. D'après la figure, nous remarquons qu'avec la clé modifiée nous obtenons une image déchiffrée « plaintext » complètement différente de l'image correctement déchiffrée « plaintext » sur la figure 3.38 précédente.

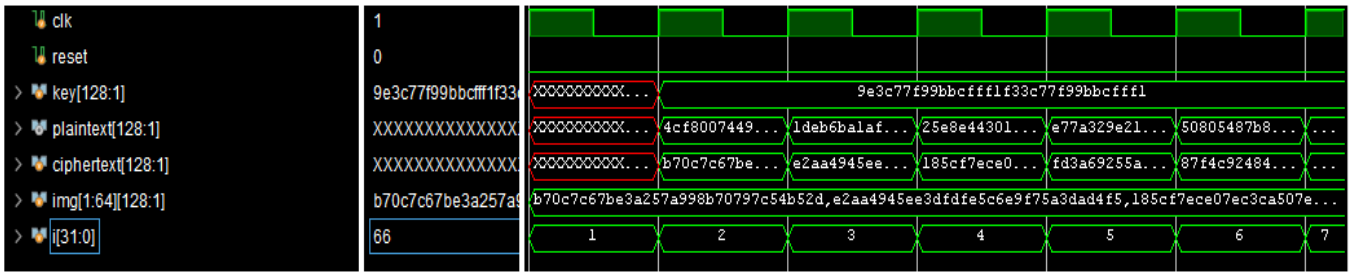

Figure.3.40. Résultat de simulation de déchiffrement avec une erreur en AES-ECB

3.9.4. Implémentation, comparaison et discussion

La figure 3.41 montre les résultats des implémentation obtenus sur VIVADO. Ces résultats ont été transférés pour affichage avec la fonction prédéfinie de MATLAB imshow.

a)

b)

c)

Figure 3.41. Résultats de l'implémentation : a) DES-ECB, b) AES-ECB, c) TDES-ECB

Nous avons également comparé les performances de nos trois implémentations sur SoC-FPGA Zynq 7000 en termes de rapidité d'exécution avec ceux réalisés sur MATLAB et un PC avec un CPU Intel I5-4310U@2.00GHz (4 COEURS) RAM 4Go. Les résultats sont illustrés sur le tableau 3.13.

Temps Exécution	MATLAB		SoC-FPGA (estimation en simulation)	
	Chiffrement	Déchiffrement	Chiffrement	Déchiffrement
DES	8.356s	8.390s	2,580 ns	2,580 ns
AES	1.393 s	2.165s	1,300 ns	1,300 ns
TDES	25.662 s	26.953s	2,580 ns	2,580 ns

Tableau 3.13. Comparaison des performances AES, DES et TDES entre MATLAB et le SoC-FPGA

Dans le cas du DES, dans les deux implémentations nous avons pris comme clé : $(\text{F33C77F99BBCFFF1})_{16}$.

Dans le cas du TDES, Dans les deux implémentations AES, nous avons pris comme clé : $(\text{F33C77F99BBCFFF1})_{16}$.

Dans le cas du AES, Dans les deux implémentations AES, nous avons pris comme clé : $(\text{F33C77F99BBCFFF1F33C77F99BBCFFF1})_{16}$.

Notre travail est dédié aux applications à haut débit qui nécessitent le chiffrement de grandes quantités d'information. L'objectif est de réduire le temps de chiffrement et de déchiffrement, c'est pourquoi nous sommes tournés vers LUT pour la programmation de bas niveau.

Cela nous donne un temps d'exécution satisfaisant en FPGA par rapport à MATLAB, ce que nous remarquant à partir de tableau 3.13 est fortement réduit.

Nous remarquons également que l'algorithme le plus rapide est AES, en raison de bloc de 128 bits par rapport à l'algorithme DES de bloc de 64 bits, et l'algorithme le plus lent est TDES, en raison de la taille de bloc (64bits), et il utilise également trois fois l'algorithme DES.

3.10. Conclusion

Nous avons introduit ce chapitre par une brève nécessaires pour le développement de notre cryptographie des différents outils système en présentant les différentes étapes d'implémentation sur un circuit programmable FPGA, et en décrit le langage de description matérielles utilisée puis l'environnement de travail. Par la suite, nous avons exposé nos résultats de simulation de chaque algorithme pour les 4 mode pour attester leurs bons fonctionnements, en les validant par MATLAB, nous avons implémenté notre algorithmes (AES, DES et TDES) sur un circuit FPGA, en précisant le temps d'exécution.

Conclusion générale

Conclusion générale

Le thème de notre projet étant l'étude et l'implémentation sur SoC-FPGA d'une méthode de cryptage symétrique. Une partie théorique, à travers une série des synthèses, pour établir une généralité sur la cryptographie, en particulier les algorithmes de cryptage symétrique (DES, 3DES et AES). Ensuite, il y a la partie pratique qui consiste en mesure de créer une IP pour le crypto-système DES, AES et 3DES dans les quatre modes (ECB, CBC, CFB et OFB) et de l'implémenter sur le circuit FPGA.

Notre conception IP a été implémentée par programmation sur le niveau le plus bas des circuits FPGA, c'est-à-dire sur (look Up Table) ou LUT, ce qui nous a permis d'étude les performances de chiffrement et de déchiffrement de chaque algorithme et en réduisant les délais de routage.

