REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE M'HAMED BOUGARA - BOUMERDES
FACULTE DES SCIENCES
Département : Physique

Mémoire

Présenté en vue de l'obtention du diplôme de

Master

En Infotronique

Thème

Conception d'une architecture matérielle pour l'AES et implémentation sur circuit FPGA

Présenté par :

- **TAFIROULT Mohamed Hussein**
- **TERMECHE Hayet**

Promoteurs :

- **Dr M. ANANE Mohamed (ESI)**
- **Mme H. BOUMERIDJA (UMBB)**
- **Mr O. AZZOUZI (CDTA)**

Promotion 2015/2016

Dédicace

Je dédie ce modeste travail:

Spécialement à mes chers parents, mon précieux père,
qui m'a vraiment aidé par tous les moyens en sa possession, et
ma chère mère, qui a était toujours présente pour moi.
Quoique je puisse dire,
je ne peux exprimer ma gratitude et mes sentiments d'amour et
de respect envers eux.
À mon frère et à mes sœurs, à toute ma famille,
à mes amis,
spécialement les membres
du groupe « craziness ».
À tous ceux qui
m'aiment et
à tous ceux
que
j'aime.

TAFIROULT M^{ed} Hussein

Dédicace

Je dédie ce modeste travail à tous ceux qui me sont chers:

***A** l'être le plus cher pour moi dans cette vie, ma mère et,*

À l'être que je respecte le plus dans ce monde, mon cher père,

pour leur sacrifice et encouragement afin de finir ce travail. Je leur souhaite une vie plein de joie, de santé et de bonheur, que ALLAH me les garde pour le restant de ma vie.

***A** mes oncles et leurs femmes, mes tantes et leurs maris et toute ma grande Famille*

***A** mes chères amies*

***A** tous mes ami(e)s du groupe SID*

TERMECHE Hayet

Remerciements

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

Nos sincères remerciements à notre encadreur, M. ANANE Mohamed pour avoir accepté de nous accueillir. Par sa compétence, ses conseils, ses critiques et son entière disponibilité, il a joué un rôle déterminant dans le progrès de notre travail. 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.

Nous remercions respectueusement M. AZZOUZI Oussama pour son temps précieux qu'il nous a accordé, son aide et sa patience.

Nous tenons à exprimer notre profonde gratitude et nos remerciements les plus sincères à Mme H. BOUMERIDJA, sous la direction de laquelle nous avons eu le plaisir de travailler. Ses conseils, ses critiques et sa rigueur scientifique nous ont permis de mener ce travail à son terme.

Un vif merci à nos grandes familles TAFIROULT et TERMECHE.

Nous tenons à remercier tous ceux qui, de près ou de loin, ont d'une manière ou d'une autre aidé à réaliser ce travail, nous citons en particulier Mme. ANANE, M. BELLEMOU, M. ISAAD.

Nous remercions sincèrement les enseignants du département INFOTRONIQUE, ainsi que nos chers amis et collègues.

TERMECHE et TAFIROULT

Résumé

Résumé

Le travail présenté dans ce mémoire a pour objectifs : l'étude, la conception et l'implémentation d'un IP pour le crypto-système AES. Cet IP d'AES sera modélisé par VHDL et implémenté sur circuit FPGA de la famille Virtex-7 de XILINX en le programmant au plus bas niveau et en utilisant certaines techniques d'optimisation.

Nous avons cherché à atteindre un bon compromis entre la vitesse de fonctionnement et la surface occupée par notre IP, pour cela l'architecture parallèle-série du module AES a été choisie.

L'environnement de travail utilisé est XILINX ISE 13.2, où les programmes ont été simulés par l'intermédiaire de l'outil de simulation VHDL ISIM et les résultats d'implémentation de notre IP d'AES ont été comparés avec un exemple publié par le NIST et ont été satisfaisants.

Mots clés : conception, implémentation, crypto-système AES, VHDL, FPGA.

Summary

The work presented in this memory aims to study the conception and implementation of an IP for the AES crypto-system. This AES IP will be programmed by VHDL and implemented on FPGAs circuit of Virtex-7 family of XILINX by programming it in the lowest level and using some optimization techniques.

We aimed to achieve a good compromise between the execution speed of our IP and its occupied area, therefore the parallel architecture of the AES unit has been selected.

The environment tool used is XILINX ISE 13.2, where programs were simulated through the VHDL simulation tool ISIM and the implementation results of our IP have been compared with an example published by NIST.

Keywords: conception, implementation, AES crypto-system, VHDL, FPGA.

Table des matières

Liste des figures

Liste des tableaux

Acronymes et abréviations

AES : Advanced Encryption Standard

CLB : Configurable Logic Bloc

DES : Data Encryption Standard

FPGA : Field programmable Gate Array

IP : Intellectual Property

ISE : Integrated Synthesis Environment

LuT : Look-up Table

NIST : National Institute of Standards and Technology

RAM : Random Access Memory

ROM : Read-Only Memory

RSA : Rivest-Shamir-Adleman

VHDL : VHSIC Hardware Description Language

Introduction générale

Introduction Générale

Les progrès technologiques récents dans les réseaux informatiques, tel que l'Internet, ont conduit à une augmentation de l'utilisation des ordinateurs et des systèmes de communication pour partager les données, où dans de nombreuses applications une connexion sécurisée est nécessaire.

La cryptographie joue un rôle important dans la protection des données échangées contre les attaques et diminue le risque de vol de l'information par ses deux types de cryptographies symétrique et asymétrique. La cryptographie symétrique utilise une même clé de petite taille (128 à 256 bits) pour chiffrer et déchiffrer l'information ce qui la rend rapide et utilise des opérations simples tels que : des décalages, des permutations et des additions. Par contre la cryptographie asymétrique utilise une paire de clés de (1024 à 2048) bits : une clé publique pour le chiffrage et une clé privée pour le déchiffrage, ce qui résulte des calculs très longs.

L'AES (Advanced Encryption Standard) est le protocole de chiffrement symétrique le plus utilisé actuellement. Il consiste en l'exécution de 10, 12 ou 14 rounds séquentiellement en utilisant des clés de 128, 192 ou 256 bits sur des blocs de données de 128 bits. Chaque round est constitué de 4 opérations *SubBytes*, *ShiftRows*, *MixColumn* et *AddRoundKey*.

Accélérer l'AES revient à accélérer ces opérations en les implémentant sur matériel tel que l'utilisation des circuits programmables comme les FPGAs vu les avantages qu'ils offrent : reprogrammation, flexibilité et ressources matérielles. Encore, l'AES peut être implémenté sur matériel en utilisant les architectures Série/Série, Parallèle/Série où Parallèle/Pipeline offrant des niveaux de performances : faible, moyen et élevé.

Dans ce projet dont l'intitulé est: " Conception d'une architecture matérielle pour l'AES et implémentation sur circuit FPGA ", il est question d'étudier les opérations mathématiques utilisées par l'AES pour en concevoir une architecture performante réalisant le chiffrement/déchiffrement et qui sera implantable sur circuits FPGA de Xilinx.
Le but dans ce projet de fin d'études est la réalisation d'un IP (Intellectual property) en VHDL pour le crypto système AES. Ce dernier sera capable de supporter les applications embarquées avec l'accélération de leurs temps d'exécution en diminuant leurs temps de chiffrement et de

déchiffrement tout en réduisant l'écart entre ces temps en maintenant un bon compromis entre la surface occupée par notre AES IP sur circuit FPGA et son temps d'exécution.

Ce rapport est structuré en quatre chapitres :

Le premier chapitre traite les concepts et les principes de base de la cryptographie où ses deux types : symétrique et asymétrique ont été exposés.

Le deuxième chapitre donne une description détaillée de l'algorithme AES ainsi que de ces transformations.

Le troisième chapitre décrit la conception des architectures proposées pour les processus de chiffrement, déchiffrement et la génération des sous clés en exposant tous les blocs de chaque architecture.

Le dernier chapitre présente les résultats de simulation et d'implémentation de notre IP sur un circuit FPGA de Xilinx en utilisant l'environnement de conception ISE.

Enfin, notre rapport se termine par une conclusion générale et quelques perspectives.

Chapitre I

Généralités sur la cryptographie

I.1. Introduction

Les communications ont toujours constitué un aspect important dans l'acquisition de nouvelles connaissances et l'essor de l'humanité. Le besoin d'être en mesure d'envoyer un message de façon sécuritaire est probablement aussi ancien que les communications elles-mêmes. Ce qui a donné naissance à la science que nous appelons cryptologie qui a pour objectifs d'assurer l'intégrité, l'authenticité et la confidentialité.

La cryptologie évolue de plus en plus en parallèle avec les tentatives de casser l'un de ses objectifs par un tiers malveillant, afin de subir des modifications du message chiffré transmis ou d'en prendre connaissance.

Actuellement, il y a de plus en plus d'informations qui doivent rester secrètes ou confidentielles telles que les informations échangées par les banques où un mot de passe ne doit pas être divulgué et personne ne doit pouvoir le déduire, c'est pourquoi ce genre d'information est crypté [FIG 00].

Dans ce chapitre, nous allons présenter des généralités sur la cryptographie et ses deux types symétrique et asymétrique qui sont nécessaires pour comprendre le fonctionnement d'un crypto système.

I.2. Définition de la cryptologie

Depuis toujours, l'être humain a essayé de mettre en place des protocoles servant à un échange sûr de l'information tels que les cachets d'enveloppes et les signatures manuscrites. De nos jours ces mêmes procédés trouvent leurs analogues dans le monde informatique et ceci grâce à la cryptologie.

La cryptologie est la science qui traite la façon de modifier ou de masquer une information afin de la rendre incompréhensible aux yeux des autres ; à l'exception de celui à qui est destinée. Celui-ci lui rendra son aspect initial grâce au secret qu'il détient qui est la clé.

La cryptologie est basée sur les mathématiques et l'informatique et comporte deux branches : la cryptographie et la cryptanalyse [MES 06].

I.2.1. La cryptographie

La cryptographie concerne la transformation d'un message (texte, image, chiffres) intelligible vers un message codé, incompréhensible à tous sauf pour les détenteurs de la clé de chiffrement.

C'est une discipline qui étudie les méthodes pour assurer le secret et l'authenticité des messages. Le terme « cryptographie » vient du grec « kriptos » (caché) et « graphein » (écrite).

L'art de définir des codes est la cryptographie (un spécialiste de cryptographie est un cryptographe). Elle définit les moyens et les méthodes utilisés pour assurer la confidentialité (et d'autres fonctions de sécurité) des informations que l'on va acquérir, stocker, traiter, diffuser et échanger.

I.2.2 La cryptanalyse

Depuis l'existence des codes secrets, on a cherché à les casser et à comprendre les messages chiffrés bien que l'on n'en soit pas le destinataire légitime, autrement dit les décrypter.

Décrypter ou casser le code c'est parvenir au texte en clair sans posséder au départ les règles ou documents nécessaires au chiffrement.

L'art de casser des codes est la cryptanalyse (un spécialiste de cryptanalyse est un cryptanalyste, cryptologue ou casseur de codes).

I.3. Objectifs de la cryptographie

Les principaux services offerts par la cryptographie moderne sont [CAN 06] :

- **Confidentialité** : assurer que les données concernées ne pourront être dévoilées que par les personnes autorisées.
- **Intégrité** : assurer que les données ne seront pas altérées pendant leur transmission ou leur stockage.
- **Authentification/Identification** : prouver l'origine d'une donnée ou s'assurer de l'identité d'une personne.
- **Non-répudiation** : garantir que les actions ne seront pas reniées.
-

I.4. Historique de la cryptographie

L'origine de la cryptographie remonte sans doute aux origines des hommes, dès que ceux-ci apprirent à communiquer. Alors, ils durent trouver des moyens d'assurer la confidentialité d'une partie de leurs communications. On rapporte son utilisation en Egypte il y a 4000 ans. Cependant, la première attestation de l'utilisation délibérée d'un moyen de chiffrement des messages vint de la Grèce vers le VI^{ème} siècle avant J.C, et se nomme le *scylate*, qui était un bâton sur lequel l'expéditeur enroulait une bandelette autour et écrivait longitudinalement le message sur le bâton, puis déroulait la bandelette et l'expédiait au destinataire. Sans la connaissance du diamètre du bâton qui jouait le rôle de clé, il était impossible de déchiffrer le

message. Plus tard, les romains adoptèrent un chiffrement qui consistait en une substitution mono alphabétique simple en décalant de trois lettres de l'alphabet. Cette technique était connue sous le nom de chiffre de *Jules César*.

Puis, pendant des siècles, on assista à la mise au point de plusieurs techniques de chiffrement mais qui étaient pour la plupart limitées aux besoins de l'armée et de la diplomatie ; on peut citer parmi ces techniques :

- En 1918 le système ADFGVX a été mis dans le service par les allemands à la fin de la première guerre mondiale et Arthur Scherbius fait breveter sa machine à chiffrer *Enigma*.
- En 1929 : Lester S. Hill invente le chiffre de Hill. C'est un chiffre polygraphique où l'on utilise des matrices et des vecteurs.
- En 1960 : le code ASCII est adopté comme standard. Il permet le codage de caractères sur 8 bits, soit 256 caractères possibles.

Dans les années 70, le développement de l'informatique et l'émergence des réseaux de communications modifient la situation. La sécurité des nouveaux moyens de communications doit être assurée. C'est pourquoi, en 1975, le Bureau Américain des Standards propose de normaliser un système de chiffrement : le DES (Data Encryption Standard) qui est un système de chiffrement par blocs de 64 bits basée sur l'utilisation d'une clé secrète identique pour le chiffrement et le déchiffrement dont la taille est de 56 bits.

Plusieurs organismes ont analysé le DES et l'ont trouvé mathématiquement sain mais la longueur de sa clé a été jugée trop faible pour des applications de haute sécurité. Ainsi le DES a été remplacé par le nouvel algorithme AES (*Advanced Encryption Standard*) qui a des clés de longueur plus importante (128, 192 et 256 bits) ainsi que des blocs de taille plus grande (128 bits contre 64 pour DES).

Le standard de chiffrement AES fut adopté en 2000 par le NIST en remplacement du DES.

Ce chiffrement est constitué de substitutions, de décalages, de « ou exclusif » et de multiplications dans un corps fini de polynômes fixes ; ces opérations sont élémentaires, simples et rapides à calculer.

Il permet de crypter des blocs de 128, 192 ou 256 bits en utilisant des clés symétriques de 128, 192 ou 256 bits. Le choix de la taille de la clé et de la taille des blocs sont indépendants, il y a donc au total 9 combinaisons possibles. Ceci laisse une plus grande flexibilité à l'utilisateur de l'AES en fonction du niveau de sécurité et de la vitesse de calcul désirés.

Parallèlement en 1976, *W. Diffie* et *M. E. Hellman* publient leur célèbre papier « New *Directions in Cryptography* ». Ils y décrivent les fondements de la cryptographie asymétrique moderne permettant de résoudre en partie les problèmes d'échange des clés secrètes. Ce type de crypto système utilise une clé secrète pour le déchiffrement, alors que c'est une clé publique qui est employée pour chiffrer le message.

La première application pratique de la cryptographie asymétrique est le système **RSA** proposée en 1978 par R.L. Rivest, A. Shamir et L. Adleman. Ce système est d'ailleurs le crypto système asymétrique le plus répandu à l'heure actuelle.

Aujourd'hui, la cryptologie ne se restreint plus au simple chiffrement des messages pour en garantir la confidentialité : elle s'attache aussi à permettre l'authentification des diverses entités qui communiquent à distance dans un monde virtuel, à autoriser la signature de documents numériques immatériels, à garantir l'intégrité des données dans des réseaux ouverts, à protéger l'anonymat des données personnelles sensibles, à contribuer à la protection des contenus, etc. Elle emprunte ses méthodes à des domaines scientifiques divers, mathématiques et informatiques en premier lieu, mais également physiques, notamment quantiques.

Elle n'est plus réservée aux diplomates et militaires : aujourd'hui, des centaines de millions d'individus, à travers le monde, ont en permanence sur eux un ou plusieurs processeurs cryptographiques, pour leur téléphone mobile ou leur carte bancaire en particulier [RAM 09]. Dans la cryptographie moderne toute la sécurité est basée sur la clé (ou les clés), et non dans les détails des algorithmes. Cela signifie qu'un algorithme peut être publié et analysé, mais la clé doit être protégée.

I.5. Définition de la clé

Une ***clé*** est un paramètre utilisé en entrée d'une opération cryptographique (chiffrement, déchiffrement, signature numérique, vérification de signature). Elle doit être variable et maintenue en secret (sauf dans certains algorithmes où une part de la clé reste exposée).

Elle peut se présenter sous plusieurs formes : mots ou phrases, procédure pour préparer une machine de chiffrement (connexions, câblage, etc.), données codées sous une forme binaire (cryptologie moderne). La protection apportée par un algorithme de chiffrement est liée à la longueur de la clé, qui peut s'exprimer en bits. Les clés les plus grandes resteront crypto graphiquement sûres pour une plus longue période. Si ce que l'on chiffre doit rester caché pendant de nombreuses années, il faut utiliser une clé très grande.

I.6. Différents types de cryptographie

Les techniques cryptographiques se découpent en deux grandes parties :

- La cryptographie à clés secrètes ou cryptographie symétrique.
- La cryptographie à clés publiques ou cryptographie asymétrique.

Ils ont tous deux leurs avantages et leurs inconvénients. La différence qui existe entre ces deux types se situe au niveau de la clé.

I.6.1 Cryptographie symétrique

La cryptographie symétrique est basée sur l'utilisation d'une clé secrète identique pour le chiffrement et le déchiffrement. Le schéma de principe de la cryptographie à clé secrète est représenté sur la figure 1.1.

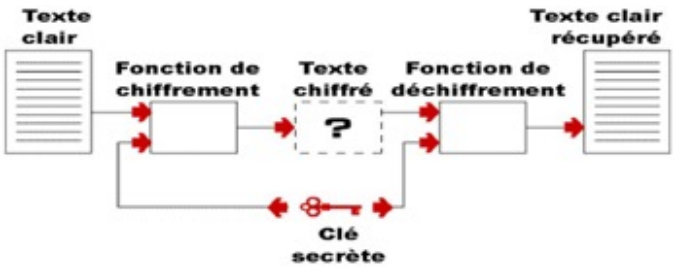


Figure 1.1. Schéma de principe de la cryptographie symétrique [MES 06].

La cryptographie symétrique est très utilisée et se caractérise par une grande rapidité et des opérations simples : décalage de certains bits, XOR bit à bit, permutations de certains bits, etc., et par des implémentations aussi bien software que hardware ce qui accélère nettement les débits et autorise son utilisation massive.

De plus, le message (clair) est séparé en blocs successifs, chaque bloc étant chiffré individuellement. Cela permet de disposer de systèmes de chiffrement très performants et très simples.

Les deux parties communicantes doivent se mettre d'accord sur la clé secrète utilisée. Le problème est qu'on ne dispose pas d'un canal de communication sûr pour échanger les clés.

I.6.2. Cryptographie asymétrique

La cryptographie à clé publique s'attache à résoudre le problème de gestion des clés. Son principe fondamental est de donner à chaque utilisateur deux clés associées, l'une secrète et l'autre rendue publique.

Afin de chiffrer un message à l'intention d'un utilisateur, l'idée consiste à n'utiliser que sa clé publique alors que le déchiffrement doit nécessiter la connaissance de la clé secrète.

Ce concept naturel permet de communiquer de manière confidentielle sans avoir à partager la moindre information secrète initialement.

Le schéma de principe de la cryptographie à clé publique est représenté sur la Figure 1.2

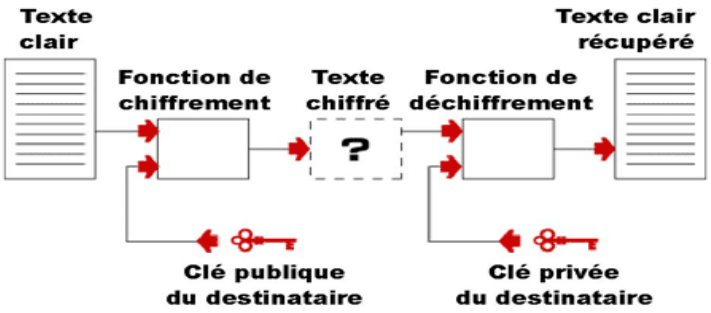


Figure 1.2. Schéma de principe de la cryptographie asymétrique [MES 06].

Du fait de l'utilisation de grands nombres premiers, les crypto systèmes asymétriques nécessitent une quantité de calcul importante, ce qui les rend très lents par rapport aux systèmes symétriques.

Le RSA, nommé d'après les noms de ses inventeurs (Rivest, Shamir et Adelman), est le premier algorithme asymétrique publié en 1977. La sécurité du RSA vient de la difficulté de factoriser des grands nombres premiers : s'il est facile de multiplier deux grands nombres premiers, il est très difficile de décomposer le très grand nombre obtenu en ses deux facteurs premiers quand on ne les connaît pas.

Les clés publique et privée sont une fonction d'un couple de grands nombres (1024 bits ou plus) premiers. Découvrir le texte en clair à partir de la clé publique et du texte crypté est conjecturé comme équivalent à factoriser le produit des deux grands premiers.

La clé publique contient le produit de deux nombres premiers très grands, et un autre nombre qui lui est propre. L'algorithme de chiffrement utilise ces nombres pour chiffrer le message par blocs. L'algorithme de déchiffrement nécessite quant à lui l'utilisation d'un nombre contenu uniquement dans la clé privée.

L'algorithme RSA utilise l'arithmétique de $\mathbb{Z}n$, où n est le produit de deux nombres premiers distincts p et q, choisis de façon aléatoire : $n = p \cdot q$

Ensuite il faut choisir une clé publique e de telle manière que e et $\Phi(n)$ soient premiers entre eux avec $\Phi(n) = (p - 1)(q - 1)$.

Finalement, à l'aide de l'algorithme d'Euclide étendu, on calcule la clé de décryptage d,

tel que : $e \cdot d \equiv [1] \bmod \Phi(n)$

D'une autre façon : $d = [e^{-1}]_{\Phi(n)}$

Il faut noter que d et e sont relativement premiers. Les nombres e et n forment la clé publique et les nombres d et n sont la clé privée. Les nombres p et q ne sont plus alors nécessaires et ne doivent être jamais révélés.

Pour crypter un message m, il est tout d'abord nécessaire de le découper en blocs numériques de taille plus petite que n (normalement la plus grande puissance de 2 plus petite que n), composés de bits binaires, c.-à-d., si p et q sont de 1024 bits chacun, donc le modulo n sera sur 2048 bits et chaque bloc m_i doit être légèrement au-dessous de 2048 bits. Le message crypté C sera aussi divisé en blocs C_i de la même taille que les m_i.

La formule du cryptage est la suivante : $C_i = [m^e_i]_n$.

Et le décryptage est obtenu simplement par : $m_i = [C^d_i]_n$.

Puisque : $C^d_i = (m^e_i)^d = m^{ed}_i = m_i^{k(p-1)(q-1)+1} = m_i \times 1 = m_i$.

Les clés RSA sont habituellement de longueur comprise entre 1024 et 2048 bits [BAB 12].

I.7. Comparaison entre les chiffrements symétrique et asymétrique

Les chiffrements symétriques et asymétriques présentent chacun des avantages et des inconvénients.

La principale difficulté des algorithmes de chiffrement symétrique ou à clés secrètes réside dans la sécurité de l'échange des clés. Un autre inconvénient est que tout couple d'utilisateurs doit au préalable s'entendre sur une clé commune. La gestion des clés devient vite problématique. Toutefois, les systèmes symétriques sont très efficaces. Ils demeurent sûrs, rapides et peuvent chiffrer et déchiffrer une grande quantité de données en des temps records.
Les crypto systèmes asymétriques actuels sont beaucoup plus lents que leurs homologues symétriques, et nécessitent de plus longues clés. Leur avantage principal réside dans la simplicité de la gestion des clés : le secret n'est pas partagé, les clés publiques peuvent être publiées dans l'annuaire, et le nombre de clés est bien inférieur lorsque plusieurs personnes veulent communiquer entre elles de façon confidentielle.

Le tableau 1 récapitule les avantages et inconvénients des chiffrements symétrique et asymétrique.

	Symétrique	asymétrique
Avantages	- Rapidité, jusqu'à 100 fois plus rapide. - Facilité d'implémentation sur Hardware. - Taille de clés 128 bits (implique 16 caractères mémorisable).	- Distribution des clés facilitée : pas d'authentification. - Permet de signer des messages facilement. - Nombre de clés à distribuer est réduit par rapport aux clés symétriques.
Inconvénients	- Nombre de clés à gérer. - Distribution des clés (authentification, confidentialité). - Certaines propriétés sont difficiles à réaliser	- Taille des clés. - Vitesse de chiffrement.

Tableau 1.1. Avantages et Inconvénients des chiffrements symétrique et Asymétrique [FOU 03].

I.8. Conclusion

Dans ce chapitre, nous avons présenté une introduction à la cryptographie où les deux types de cryptographie symétrique et asymétrique ont été exposés.

La cryptographie symétrique est rapide car elle utilise des clés de petites tailles et facile à implémenter sur matériel car elle utilise des opérations simples telles que des décalages, des XOR et des permutations et c'est à elle que nous nous intéressons.

Le prochain chapitre décrira le système cryptographique symétrique AES qui est la cible de notre implémentation matérielle sur circuit FPGA.

Chapitre II

Le protocole cryptographique AES

II.1. Introduction

Après avoir passé en revue quelques notions de cryptographie dans le premier chapitre, nous axons sur les algorithmes de chiffrement symétriques qui sont rapides et bien adaptés au chiffrement de grande quantité de données.

Le protocole de chiffrement AES offre la possibilité de chiffrer l'information avec des clés de petites tailles et par conséquent avec une faible complexité calculatoire.

II.2. L'Advanced Encryption Standard

L'AES est un algorithme de chiffrement symétrique basé sur le système Rijndael construit par Joan Daemen et Vincent Rijmen [DAE 01]. Il permet de chiffrer et déchiffrer l'information par blocs de 128 bits (16 octets binaires), à l'aide de clés de 128, 192 ou 256 bits.

L'AES s'exécute en plusieurs tours qui sont composés de plusieurs transformations [DAE 01]. Le nombre de tours Nr dépend de la taille des clés où 10 tours sont nécessaires pour des clés de 128 bits, 12 tours pour des clés de 192 bits et 14 tours pour des clés de 256 bits [AZZ 14].

II.2.1. Chiffrement et déchiffrement avec l'AES

Dans ce qui suit, nous détaillons l'AES-128, où les 128 bits de données sont répartis en 16 blocs de 8 bits (8 bits = 1 octet), eux-mêmes « dispatchés » dans un tableau 4×4. Même les 128 bits de la clé sont organisés sous forme matricielle [BAB 12]. Pour d'évidentes raisons de taille, les valeurs binaires seront notées sous forme hexadécimale.

La première étape de chiffrement consiste à combiner la matrice State (le bloc de texte clair) avec la clé. Cette opération s'appelle *AddRound* Key. À chaque tour, quatre transformations sont appliquées *SubBytes*, *ShiftRows*, *MixColumn* et *AddRoundKey* sauf pour le dernier tour, l'opération *MixColumns* n'est pas effectuée. Chaque tour utilise sa propre sous-clé qui est générée par l'opération Key Expansion à partir de la clé maitresse [AZZ 14].

Le déchiffrement est l'opération inverse du chiffrement et les transformations se réalisent dans le sens inverse.

La figure 2.1 montre les différentes opérations effectuées dans chaque round pendant le processus de chiffrement et de déchiffrement de l'AES.

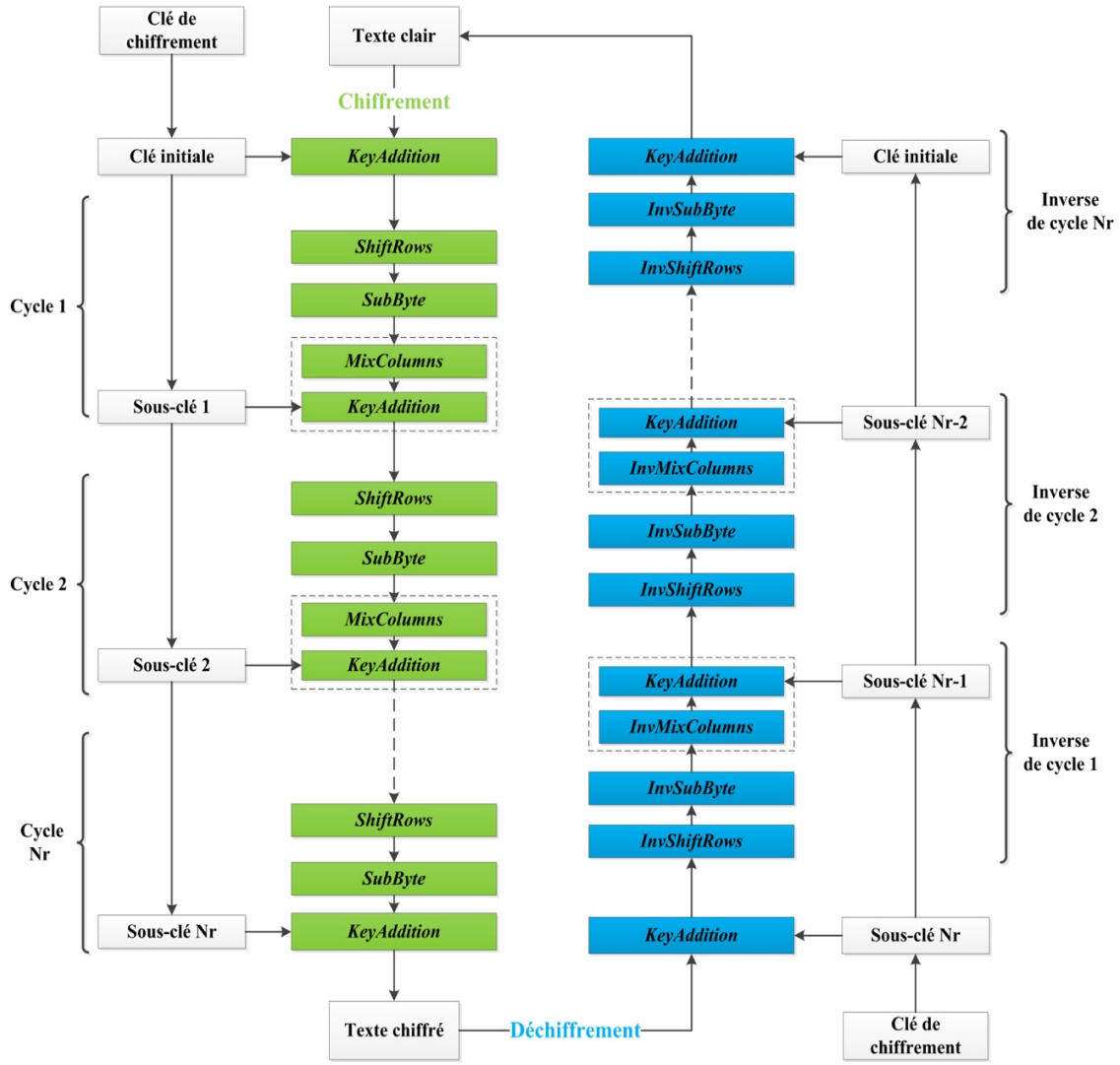


Figure 2.1. Transformations de l'AES [PAA 10].

- ***SubBytes***

L'étape *SubBytes* correspond à la seule transformation non-linéaire de l'algorithme. Dans cette étape [DAE 01], chaque élément de la matrice State est permuté selon une table de substitution inversible notée S-Box comme la montre la figure 2.2.

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

Figure 2.2. Table S-Box de l'AES [ABD 09].

La figure 2.3 illustre un exemple de transformation de l'élément S_{11} en l'élément S'_{11}.

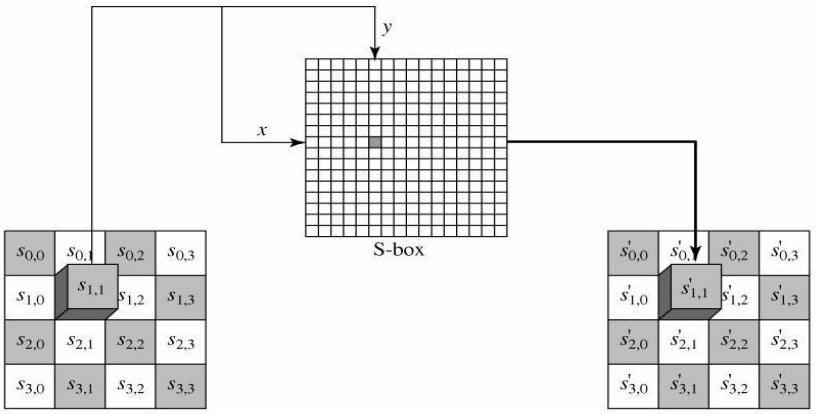


Figure 2.3. Transformation *SubBytes* [ANA 15].

La table S-Box dérive de la fonction inverse g : a → a^{-1} dans GF [2^8]. Cette fonction est connue pour ses bonnes propriétés de non-linéarité.

Afin d'éviter des attaques basées sur de simples propriétés algébriques, la table S-Box est construite en combinant sa fonction inverse avec une transformation affine inversible f. On a donc : S-Box[a] = f (g(a)), ∀ a ∈ GF [2^8] [ABD 09].