Une implémentation matérielle FPGA est physiquement plus sûr, plus rapide (haute vitesse) et plus fiable. Par conséquent, c'est une alternative prometteuse pour implémenter des chiffrements et déchiffrement par bloc. Par l'implémentation sur les algorithmes de chiffrement par bloc (DES, TDES et AES) on trouve que le premier algorithme (DES) consomme moins de ressources matérielles et la deuxième (AES) est le plus rapide possible que le premier algorithme sur FPGA.

Le chiffrement DES se caractérise par sa facilité d'implémentation. Il n'y a en outre pas de différence entre le chiffrement et le déchiffrement (juste l'ordre dans lequel les bits de clé présentés à la fonction f). C'est pourquoi on consomme moins de ressources matérielles mais la taille de la clé le rend lent. Par contre l'algorithme AES qui contient des fonctions complexe de chiffrement et de déchiffrement ce qui le fait consommer plus de ressources matérielles mais la taille de la clé est plus rapide qu'en DES.

Enfin, nous recommandons comme travaux future :

- Utiliser des conceptions plus élaborées et plus performantes que les versions utilisées.
- Introduire l'utilisation des techniques pipeline ce qui augmentera la productivité.

Bibliographie

Bibliographie

[1] : S. AZOUG, « développement et implémentation des technique de cryptage des signaux image et vidéo », Thèse de Doctorat, Université FARHAT-ABASS-SETIF1, Faculté de Science et Technologie ,26 Mai 2016.

[2] : T. BEKKOUCH, « Développement et implémentation des technique de cryptage des données basées sur les transformées discrètes » Thèse de Doctorat, Université FERHAT ABBAS SETIF-1, Faculté de Technologie.

[3] : R. RIMANI « Sécurisation des images par une combinaison des techniques de chiffrement et de recalage d'image », Thèse de Doctorat, Université des Science et de la Technologie d'Oran MOHAMED BOUDIAF, 2021

[4] : N. Hassan, « Conception et simulation des générateurs, crypto-systèmes et fonctions de hachage bases chaos performants », Thèse de doctorat en Electronique. UNIVERSITE DE NANTES, 2012. Français

[5] : F. OMARY, « Application des algorithme évolutionnistes à la cryptographie », Thèse de Doctorat d'état, Université MOHAMED V- AGDAL RABAT (MAROC) ,26. juillet.2006

[6] : R. BENIANI, « Sécurité des images Numériques compressées JPEG », Thèse de Doctorat 3ème Cycle, Université DjilalLiebes– Sidi Bel Abbes – Faculté des Sciences Exactes.

[7] : K. DICHOU, « contribution à l'étude des cartes à puce avancées », Thèse de Doctorat-LMD, Université M'HAMED BOGARA-BOUMERDES ,2016

[8] : J. PHILIPPE Aumasson « SERIOUS CRYPTOGRAPHY A Practical Introduction to Modern Encryption », 2018

[9] : A. BELOUCIF, « Contribution à l'étude des mécanismes cryptographes », Thèse de Doctorat en informatique, Université de BATNA-2, 22. Septembre.2016

[10] : D. BARSKY & G. DARTOIS, « Cryptographie Paris 13 », version 2010/2011.

[11] : C. PAAR, J. PELZL, Understanding Cryptography, Springer-Verlag Belin Heilberg, ISBN 978-3- 642-04101-3

[12] : J. DAEMEN, V. RIJMEN, The Design of Rijndael, AES: The Advanced Encryption Standard, ISBN 3540425802, 2001.

[13] : Federal information Processing Standards Publication 197, « Advanced Encryption Standards (AES) », 26 November 2001.

[14] : I. ABD-ELGHAFAR, A. ROHIEM, A. DIAA, F. MOGAMMED, Generation of AES Key Dependent S-boxes using RC4 Algorithm, International Conference on AEROSPACE SCIENCE &AVIATION TECHNOLOGY, Military Technical College, KobryElkobbah, Cairo, Egypt,26 may 2009.

[15] : Galois Field in Cryptography ». Christoforus Juan Benvenuto. May 31, 2012.

[16] : Article. https://www.brainkart.com/article/AES-Key-Expansion_8410/].

[17] : S. PILLEMENT, « Conception d'architectures reconfigurables dynamiquement : Du silicium au système », Micro et nanotechnologies/Microélectronique, Université Rennes 1, 22 Octobre 2010.

[18] : I. KUON, R. TESSIER, J. ROSE, « FPGA Architecture: Survey and Challenges », Foundations and Trends in Electronic Design Automation, Vol. 2, No, 2 (2007) 135-253.

[19] : T. NACHEF, « Implémentation d'une instrumentation sur un FPGA », Mémoire de magister en électronique, Université MOULOUD MAMMERI DE TIZI-OUZOU, 28 novembre 2011.

[20] : D. SAPTONO, « Conception d'un outil de prototypage rapide sur le FPGA pour des applications de traitement d'images », Thèse de Doctorat, Université de BOURGOGNE, Faculté de Science et Technologie E2S, 4 Novembre 2011.

[21] : A.TANEEM, D.PAUL, KUNDAREWICH, H.JASON, ANDERSON, « Packing Techniques for Virtex-5 FPGAs», Reconfig. Techno. Syst. 2, 3, Article 18 (Septembre 2009), 24 pages.

[22] : XILINX : https://www.xilinx.com.