La fonction affine f : GF [2^8] → GF [2^8], est définie par :

$$b = f(a) \equiv \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

➢ ***Shiftrows***

Cette étape effectue pour chaque élément d'une ligne, un décalage cyclique vers la gauche. Le nombre de décalages varie selon la ligne. La première ligne est inchangée, la seconde subit un décalage de 1 vers la gauche, la troisième de 2 vers la gauche et la quatrième de 3 vers la gauche. Son opération inverse consiste à faire ce même décalage pour toute les lignes dans le sens inverse, comme le montre la figure 2.4 présentant le *ShiftRow* et son inverse.

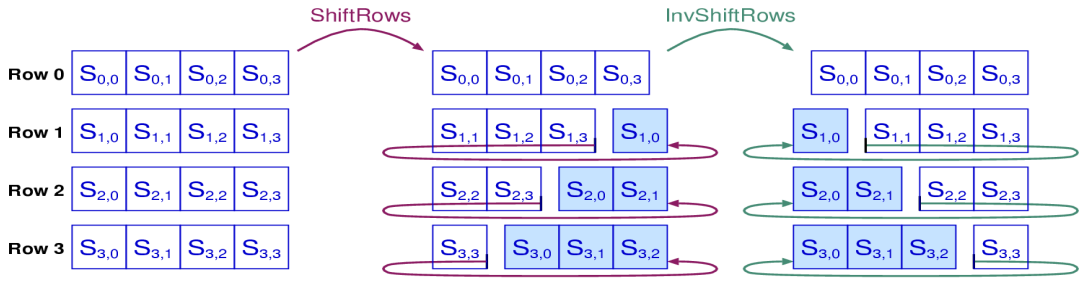


Figure 2.4. *ShiftRows* et Inverse *ShiftRows* [GUR 06].

➢ ***MixColumns***

Cette étape est une transformation linéaire qui mélange chaque colonne de la matrice state. Puisque chaque octet d'entrée influe sur quatre octets de sortie et chaque colonne de 4 octets est considérée comme un vecteur et est multipliée par une matrice fixe de (4×4), définie comme suit ;

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{bmatrix} \times \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

L'opération inverse consiste à multiplier chaque colonne de la matrice state par ma matrice prédéfinie ci-dessous ;

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

La multiplication et l'addition se font dans GF [2^8].

➢ ***AddRoundKey***

Les deux entrées de l'opération *AddRoundKey* sont la matrice state et la sous clé. Les deux entrées sont combinées par une opération XOR octet par octet. A noter que l'opération XOR est égale à l'addition dans le corps de Galois GF (2^8) [PAA 10].

II.3. *KeyExpansion*

C'est l'algorithme de dérivation des sous-clés à partir de la clé secrète [BAB 12]. Le nombre de sous-clés est égal au nombre de tours plus un. Les sous-clés sont calculées de manière récursive, afin de dériver une sous-clé k_i, la sous-clé k_{i-1} doit être connue [AZZ 14].

Le nombre de cycle du protocole cryptographique AES dépend de la taille de la clé utilisée et le processus de génération de clé diffère aussi selon la taille de cette clé.

II.3.1. Clés de 128 bits

La génération des clés est orienté mots, où 1 mot = 32 bits. Les 11 sous-clés sont stockées dans la matrice Key expansion w qui est constituée des mots (*W* [0], . . ., *W* [43]) comme le montre la figure 2.5 [PAA 10].

La première sous-clé est la clé d'origine AES qui est copiée dans les quatre premiers éléments de W. Les autres éléments de la matrice sont calculés comme suit :

Chaque mot *W[i]* dépend du *W [i-1]* et *W [i-4]*. Pour le mot dont sa position est un multiple de 4, une fonction plus complexe *g* est utilisée [AZZ 14]. g () est une fonction non linéaire avec une entrée et une sortie à quatre octets qui consiste à exécuter un décalage vers le haut d'un octet, substituer chaque octet en utilisant la table *Sbox* et ajouter un coefficient RC à elle. Le coefficient RC est seulement ajouté à l'octet le plus à gauche dans la fonction g ().

Ce coefficient varie d'un tour à un autre selon la règle suivante :

$$\begin{aligned} RC[1] &= x^0 = (00000001)_2, \\ RC[2] &= x^1 = (00000010)_2, \\ RC[3] &= x^2 = (00000100)_2, \\ &\vdots \\ RC[10] &= x^9 = (00110110)_2. \end{aligned}$$

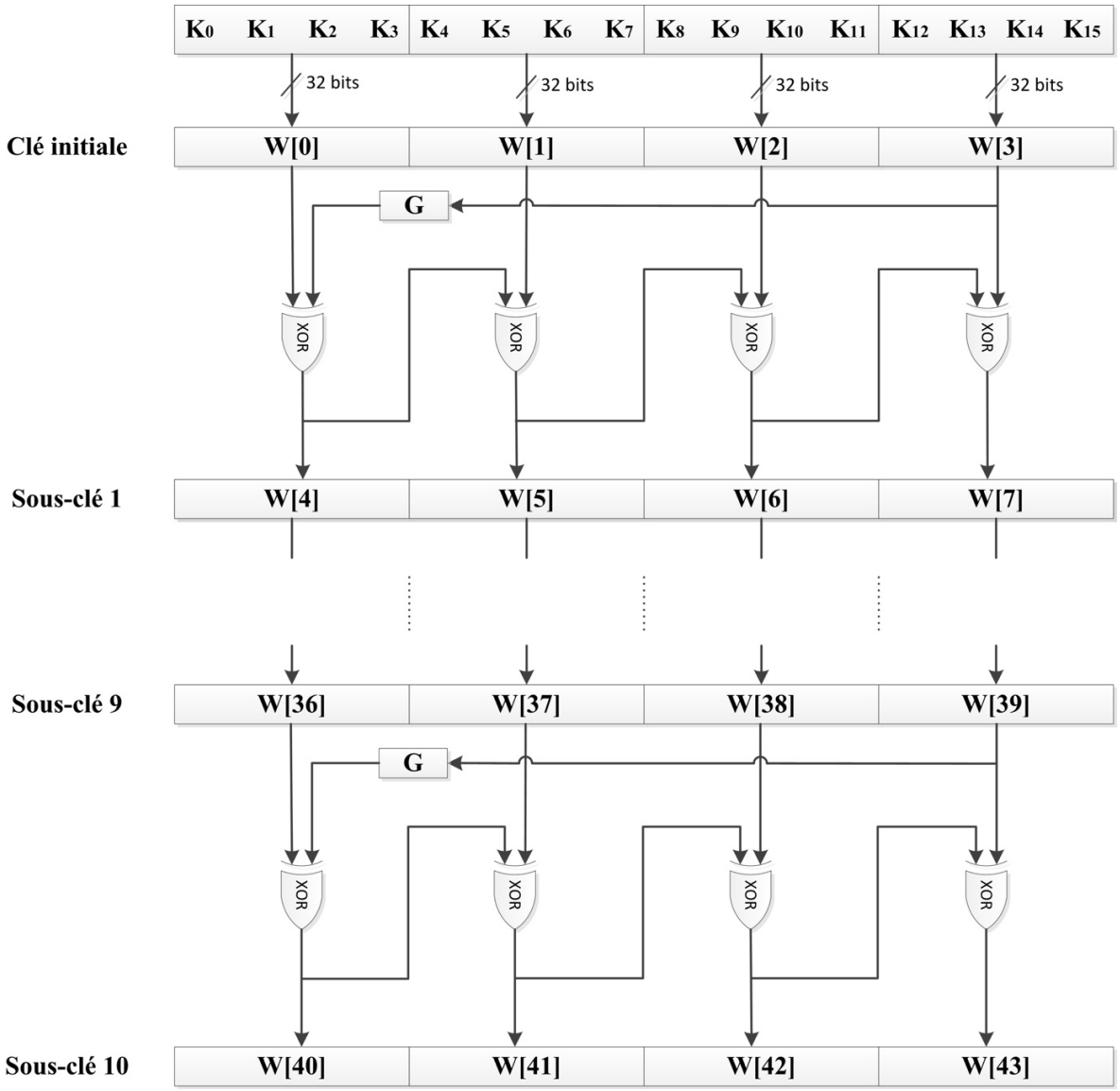


Figure 2.5. Génération des sous clés à partir d'une clé de 128 bits [PAA 10].

II.3.2. Clés de 192 bits

Dans le cas de clés de 192 bits, le processus de chiffrement/déchiffrement subit a 12 cycles et 13 sous-clés de 128 bits chacun. Le calcul des sous-clés est similaire à celui de la clé de 128 bits, il y'a huit itérations de key expansion. Chaque itération calcule six nouveaux mots du tableau W. La sous-clé pour le premier tour est formée par les éléments du tableau (W [0], W [1], W [2], W [3]), la deuxième sous-clé par les éléments (W [4], W [5], W [6], W [7]), et ainsi de suite. Huit coefficients RC [i] sont nécessaires au sein de la fonction g (), ils sont calculés comme dans le cas de 128 bits et rangés de RC [1], . . ., RC [8]. La figure 2.6 montre le processus de génération de clés dans le cas de 192 bits.

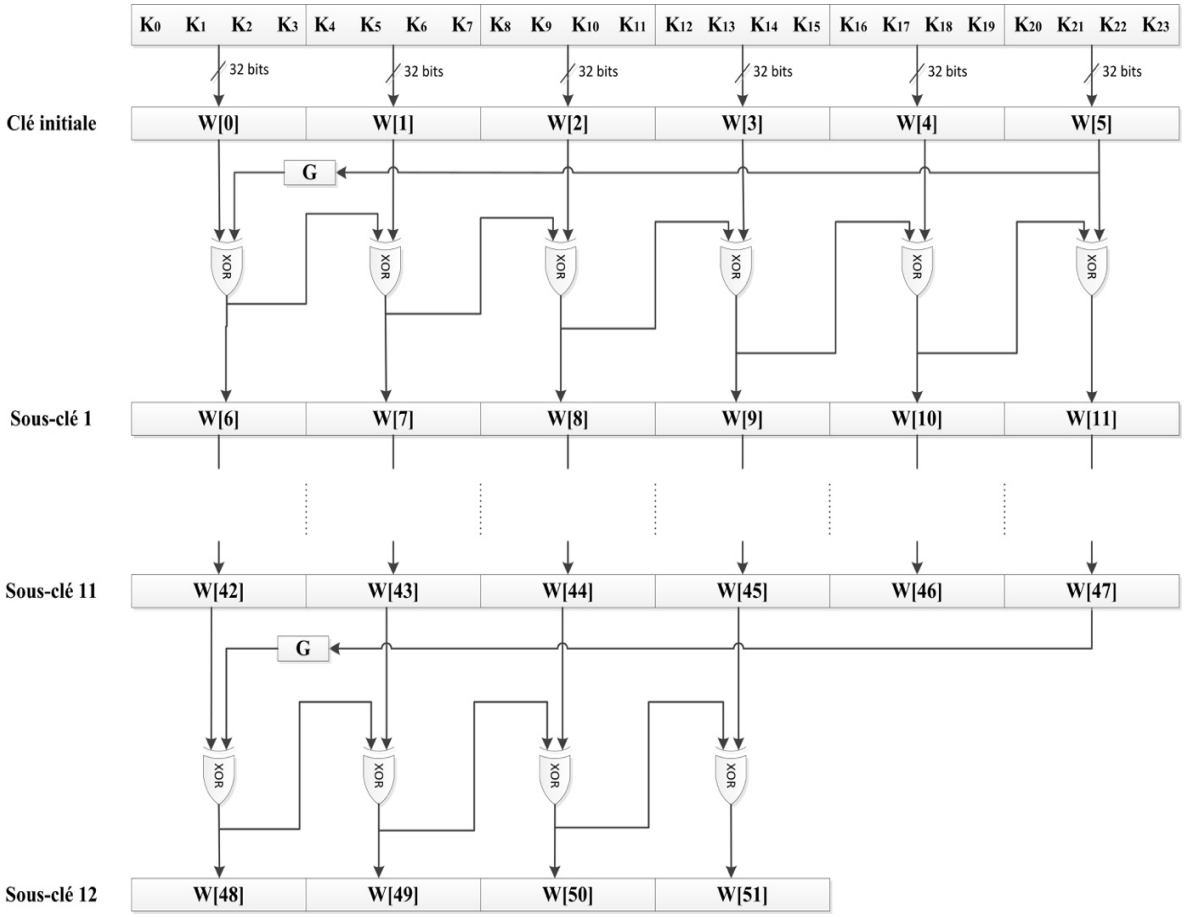


Figure 2.6. Génération de clés pour 192 bits [PAA 10].

II.3.3. Clés de 256 bits

AES avec une clé de 256 bits a besoin de 15 sous-clés. Les sous-clés sont stockées dans les 60 mots W[0], . . ., W[59]. Le calcul des éléments du tableau est similaire à celui de la clé de 128 bits et le Key Schedule comprend 7 itérations, où chacune calcule huit mots pour les sous-clés. La sous-clé pour le premier tour est formée par les éléments du tableau (W[0], W[1], W[2], W[3]), la deuxième sous-clé par les éléments (W[4], W[5], W[6], W[7]), et ainsi de suite. Il y a sept coefficients RC[1] . . . RC[7], nécessaires dans la fonction g(), qui sont calculés comme dans le cas de 128 bits. Le Key Schedule a une fonction h() avec des entrées /sortie de 4 octets. La fonction applique la S-Box à tous les quatre octets d'entrée. La figure 2.7 montre le processus de génération de clé dans le cas de 256 bits [PAA 10].

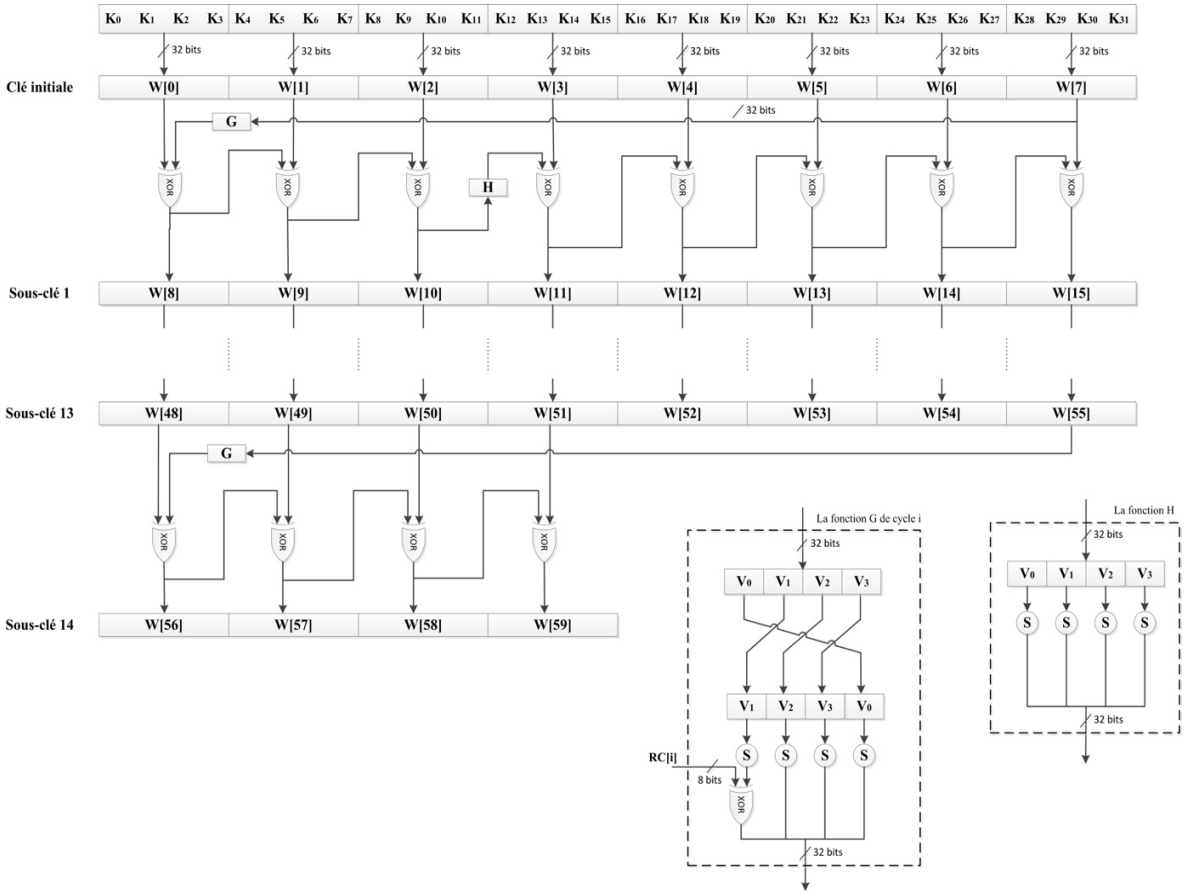


Figure 2.7. Processus de génération de clés pour 256 bits [PAA 10].

II.4. Les modes d'opération

En cryptographie, un mode d'opération est la manière de traiter les blocs de texte clairs et chiffrés au sein d'un algorithme de chiffrement par bloc. L'algorithme est combiné à une série d'opérations simples en vue d'améliorer la sécurité sans pour autant pénaliser l'efficacité de l'algorithme. Cette combinaison est appelée un mode cryptographique.

Dans ce qui suit, on va citer les quatre modes les plus courants tels que :

II.4.1. Le mode ECB

Le carnet de codage électronique, en anglais « Electronic Code Book » (ECB), illustré sur la figure 2.8 : le message à chiffrer est subdivisé en plusieurs blocs qui sont chiffrés séparément les uns après les autres. Le gros défaut de cette méthode est que deux blocs avec le même contenu seront chiffrés de la même manière, on peut donc tirer des informations à partir du texte chiffré en cherchant les séquences identiques. On obtient dès lors un « dictionnaire de codes » avec les correspondances entre le clair et le chiffré d'où le terme codebook.

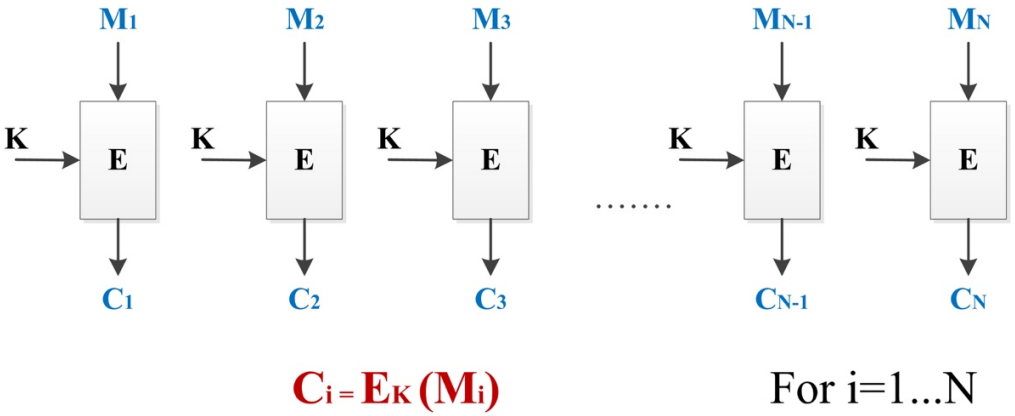


Figure 2.8. Le mode ECB

Parmi les problèmes de sécurité connus dans ce mode, on trouve que si on utilise deux fois le même texte en clair et la même clé de chiffrement, le résultat du chiffrement sera identique. Des régularités dans le texte en clair se reproduisent dans le texte chiffré. Le texte chiffré permet donc d'obtenir des informations sur le texte en clair, même si on n'arrive pas à le déchiffrer. Ce mode est, pour ces raisons, fortement déconseillé dans toute application cryptographique. Le seul avantage qu'il peut procurer est un accès rapide à une zone quelconque du texte chiffré et la possibilité de déchiffrer une partie seulement des données.

II.4.2. Le mode CBC

Le mode de chiffrement par chaînage de blocs est présenté sur la figure 2.9, de l'anglais « Cipher Block Chaining » (CBC). Il offre une solution à la plupart des problèmes du mode ECB. Le chaînage utilise une méthode de rétroaction comme le résultat du chiffrement du bloc précédent est réutilisé pour le chiffrement du bloc courant. Plus précisément, l'opérateur binaire XOR est appliqué entre le bloc actuel du texte en clair et le bloc précédent du texte chiffré et on applique ensuite l'algorithme de chiffrement au résultat de cette opération.

Pour le tout premier bloc : un bloc ayant un contenu aléatoire, appelé vecteur d'initialisation (VI), est généré et utilisé pour l'application de l'opération XOR. Son rôle est d'empêcher que si deux textes en clair débutent de la même façon, les textes chiffrés correspondants le sont également. Donc il doit être différent pour chaque message chiffré avec la même clé.

Ainsi, chaque bloc chiffré dépend non seulement du bloc de texte en clair correspondant, mais également de tous les blocs chiffrés qui le précèdent comme le montre la figure 2.9.

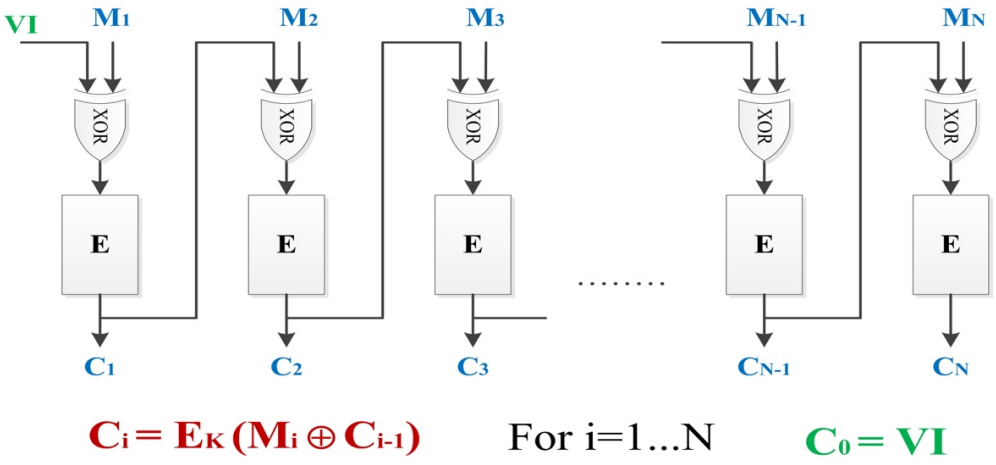


Figure 2.9. Le mode CBC

II.4.3. Le mode CFB

Le mode de chiffrement par rétroaction est présenté sur la figure 2.10, de l'anglais « Cipher FeedBack » (CFB). Le tout premier bloc de ce mode est un bloc ayant un vecteur d'initialisation (VI) qui sera chiffré avec l'algorithme de chiffrement utilisé et on le combine par un ou exclusif avec le texte en clair pour obtenir ainsi le texte chiffré qu'on peut alors transmettre. Le résultat de cette combinaison sera réutilisé pour le chiffrement du bloc suivant. Plus précisément, on le chiffre avec le même algorithme de chiffrement et on le combine par la suite avec la clé par un ou exclusif.

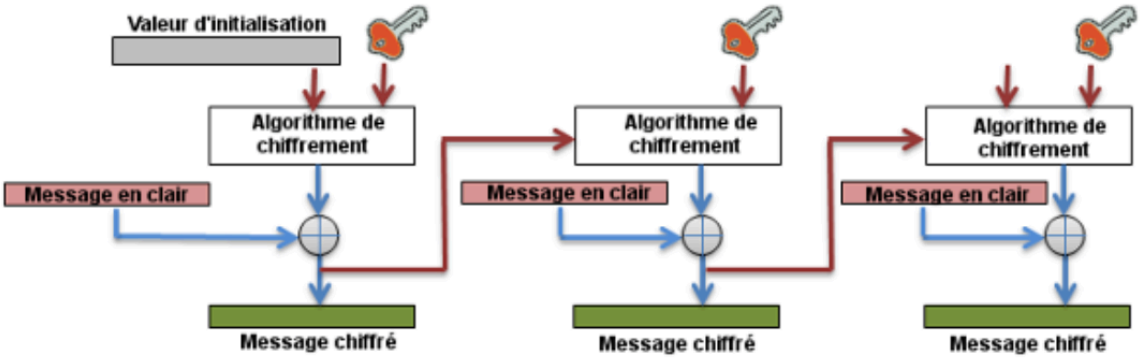


Figure 2.10. Le mode CFB

Le mode CFB offre une grande sécurité. Le seul problème est que le texte en clair est seulement soumis à un XOR. Si le texte en clair est connu, un texte en clair complètement diffèrent peut-être substitué en inversant les bits du texte chiffré au même endroit où on inverse les bits du texte en clair.

II.4.4. Le mode OFB

Le mode de rétroaction de sortie, en anglais "Output Feedback" (OFB), ressemble beaucoup au mode CFB, sauf que le résultat du chiffrement du vecteur d'initialisation du bloc précédent qui sera réutilisé pour le chiffrement du bloc courant [DAE 01].

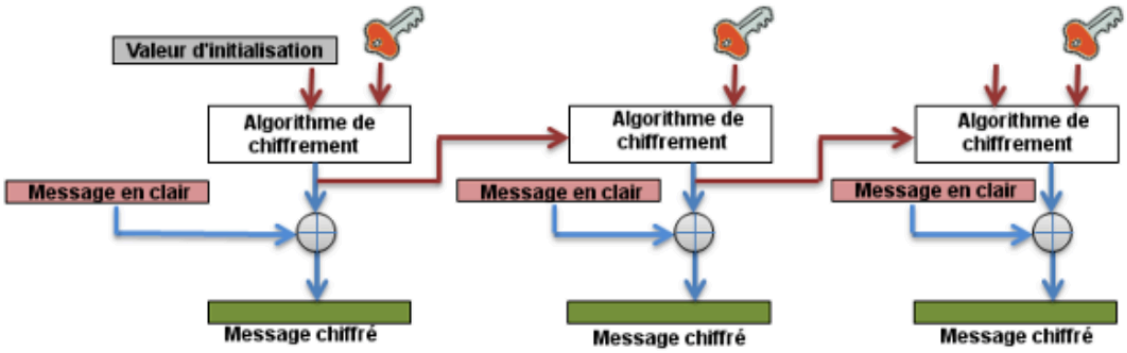


Figure 6.3. Le mode OFB

II.5. Conclusion

Dans ce chapitre, nous avons présenté le principe de l'AES. Les principales opérations utilisées dans ce protocole de chiffrement ont été exposées où nous avons remarqué que 75% de la sécurité de ce protocole repose sur l'opération SubBytes.

Dans le chapitre qui suit, il sera question d'étudier les circuits FPGA et voir les différentes technologies ainsi la méthodologie utilisée pour la conception et l'implémentation d'un tel système.

Chapitre III

Conception & contribution

III.1. Introduction

Après avoir vu en détail le protocole cryptographique AES, nous allons exposer la démarche pour la conception de notre crypto-système sur une plateforme purement matérielle modélisée en VHDL et implémentée sur circuit FPGA.

Dans l'objectif d'atteindre un bon compromis entre le temps d'exécution et la surface occupée, notre approche est basée sur la bonne utilisation des ressources disponibles sur des circuits programmables les plus récents d'FPGA de XILINX.

III.2. Architectures matérielles de l'AES

En général, il existe trois types d'architectures pour l'implémentation du protocole de chiffrement et déchiffrement AES :

- **L'architecture série-série,** où les quatre blocs de 32 bits constituant le message d'un round sont exécutés en série et les rounds en séquentiel.
- **L'architecture parallèle-série**, où les quatre blocs de 32 bits constituant le message d'un round sont exécutés en parallèle et les rounds en séquentiel.
- **L'architecture parallèle-pipeline,** où les quatre blocs de 32 bits constituant le message sont exécutés en parallèle et les rounds en pipeline [AZZ 14].

Concernant le choix de l'architecture adéquate à la conception de notre crypto-système AES, nous constatons que l'architecture série-série occupe moins de ressources par rapport aux autres architectures, mais en revanche un temps très important est perdu en exécutant les quatre blocs de 32 bits séquentiellement. De même, l'architecture parallèle-pipeline est dédiée aux applications fortes débit, mais elle nécessite beaucoup de ressources matérielles.

Ce raisonnement nous a conduit de choisir une architecture parallèle-série pour la conception de notre système. Cette architecture offre un bon compromis entre la rapidité d'exécution et les ressources utilisées. En effet, la matrice d'entrée « state » de 128 bits sera découpée en quatre blocs de 32 bits qui seront traités en parallèle. L'exécution d'un cycle a besoin de 4 blocs de 32 bits pour la transformation *MixColumns* et 16 *Sboxs* implémentés en parallèle, travaillant avec des données indépendantes, ce qui augmente le débit de cryptage/décryptage.

La figure 3.1 illustre l'architecture parallèle-série globale d'un round AES.

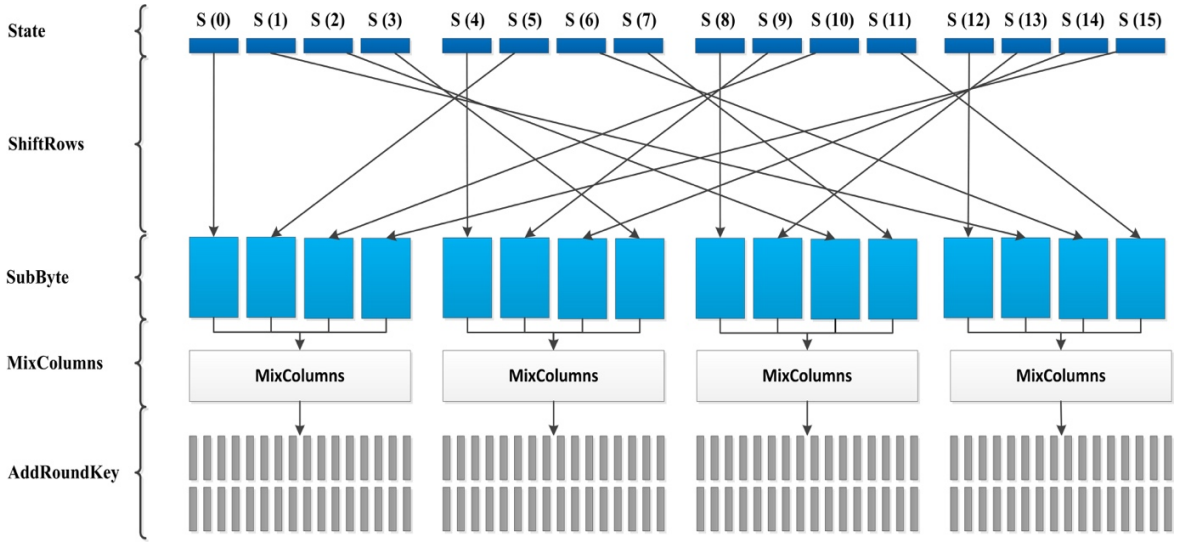


Figure 3.1. Architecture parallèle-série d'un cycle AES.

III.3. La partie de chiffrement

Le processus de chiffrement opère sur un message de 128 bits et une clé de chiffrement de même taille, nous pouvons la voir de l'extérieur comme étant une boite noire paramétrée par deux matrices carrées « STATE » et « KEY », et une matrice de sortie qui représente le texte chiffré.

La matrice « STATE » est représenté comme suit :

$$STATE = \begin{bmatrix} S_{15} & S_{11} & S_7 & S_3 \\ S_{14} & S_{10} & S_6 & S_2 \\ S_{13} & S_9 & S_5 & S_1 \\ S_{12} & S_8 & S_4 & S_0 \end{bmatrix}$$

La manière d'accéder au données de la matrice STATE est effectuée par diagonale octet par octet en commençant par S_{15}.

La première opération exercée sur la matrice « STATE » est un ou-exclusif (XOR) bit par bit avec la clé de chiffrement « KEY », le résultat intermédiaire va subir les mêmes transformations avec quatre blocs de 32 bits exécutés en parallèle, comme le montre la figure 3.2. Afin d'effectuer le bouclage des neuf rounds, nous avons rajouté à notre système un multiplexeur (2: 1) contrôlé par un signal de sélection « sel1 » qui se présente juste après l'opérateur XOR. Son rôle consiste à différencier entre le résultat du premier XOR pendant l'itération initiale et le résultat intermédiaire pendant les neufs rounds.

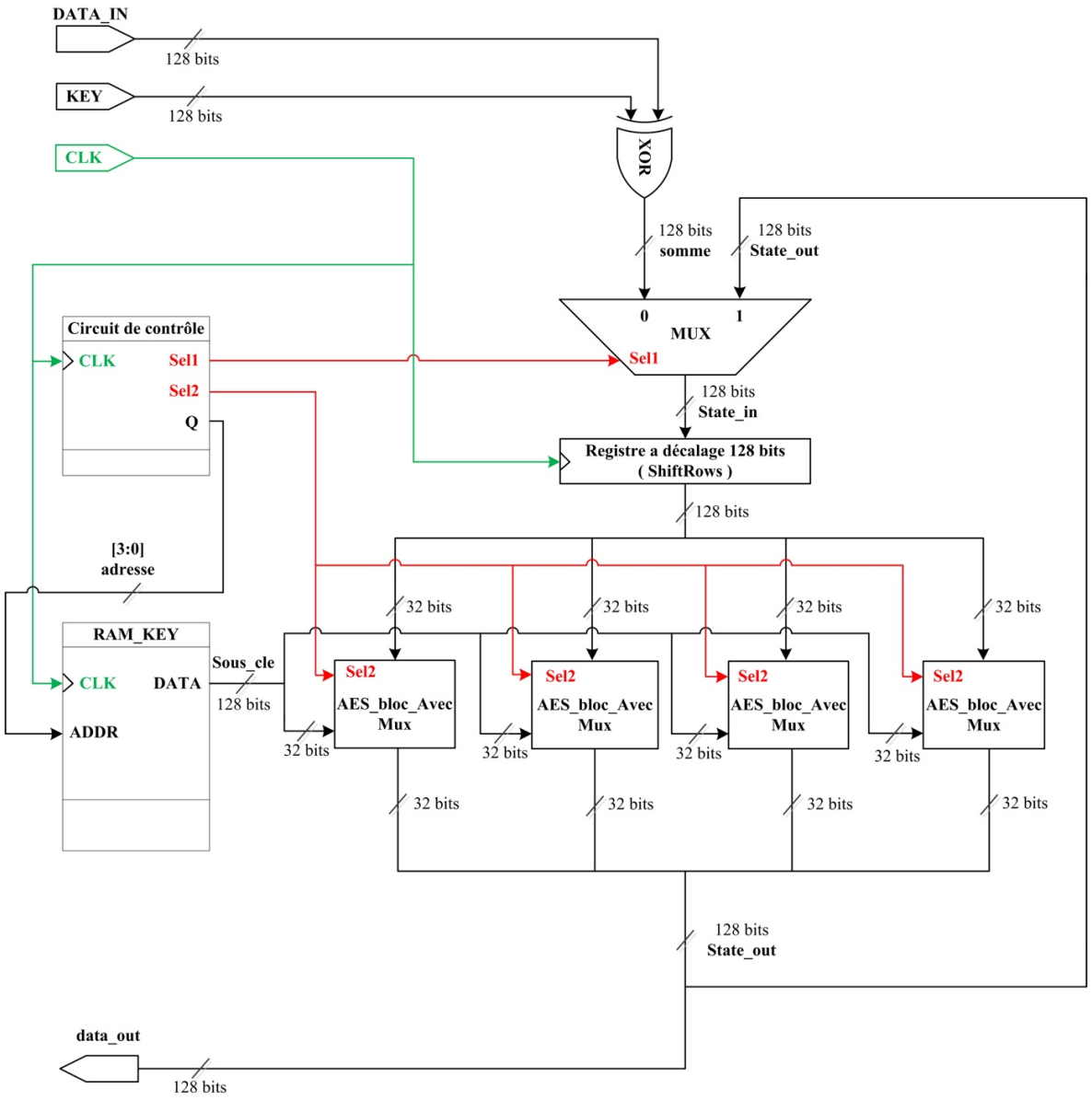


Figure 3.2. Architecture parallèle-série pour le chiffrement AES.

Pour la réalisation de l'opération *ShiftRow*, la matrice « STATE » traversera un registre à décalage synchronisé par une horloge « CLK » et une matrice en sortie, comme le montre le système d'équations suivant :

$$\begin{cases} W_1 = S_{15}, S_{14}, S_{13}, S_{12} \\ W_2 = S_{11}, S_{10}, S_9, S_8 \\ W_3 = S_7, S_6, S_5, S_4 \\ W_4 = S_3, S_2, S_1, S_0 \end{cases}$$

Après que l'étape *ShiftRows* est achevée, chaque colonne de la matrice intermédiaire state traversera un bloc de 32 bits qui contient les trois transformations : *SubByte*, *MixColumns* et *AddRoundKey*. Ce bloc de 32 bits sera dupliqué quatre fois pour avoir quatre blocs travaillant en parallèle.

Nous avons aussi rajouté un circuit de contrôle : ce dernier permet la génération des deux signaux de sélection « sel1 » et « sel2 » ainsi que l'adressage de la mémoire RAM_KEY. Cette dernière contient les différentes sous-clés générées par le processus *Key Expansion* qui va être détaillé ultérieurement.

Après avoir donné la description de l'architecture globale, nous passons à un niveau supérieur dans cette hiérarchie où nous rassemblons les trois opérations *SubByte*, *MixColumn* et *AddRoundKey* dans un seul composant AES_Bloc_avecMux pour les dix cycles. Le schéma bloc représentatif de son architecture est montré sur la figure 3.3.

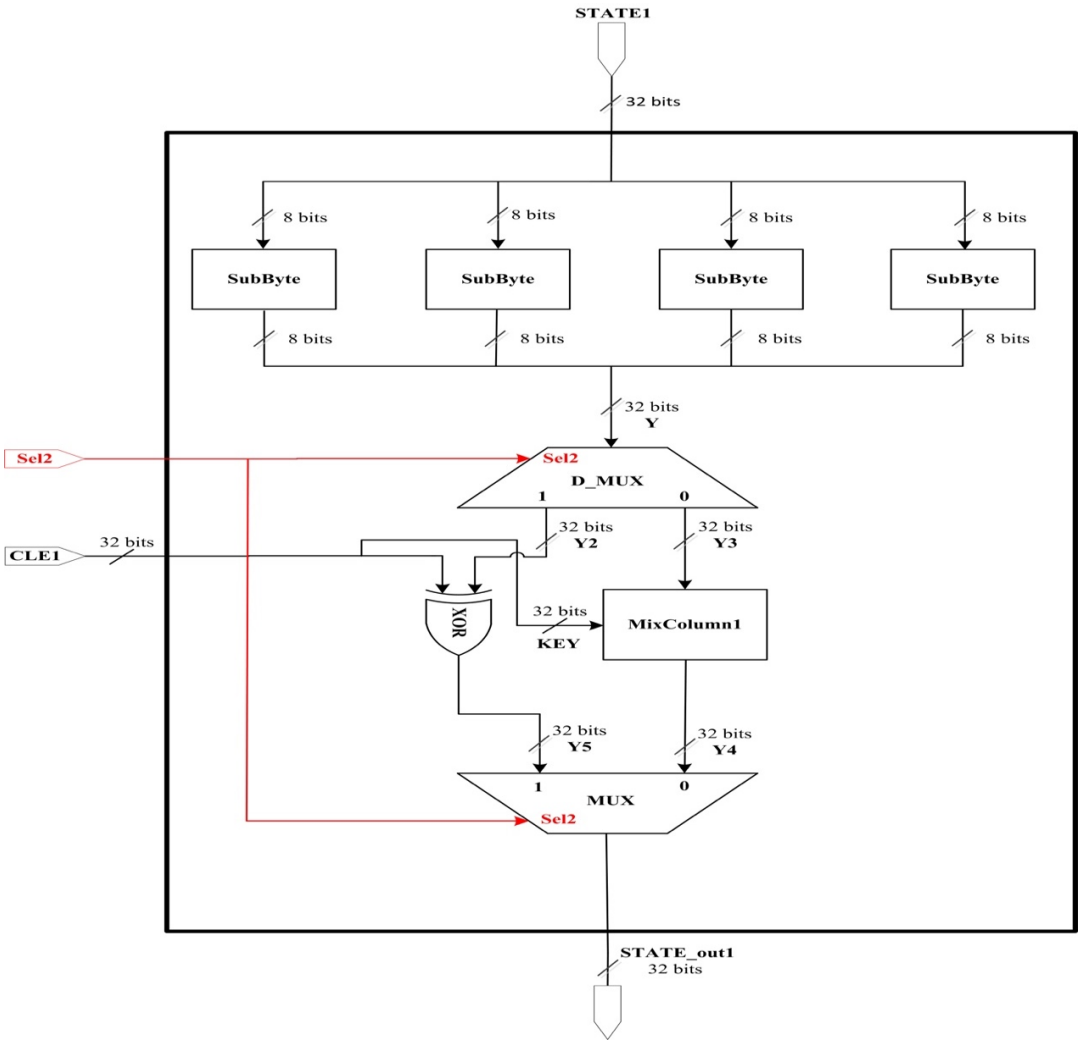


Figure 3.3. Architecture d'un bloc AES 32 bits.

A l'entrée de ce schéma, la donnée va subir en premier lieu une transformation *SubByte*. Les 32 bits d'entrée vont être subdivisés en quatre vecteurs de 8 bits dont le traitement se fait en parallèle.

Nous savons auparavant que le dernier cycle de l'AES ne comporte pas la transformation *MixColumns*, pour cela nous avons rajouté un Mux et D_Mux avec un signal de sélection « Sel2 », à l'intérieur de chaque bloc de 32 bits.

Afin d'implémenter la *SubByte*, nous avons utilisé le composant de plus bas niveau du circuit FPGA à savoir les LuT. Ce dernier est représenté comme étant une mémoire avec des valeurs pré-calculées. Ce mécanisme permet de retourner une valeur précise de la table *Sbox* en fonction de l'entrée de 8 bits [ANA 15].

Le schéma 3.4 montre le circuit de l'implémentation du *Sbox* au niveau le plus bas du circuit FPGA.

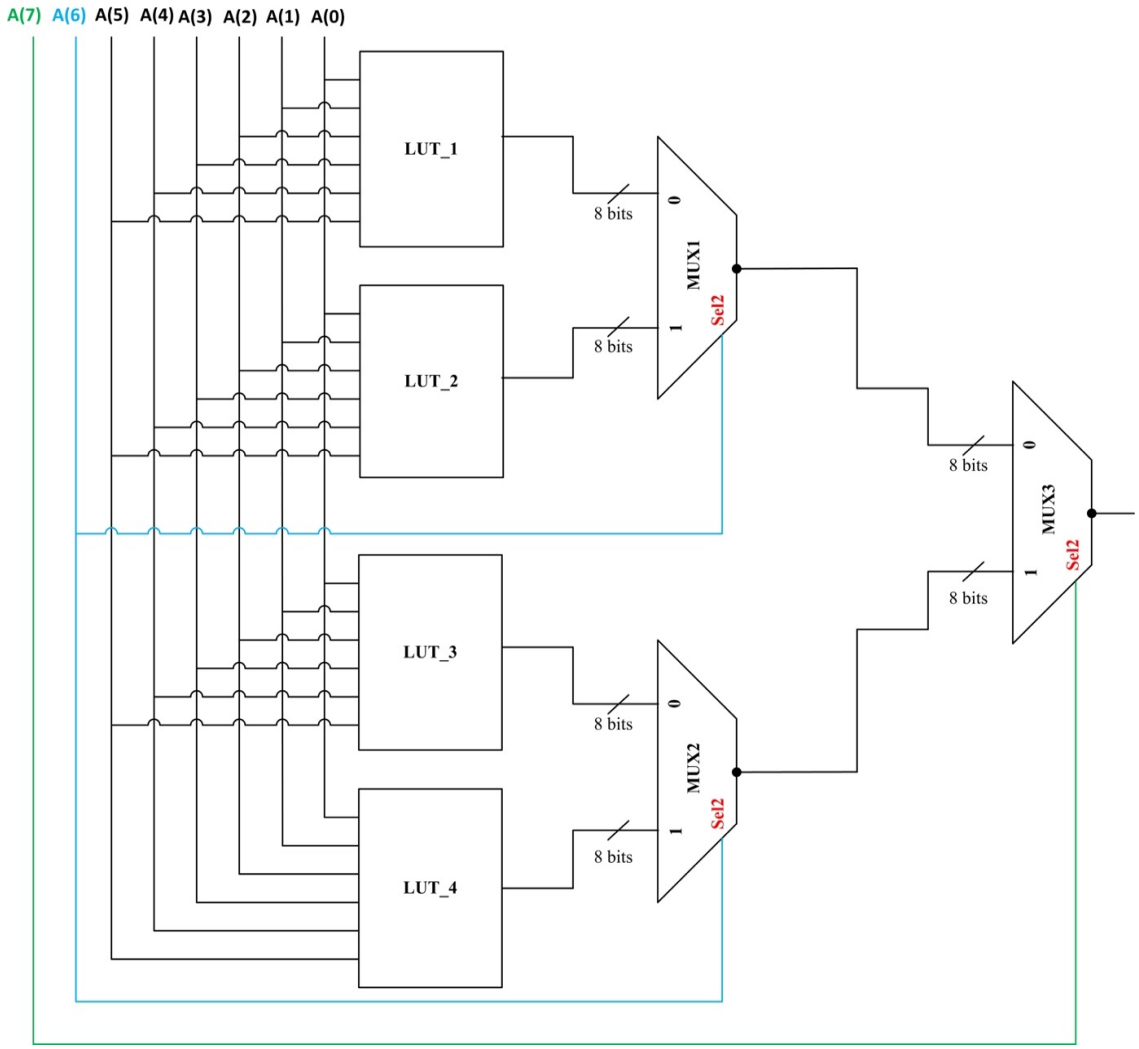


Figure 3.4. Architecture du circuit *Sbox* [ANA 15].

La sortie de ce circuit représente un bit du résultat de *Sbox*, donc pour avoir le résultat complet de 8 bits il nous faut 32 LUTs. Les quatre 8 bits du résultat forment un vecteur de 32 bits qui va passer par la suite par la transformation *MixColumns*.

Comme il a été vu dans le chapitre précédent, *MixColumns* représente un produit matriciel entre chaque vecteur de 32 bits de la matrice « STATE » et la matrice circulante prédéfinie.

En prenant la quatrième colonne de la matrice state après l'étape *Sbox* comme suit : $W_4 = E_3, E_2, E_1, E_0$. Le produit matriciel entre le vecteur W_4 et la matrice circulante nous donne le système d'équation suivant.

$$\begin{cases} S_0 = (\{02\} * E_0) \oplus (\{03\} * E_1) \oplus E_3 \oplus E_3 \\ S_1 = E_0 \oplus (\{02\} * E_1) \oplus (\{03\} * E_2) \oplus E_3 \\ S_2 = E_0 \oplus E_1 \oplus (\{02\} * E_2) \oplus (\{03\} * E_3) \\ S_3 = (\{03\} * E_0) \oplus E_1 \oplus E_2 \oplus (\{02\} * E_3) \end{cases} \text{------} \quad \textbf{(1)}$$

Le calcul du *MixColumns* revient à résoudre ce système d'équations. D'après ce système, nous constatons que deux opérations de base sont nécessaires, l'opérateur XOR et la multiplication par une constante.

Notre travail consiste à simplifier ces multiplications en fonction de l'opérateur XOR.

En commençant, soit B un élément de GF (2^8), cet élément peut être représenté par le polynôme suivant : $b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0$ / b_i ε {0, 1}. $b_i \in \{0, 1\}$.

- **La multiplication de B*01**

La multiplication par 01 désigne l'élément neutre du corps du Galois GF (2^8). Cette multiplication retourne la même valeur.

- **La multiplication de B*02**

C'est la multiplication de B*{X} modulo le polynôme irréductible [$x^8+x^4+x^3+x+1$].

Pour calculer le produit ci-dessus, nous procédons comme suit :

$$\begin{array}{lr} & b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0 \\ \times & x \\ \hline & b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x + b_7 \\ \textbf{Mod} & x^8+x^4+x^3+x+1 \\ \hline & b_6x^7+b_5x^6+b_4x^5+ (b_7+ b_3)\, x^4+ (b_7+ b_2)\, x^3+ b_1x^2+ (b_7+ b_0)\, x+ b_7 \end{array}$$

Nous obtenons le résultat suivant: $\{s_0=b_7,\ s_1=b_0+b_7,\ s_2=b_1,\ s_3=b_2+b_7,\ s_4=b_3+b_7,\ s_5=b_4,\ s_6=b_5,\ s_7=b_6\}$.

- **La multiplication de B*03**

Celle-ci revient à multiplier B par {X+1}, modulo le polynôme irréductible. Le résultat est :

$$
\begin{array}{rr}
 & (b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0) \\
\times & x+1 \\
\hline
 & b_7x^8+(b_6+b_7)x^7+(b_5+b_6)x^6+(b_4+b_5)x^5+(b_3+b_4)x^4+(b_2+b_3)x^3+(b_2+b_1)x^2+(b_0+b_1)x+b_0 \\
\textbf{Mod} & x^8+x^4+x^3+x+1 \\
\hline
 & (b_6+b_7)x^7+(b_5+b_6)x^6+(b_4+b_5)x^5+(b_3+b_4+b_7)x^4+(b_2+b_3+b_7)x^3+(b_2+b_1)x^2+(b_0+b_1+b_7)x+(b_0+b_7)
\end{array}
$$

Nous obtenons par la suite: $\{s_0=b_0+b_7,\ s_1=b_0+b_1+b_7,\ s_2=b_1+b_2,\ s_3=b_2+b_3+b_7,\ s_4=b_3+b_4+b_7,\ s_5=b_4+b_5,\ s_6=b_5+b_6,\ s_7=b_6+b_7\}$.

En utilisant le résultat du produit matriciel de *MixColumns*, représenté en dessus par le système d'équations (1), et en remplaçant chaque multiplication par son équivalent en fonction des opérateurs XOR, nous aurons les équations de chaque bit de sortie $\{S_0(0), S_0(1), \ldots, S_3(6), S_3(7)\}$ en fonction des bits d'entrées $\{E_0(0), E_0(1), \ldots, E_3(6), E_3(7)\}$.
Le système donné pour les premiers 8 bits de la sortie S(0) est :

$$
\begin{cases}
S_0(0) = E_0(7) \oplus E_1(0) \oplus E_1(7) \oplus E_2(0) \oplus E_3(0) & <5\text{ var}> \\
S_0(1) = E_0(0) \oplus E_0(7) \oplus E_1(0) \oplus E_1(1) \oplus E_1(7) \oplus E_2(1) \oplus E_3(1) & \mathbf{<7\text{ var}>} \\
S_0(2) = E_0(1) \oplus E_1(1) \oplus E_1(2) \oplus E_2(2) \oplus E_3(2) & <5\text{ var}> \\
S_0(3) = E_0(2) \oplus E_0(7) \oplus E_1(2) \oplus E_1(3) \oplus E_1(7) \oplus E_2(3) \oplus E_3(3) & \mathbf{<7\text{ var}>} \\
S_0(4) = E_0(3) \oplus E_0(7) \oplus E_1(3) \oplus E_1(4) \oplus E_1(7) \oplus E_2(4) \oplus E_3(4) & \mathbf{<7\text{ var}>} \\
S_0(5) = E_0(4) \oplus E_1(4) \oplus E_1(5) \oplus E_2(5) \oplus E_3(5) & <5\text{ var}> \\
S_0(6) = E_0(5) \oplus E_1(5) \oplus E_1(6) \oplus E_2(6) \oplus E_3(6) & <5\text{ var}> \\
S_0(7) = E_0(6) \oplus E_1(6) \oplus E_1(7) \oplus E_2(7) \oplus E_3(7) & <5\text{ var}>
\end{cases} \quad \textbf{(2)}
$$

Le même procédé est appliqué pour les autres sorties S(1), S(2) et S(3) [SAM 02].

Nous pouvons implémenter ce système en utilisant directement les opérateurs XOR comme le montre la figure 3.5.

Dans ce cas de figure, l'outil de synthèse de Xilinx choisit l'architecture et le nombre des LUT nécessaires pour une telle implémentation.

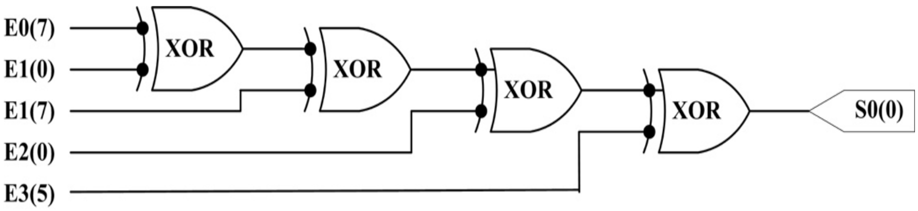


Figure 3.5. Implémentation de la sortie $S_0(0)$ par l'opérateur XOR.

Comme aussi, nous pouvons l'implémenter nous même sur des LUTs en précisant les entrées et la fonction appliquée sur chaque LuTs, comme il est illustré par la figure 3.6.

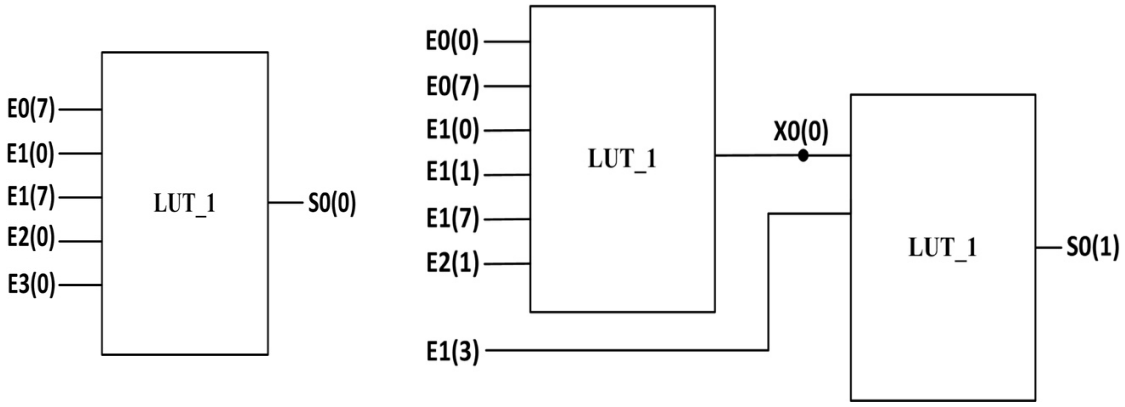


Figure 3.6. Implémentation de $S_0(0)$ et $S_0(1)$ par LuT.

La sortie $S_0(0)$ dépend de cinq variables d'entrées, donc une LuT5 suffira pour implémenter la première sortie $S_0(0)$. Par contre $S_0(1)$ dépend de 7 variables d'entrée, ce qui nous mène à utiliser une LuT6 et un MUX. Au total, l'architecture de la sortie S_0 a besoin de 8 LuTs et 3 MUXs.

De même, en rajoutant les sorties S(1), S(2) et S(3) restantes, notre architecture globale permettant le calcul de la *MixColumn*, nécessite 8×4 LuTs et 3×4 MUXs en terme de ressources matérielles et de délai de passage d'une seule LuT.

Ensuite, la matrice résultante effectue l'étape *AddRoundKey*, qui représente une addition bit par bit avec la sous-clé en utilisant l'opérateur XOR.

A ce stade, nous avons introduit l'opération XOR de chaque bit de la sous-clé avec le bit qui lui convient dans l'étape *MixColumn*, vu qu'elle est composée qu'avec des opérateurs XOR, comme le montre la figure3.7.

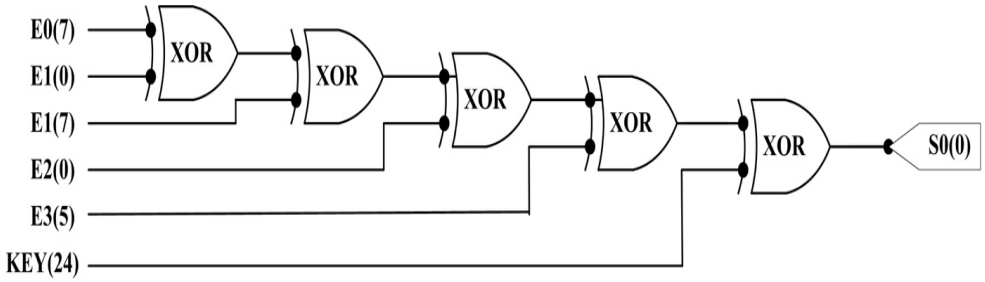


Figure 3.7. *MixColumn* et *AddRoundKey* pour $S_0(0)$ par l'opérateur XOR.

L'avantage de mixer les deux opérations *AddRoundKey* et *MixColumn*, se verra plus dans la deuxième architecture dans le cas où nous utilisons que des LuTs. A ce niveau-là, l'opération *AddRoundKey* peut être remplacée par l'ajout d'une seule entrée dans chaque LuTs non saturée, comme le montre la figure 3.8, d'une part, le temps d'exécution augmente et d'une autre part, le nombre des ressources utilisées diminue.

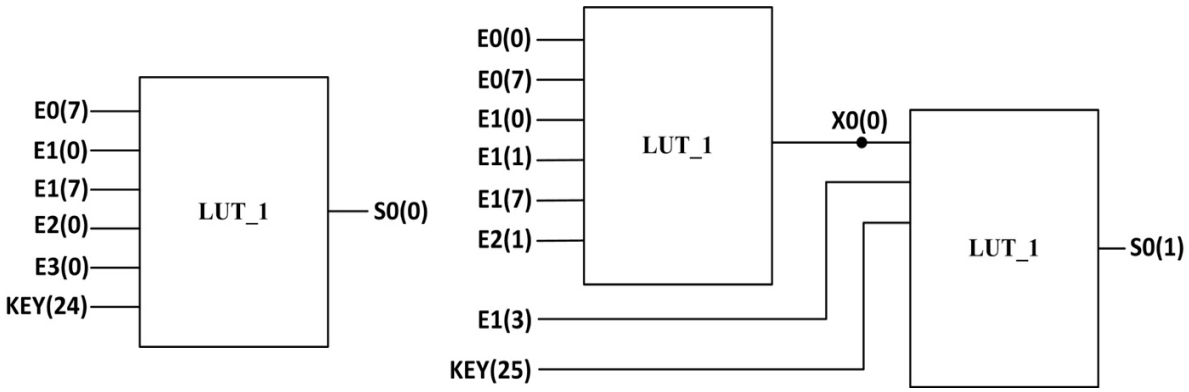


Figure 3.8. *MixColumn* et *AddRoundKey* pour $S_0(0)$ et $S_0(1)$ par LuT.

III.4. La partie de déchiffrement

Cette partie contient la même architecture de blocs que celle du chiffrement, sauf que les opérations effectuent un traitement inverse en traversant les transformations ; *InvShiftRows*, *InvSubByte*, *InvMixColumns* et *AddRoundKey*.

Elle est représentée par une boite noire paramétrée par deux matrices carrées en entrée, « STATE » qui représente dans ce cas la donnée chiffrée et « KEY » qui représente la clé pour le déchiffrement, et une matrice en sortie représente la donnée en claire. La **figure 3.9** illustre l'architecture du déchiffrement de l'AES.

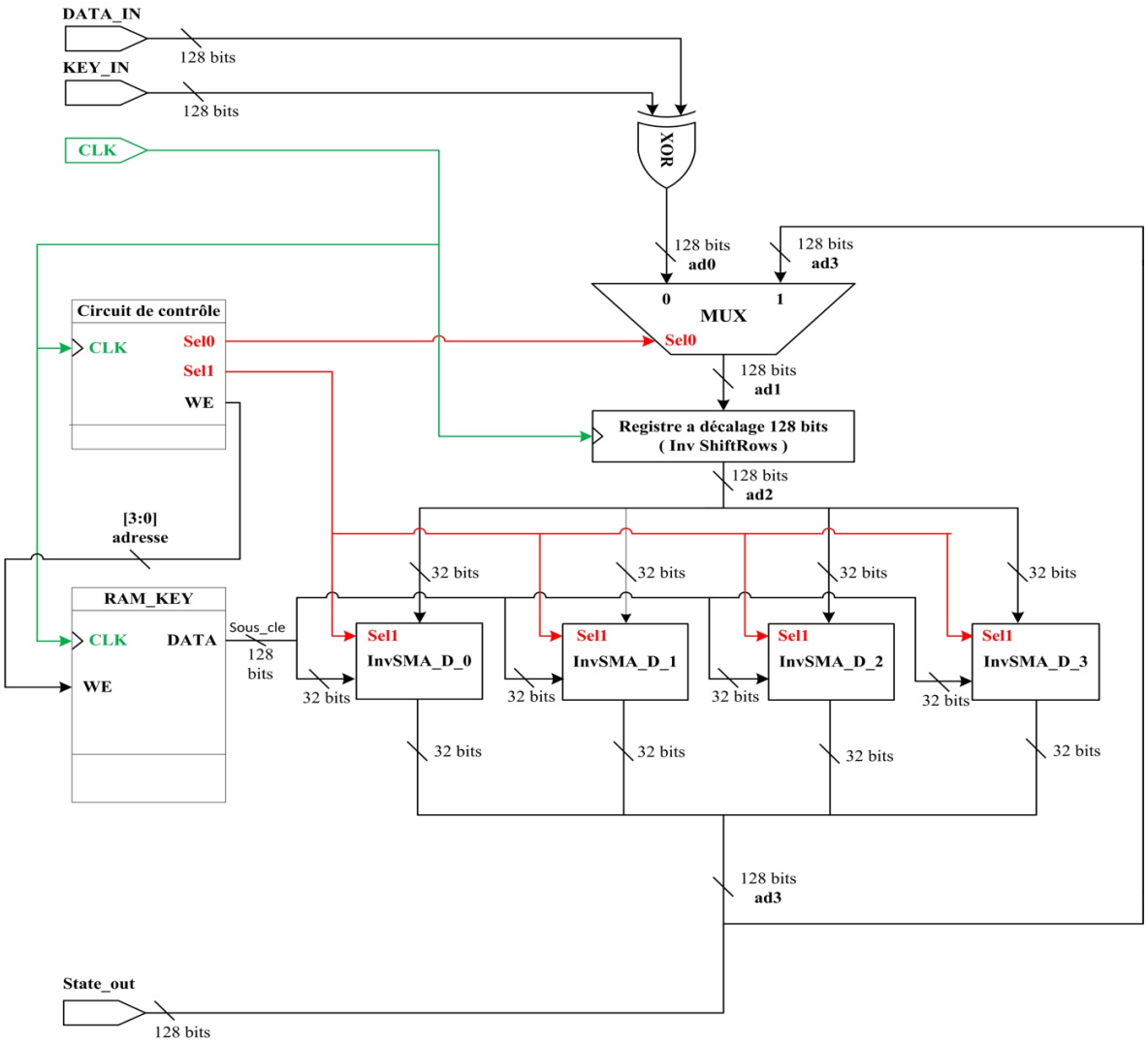


Figure 3.9. Architecture du déchiffrement de l'AES.

Nous remarquons que c'est exactement la même architecture du chiffrement, sauf que l'appellation prend les noms des opérations inverses.

La même chose pour le circuit de contrôle, la RAM_KEY et les signaux de sélection, ils ont le même fonctionnement que le chiffrement, sauf que le compteur d'adresse de la RAM_KEY, commence par la dernière adresse au lieu de la première dans le cas de déchiffrement car l'accès à la RAM_KEY pour le cas de chiffrement sera de haut en bas et cela sera inversé pour le cas de déchiffrement.

La matrice d'entrée effectue en premier lieu un XOR avec la dernière sous-clé générée. Puis, elle passe par les mêmes transformations pendant les dix cycles. C'est pour la même raison que la même architecture est conservée.

Par la suite, la matrice traverse un registre à décalage qui effectue cette fois-ci un décalage inverse sous le nom *InvShiftRows*.

L'architecture d'un bloc AES de 32 bits pour le cas de déchiffrement est montrée dans la figure 3.10.

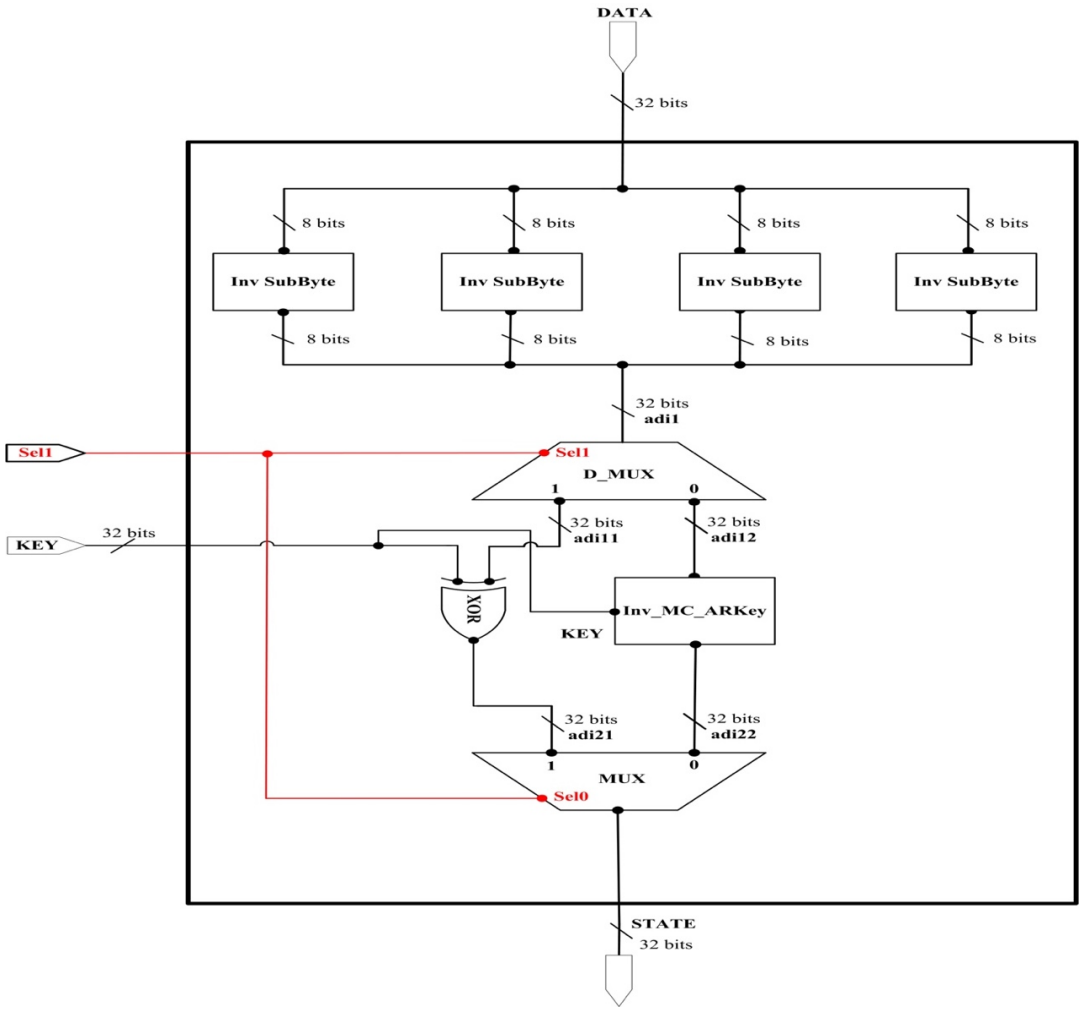


Figure 3.10. Architecture d'un bloc AES 32 bits pour le déchiffrement.

Vu que les deux tables *Sbox* et son inverse ont la même taille, nous avons gardé la même architecture d'implémentation illustrée sur la figure 3.4.

De même pour l'implémentation de l'*InvMixColumns*, nous devons écrire chaque bit de sortie en fonction des bits d'entrées en utilisant que des portes XORs. Le calcul de l'*InvMixColumns* revient à résoudre le système d'équations suivant ;

$$\begin{cases} S_0 = (\{0E\} * E_0) \oplus (\{0B\} * E_1) \oplus (\{0D\} * E_2) \oplus (\{09\} * E_3) \\ S_1 = (\{09\} * E_0) \oplus (\{0E\} * E_1) \oplus (\{0B\} * E_2) \oplus (\{0D\} * E_3) \\ S_2 = (\{0D\} * E_0) \oplus (\{09\} * E_1) \oplus (\{0E\} * E_2) \oplus (\{0B\} * E_3) \\ S_3 = (\{0B\} * E_0) \oplus (\{0D\} * E_1) \oplus (\{09\} * E_2) \oplus (\{0E\} * E_3) \end{cases} \quad \text{(3)}$$

Pour ce faire, nous devons simplifier les opérations de multiplication comme nous l'avons fait avec l'implémentation du MixColumns. Nous remarquons que la multiplication se fait par des valeurs plus grandes que celle du MixColumns, donc elle prendra plus de temps d'exécution et occupera plus de ressources.

- **La multiplication de B*09**

C'est une multiplication de deux polynômes dans GF(2^8). Donc l'entrée B sera multipliée par $\{x^3+1\}$ modulo le polynôme irréductible $[x^8+x^4+x^3+x+1]$.

$$\begin{array}{r} (b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0 \\ \times \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad x^3+1 \\ \hline b_7x^{10}+b_6x^9+b_5x^8+b_4x^7+b_3x^6+b_2x^5+b_1x^4+b_0x^3+b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0 \\ \textbf{Mod} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad x^8+x^4+x^3+x+1 \\ \hline (b_4+b_7)x^7+(b_3+b_6+b_7)x^6+(b_2+b_5+b_6+b_7)x^5+(b_1+b_4+b_5+b_6)x^4+(b_0+b_3+b_5+b_7)x^3+(b_2+b_6+b_7)x^2+ \\ (b_1+b_5+b_6)x + (b_0+b_5). \end{array}$$

Nous obtenons le système suivant: $\{s_0=b_5+b_6,\ s_1=b_5+b_7,\ s_2=b_0+b_6,\ s_3=b_0+b_1+b_5+b_6+b_7,\ s_4=b_1+b_2+b_5+b_7,\ s_5=b_2+b_3+b_6,\ s_6=b_3+b_4+b_7,\ s_7=b_4+b_5\}$.

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

C'est de multiplier B*$\{x^3+x+1\}$ modulo le polynôme irréductible. Nous obtenons le système suivant: $\{s_0= b_0+b_5+b_7,\ s_1= b_0+b_1+b_5+b_6+b_7,\ s_2= b_1+b_2+b_6+b_7,\ s_3= b_0+b_2+b_3+b_5,\ s_4= b_1+b_3+b_4+b_5+b_6+b_7,\ s_5= b_2+b_4+b_5+b_6+b_7,\ s_6= b_3+b_5+b_6+b_7,\ s_7= b_4+b_6+b_7\}$.

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

Cette multiplication revient à calculer B*$\{x^3+x^2+1\}$ modulo le polynôme irréductible. Le système obtenu est: $\{s_0=b_0+b_5+b_6,\ s_1=b_1+b_5+b_7,\ s_2=b_0+b_2+b_6,\ s_3=b_0+b_1+b_3+b_5+b_6+b_7,\ s_4=b_1+b_2+b_4+b_5+b_7,\ s_5=b_2+b_3+b_5+b_6,\ s_6=b_3+b_4+b_6+b_7,\ s_7=b_4+b_5+b_7\}$.

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

Celle-ci revient à calculer B*$\{x^3+x^2+1\}$.

Donc, le système obtenu est: $\{s_0=b_0+b_5+b_6,\ s_1=b_1+b_5+b_7,\ s_2=b_0+b_2+b_6,\ s_3=b_0+b_1+b_3+b_5+b_6+b_7,\ s_4=b_1+b_2+b_4+b_5+b_7,\ s_5=b_2+b_3+b_5+b_6,\ s_6=b_3+b_4+b_6+b_7,\ s_7=b_4+b_5+b_7\}$.

En utilisant le résultat du produit matriciel de l'*InvMixColumns* représenté par le système d'équation (2), et en remplaçant chaque multiplication par son équivalent en fonction des opérateurs XOR, nous avons les équations de chaque bit de sortie {$S_0(0)$, $S_0(1)$, …, $S_3(6)$, $S_3(7)$} en fonction des bits d'entrées {$E_0(0)$, $E_0(1)$, …, $E_3(6)$, $E_3(7)$} [SAM 02].

On aura le système suivant pour les premiers 8 bits de sortie S(0) :

$$
\begin{cases}
S_0(0) = E_0(5) \oplus E_0(6) \oplus E_0(7) \oplus E_1(0) \oplus E_1(5) \oplus E_1(7) \oplus E_2(0) \oplus E_2(5) \oplus E_2(6) \oplus E_3(0) \oplus E_3(5) \quad < 11\ var > \\
S_0(1) = E_0(0) \oplus E_0(5) \oplus E_1(0) \oplus E_1(1) \oplus E_1(5) \oplus E_1(6) \oplus E_1(7) \oplus E_2(1) \oplus E_2(5) \oplus E_2(7) \oplus E_3(1) \oplus E_3(5) \oplus E_3(6) \quad < 13\ var > \\
S_0(2) = E_0(0) \oplus E_0(1) \oplus E_0(6) \oplus E_1(1) \oplus E_1(2) \oplus E_1(6) \oplus E_1(7) \oplus E_2(0) \oplus E_2(2) \oplus E_2(6) \oplus E_3(2) \oplus E_3(6) \oplus E_3(7) \quad < 13\ var > \\
S_0(3) = E_0(0) \oplus E_0(1) \oplus E_0(2) \oplus E_0(5) \oplus E_0(6) \oplus E_1(0) \oplus E_1(2) \oplus E_1(3) \oplus E_1(5) \oplus E_2(0) \oplus E_2(1) \oplus E_2(3) \oplus E_2(5) \oplus E_2(6) \oplus E_2(7) \oplus E_3(0) \oplus E_3(3) \oplus E_3(5) \oplus E_3(7) \quad < 19\ var > \quad (4) \\
S_0(4) = E_0(1) \oplus E_0(2) \oplus E_0(3) \oplus E_0(5) \oplus E_1(1) \oplus E_1(3) \oplus E_1(4) \oplus E_1(5) \oplus E_1(6) \oplus E_1(7) \oplus E_2(1) \oplus E_2(2) \oplus E_2(4) \oplus E_2(5) \oplus E_2(7) \oplus E_3(1) \oplus E_3(4) \oplus E_3(5) \oplus E_3(6) \quad < 19\ var > \\
S_0(5) = E_0(2) \oplus E_0(3) \oplus E_0(4) \oplus E_0(6) \oplus E_1(2) \oplus E_1(4) \oplus E_1(5) \oplus E_1(6) \oplus E_1(7) \oplus E_2(2) \oplus E_2(3) \oplus E_2(5) \oplus E_2(6) \oplus E_3(2) \oplus E_3(5) \oplus E_3(6) \oplus E_3(7) \quad < 17\ var > \\
S_0(6) = E_0(3) \oplus E_0(4) \oplus E_0(5) \oplus E_0(7) \oplus E_1(3) \oplus E_1(5) \oplus E_1(6) \oplus E_1(7) \oplus E_2(3) \oplus E_2(4) \oplus E_2(6) \oplus E_2(7) \oplus E_3(3) \oplus E_3(6) \oplus E_3(7) \quad < 15\ var > \\
S_0(7) = E_0(4) \oplus E_0(5) \oplus E_0(6) \oplus E_1(4) \oplus E_1(6) \oplus E_1(7) \oplus E_2(4) \oplus E_2(5) \oplus E_2(7) \oplus E_3(4) \oplus E_3(7) \quad < 11\ var >
\end{cases}
$$

Maintenant nous pouvons implémenter les quatre systèmes d'équations {S_0, S_1, S_2, S_3} en utilisant que l'opérateur XOR et en appliquant la même idée permettant de mixer les deux opérations *AddRoundKey* et *InvMixColumn*, comme le montre la figure 3.1.

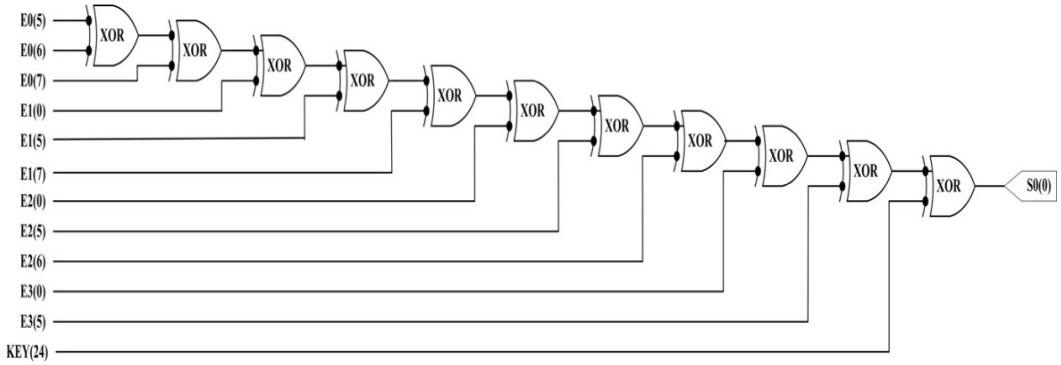


Figure 3.11. *InvMixColumn* et *AddRoundKey* pour $S_0(0)$ par XOR.

Comme nous pouvons aussi implémenter ces quatre systèmes d'équations {S_0, S1, S2, S3} directement de la même manière effectuée pour la *MixColumn*, en rajoutant l'opération *AddRoundKey*. La Figure 3.12 montre une implémentation matérielle de l'*InvMixColumn* et *AddRoundKey* en utilisant les LuTs.

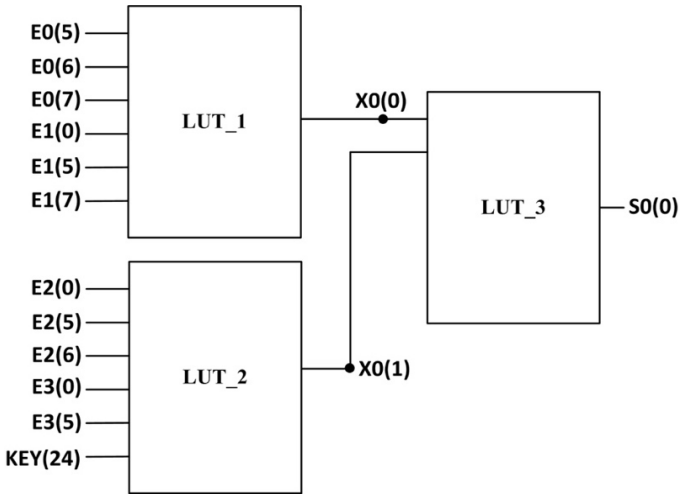


Figure 3.12. *InvMixColumns* et *AddRoundKey pour $S_0(0)$ par LuT.*

L'implémentation des quatre équations occupe 27×4=108 LuTs, mais nous avons pu optimiser d'avantage en mettant quelques LuTs en facteur commun car elles réalisent la même fonction logique à base de XOR.

Pour ce faire, nous avons représenté ces quatre système d'équations {S_0, S_1, S_2, S_3} en fonction des entrées {E_0, E_1, E_2, E_3}, ou le produit matriciel du vecteur {$E_0(0)$, $E_0(1)$, …, $E_3(6)$, $E_3(7)$} par une matrice carrée de 32 éléments binaires donne en résultat le système d'équations (4), comme le montre la figure ci-dessous [AZZ 14] ;

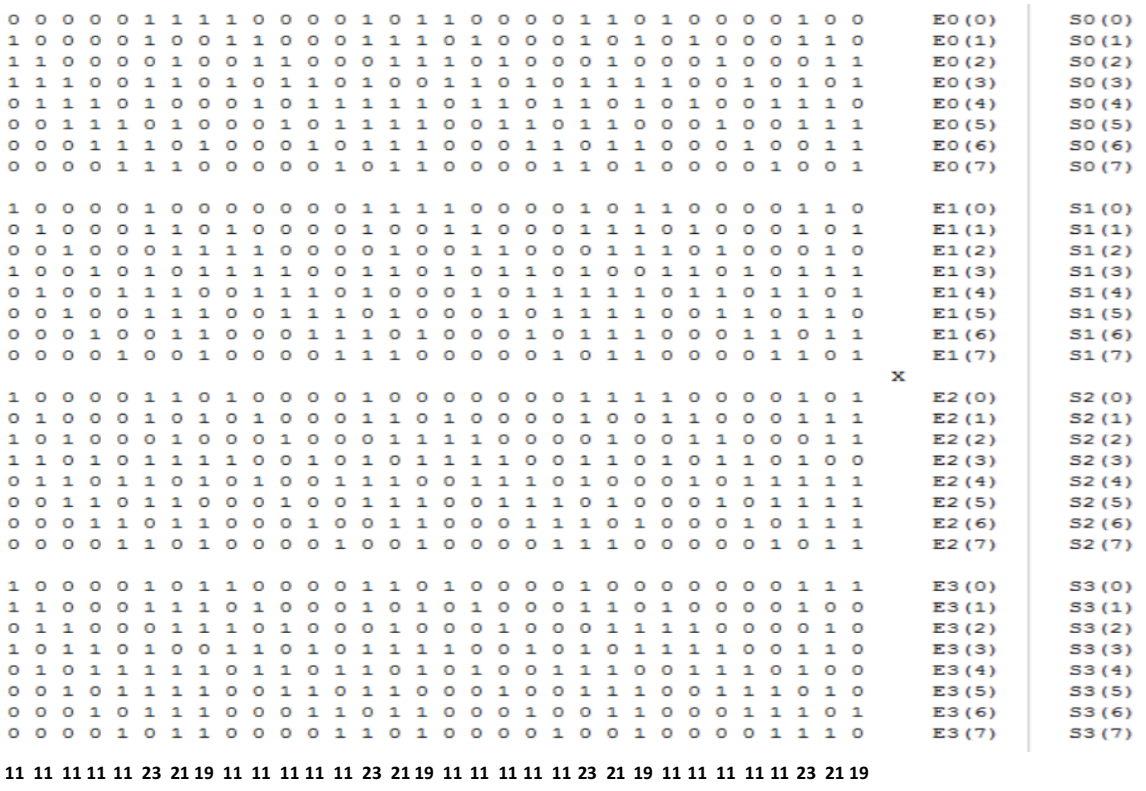


Figure 3.13. Représentation du produit matriciel de l'*InvMixColumns* [AZZ 14].

La première ligne de la matrice carrée représente la sortie $S_0(0)$, et la dernière ligne représente $S_3(7)$. Ainsi que la première colonne représente l'entrée $E_0(0)$, et la dernière colonne représente $E_3(7)$.

En additionnant tous les éléments de la première colonne, le résultat est égale à onze, cela représente le taux d'apparition de l'entrée $E_0(0)$ dans le système d'équation (4), ce qui veut dire que cette entrée se répète onze fois dans ce système d'équation d'*InvMixColumns*.

La même chose est répétée pour le reste des colonnes, c.à.d. en calculant le taux d'apparition de chaque entrée $\{E_0(0), E_0(1), \ldots, E_3(6), E_3(7)\}$ dans tout le système d'équations (4).

Une fois terminé, et vu que le nombre d'entrées d'une LuT égale à six, nous sélectionnons les six entrées avec les plus grands taux d'apparition. Comme cela est illustré sur la figure qui suit

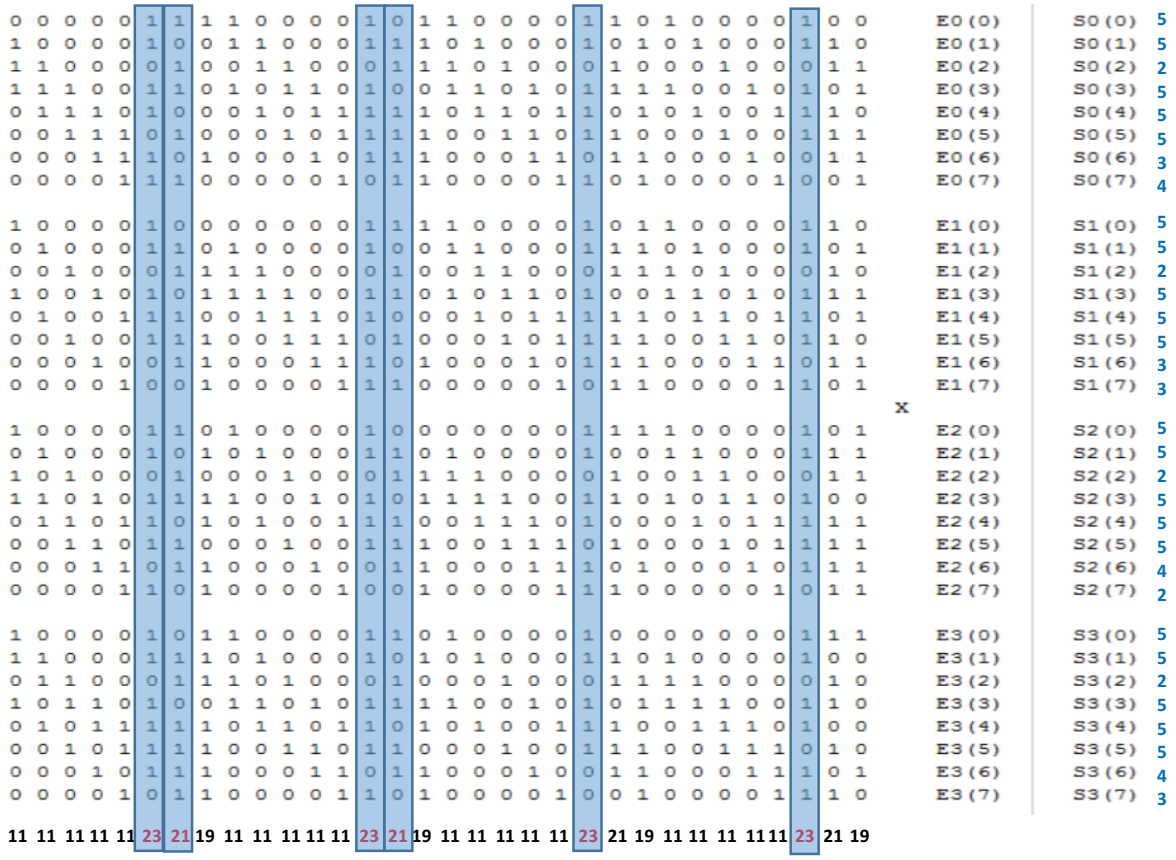


Figure 3.14. Représentation matricielle de la 1ere étape d'optimisation de l'*InvMixColumns*.

Après avoir sélectionné les six entrées avec les plus grands taux d'apparition, nous calculons le nombre de fois où ces dernières sont apparues pour chaque sortie du système d'équation (4).

D'après les résultats représentés en bleu devant chaque sortie, nous remarquons que nous n'avons aucune sortie contenant les six entrées sélectionnées à la fois. Pour cela, nous continuons de faire un balayage pour toutes les entrées en commençant par ceux qui ont les plus grands taux d'apparition d'abord.

Pour ce faire, nous avons sélectionné l'entrée qui vient juste après celle avec le plus petit taux d'apparition, comme le montre la figure ci-dessous ;

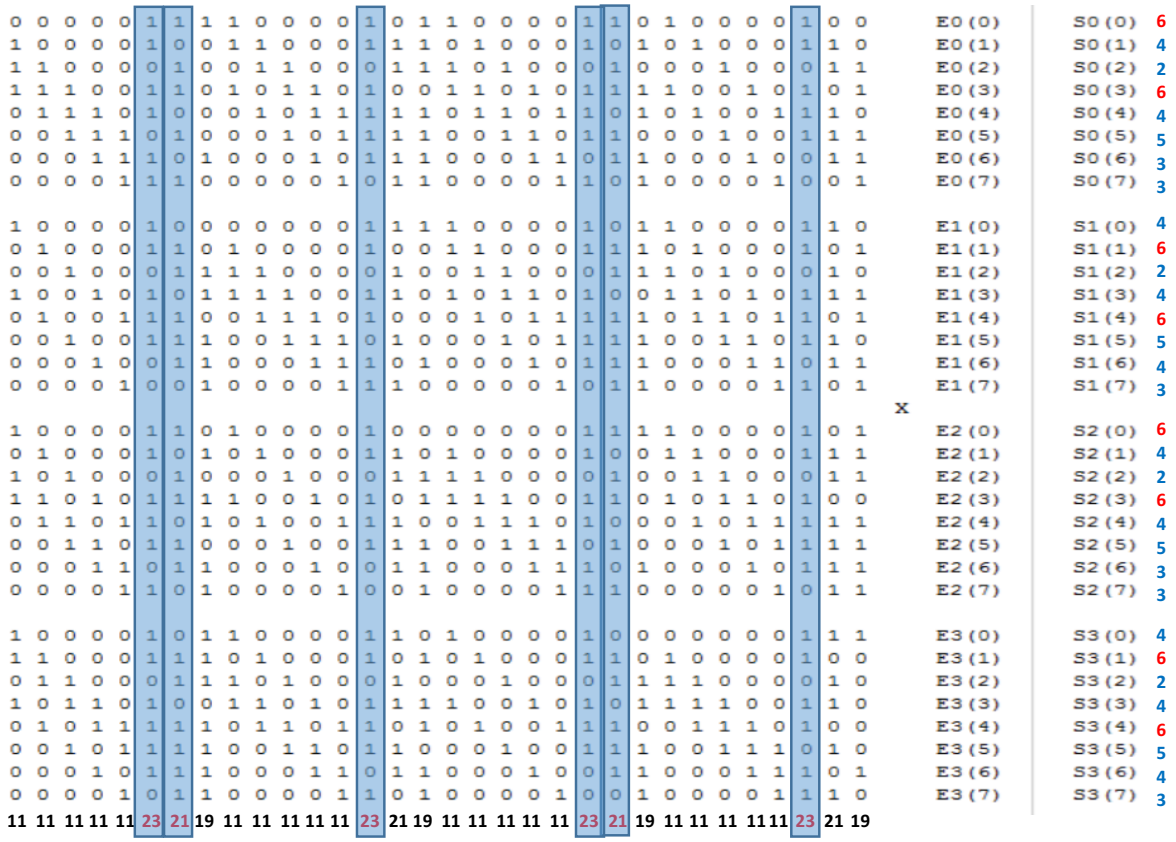


Figure 3.15. Représentation matricielle de la 2eme étape d'optimisation de l'*InvMixColumns*.

Nous remarquons d'après les résultats obtenus dans la figure ci-dessus, que nous avons cette fois-ci tous les six entrées sélectionnées {**$E_0(5)$**, **$E_0(6)$**, **$E_1(5)$**, **$E_1(6)$**, **$E_2(5)$**, **$E_3(5)$**} dans les sorties {**$S_0(1)$**, **$S_0(3)$**, **$S_1(1)$**, **$S_1(4)$**, **$S_2(0)$**, **$S_2(3)$**, **$S_3(1)$**, **$S_3(4)$**} du système d'équation (4). C'est-à-dire que dans chaque sortie de ces dernières, nous avons une LuT avec ces mêmes entrées qui exécutent la même fonction qui est XOR, donc elles vont avoir la même sortie que nous pouvons prendre en commun.

A chaque fois qu'il y a des entrées en commun, la valeur de ces entrées est mise à « 0 » puis le balayage est refait.

De cette façon, nous avons terminé le balayage de toute la matrice en abordant les entrées avec le plus grand taux d'apparition en premier lieu.

En mettant toutes les sorties ayant les même entrés en commun, nous avons pu diminuer le nombre de LUT de 108 jusqu'à 80 LUTs.

III.5. La partie de *KeyExpansion*

C'est l'étape pour laquelle les sous-clés sont générées et stockées dans une RAM_KEY. Ces clés sont utilisées pendant le processus de chiffrement et de déchiffrement. Le processus *KeyExpansion* peut être vu comme une boite noire paramétrée par une entrée de 128 bits qui représente la clé maîtresse. Cette dernière est représentée sous forme d'une matrice carrée, comme le représente la matrice suivante.

$$KEY = \begin{bmatrix} K_{15} & K_{11} & K_7 & K_3 \\ K_{14} & K_{10} & K_6 & K_2 \\ K_{13} & K_9 & K_5 & K_1 \\ K_{12} & K_8 & K_4 & K_0 \end{bmatrix}$$

La manière d'accéder aux données de cette matrice est effectuée par diagonale, octet par octet, en commençant par K_{15}.

Durant le processus de génération des sous-clés, une succession de transformations s'applique sur la matrice « KEY » afin de générer la première sous-clé. Les mêmes transformations seront appliquées d'une manière récursive permettant la génération des différentes sous-clés. A la fin, nous obtenons dix sous-clés de 128 bits qui seront enregistrées dans la RAM à chaque fois qu'une sous-clé se génère.

En ce qui concerne l'implémentation de *KeyExpansion*, la même architecture utilisée pour le cas de chiffrement et de déchiffrement de l'AES, a été proposée dans laquelle nous avons rajouté un circuit de contrôle permettant l'adressage des deux mémoires ROM et RAM. La ROM contient les valeurs de la table RC et la RAM est utilisée afin de sauvegarder les différents sous clés générées.

La figure 3.13 montre l'architecture proposée pour une implémentation matérielle de Key Expansion.

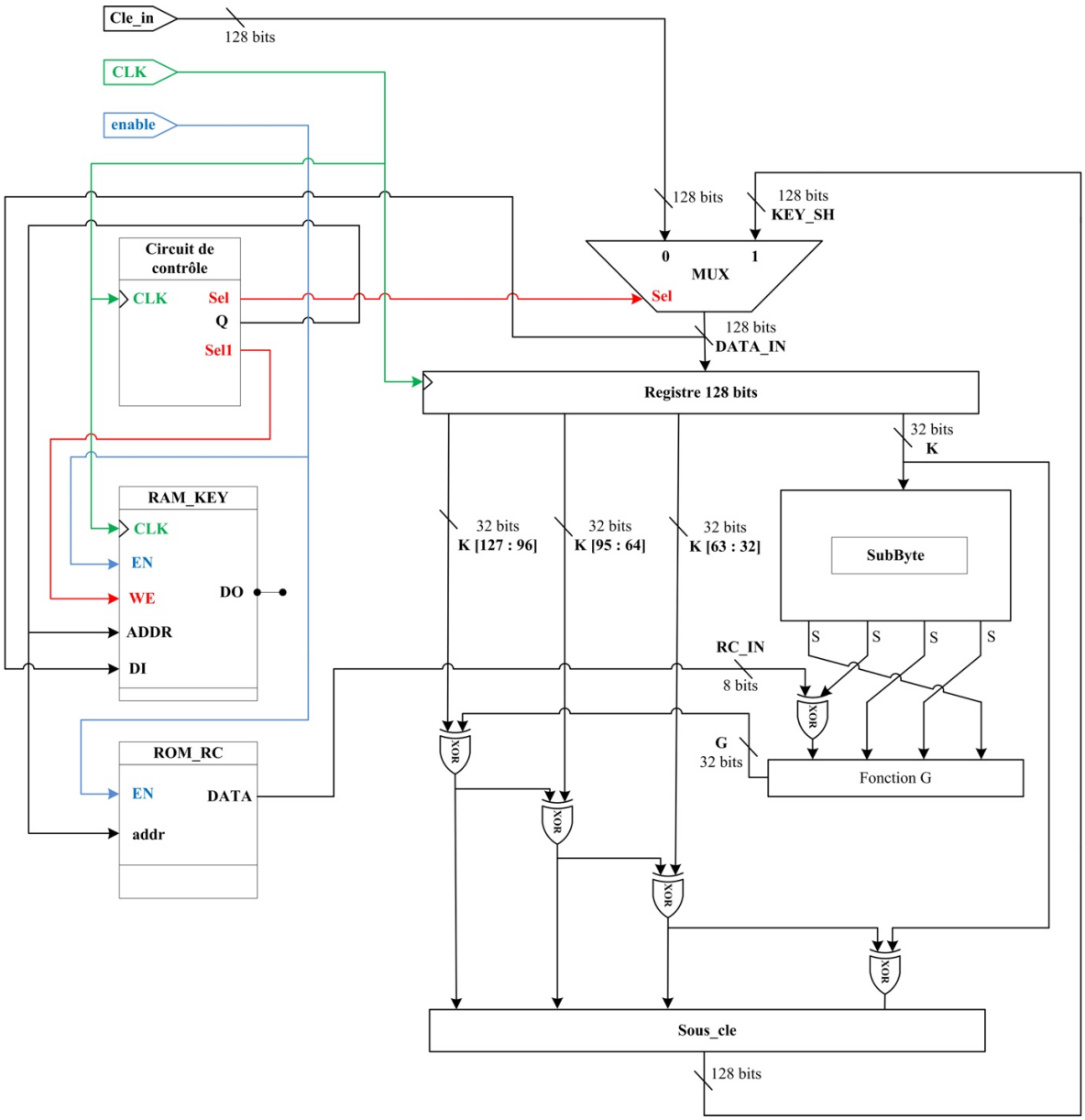


Figure 3.16. Architecture du circuit de génération des sous-clés.

III.5. Conclusion

Dans ce chapitre, nous avons présenté les trois architectures proposées pour une implémentation matérielle à savoir l'architecture de chiffrement/déchiffrement de l'AES et la génération des sous-clés. Nous avons vu aussi les différentes optimisations matérielles pour la *MixColumn*/*InvMixColumn* et la *SubByte*/*InvSubByte* en termes de temps d'exécution et de ressources occupées en utilisant uniquement les LuTs.

Dans le chapitre suivant, les résultats d'implémentation de notre crypto système seront discutés.

Chapitre IV

Synthèse & implémentation

IV.1 Introduction

Dans ce chapitre, nous exposons les résultats de simulation et d'implémentation des architectures conçues dans le chapitre précédent sur circuit FPGA de Xilinx de la famille Virtex-7 (xc7v585t-3ffg1157). Le langage de description matérielle (VHDL) a été utilisé pour la modélisation des architectures proposées. L'outil de simulation d'ISE a pu vérifier leur bon fonctionnement. En fin du chapitre, une discussion des différents résultats obtenus est effectuée.

IV.2 Les circuits FPGA

Les FPGAs (Field Programmable Gate Arrays) sont des puces de silicium reprogrammables inventés en 1984 par Xilinx [SAL 15]. Si les FPGAs rencontrent un tel succès dans tous les secteurs, c'est parce qu'ils combinent les meilleures caractéristiques des ASICs (Application-Specific Integrated Circuits) et des systèmes basés sur des processeurs. Ils offrent un cadencement par matériel qui leur assure une grande vitesse et de la fiabilité [PAP 12].

Les FPGAs sont des circuits intégrés digitaux qui ne possèdent pas une fonction prédéfinie par le fabriquant, sa fonction peut être définie (ou programmée) par l'utilisateur. En raison de la grande capacité de la logique de ces dispositifs, des systèmes complexes peuvent être développés sur un seul circuit intégré [BOU 15]. Ils sont vraiment parallèles par nature, de sorte que plusieurs opérations de traitement différentes ne se trouvent pas en concurrence lors de l'utilisation des ressources. Chaque tâche de traitement indépendante est affectée à une section spécifique du circuit, et peut donc s'exécuter en toute autonomie sans dépendre aucunement des autres blocs logiques. En conséquence, le volume de traitement effectué peut être accru sans que les performances d'une partie de l'application n'en soient affectées pour autant.

Chaque FPGA est constitué d'un nombre déterminé de ressources prédéfinies comportant des interconnexions programmables qui permettent de mettre en œuvre un circuit numérique reconfigurable et des blocs d'E/S pour que le circuit puisse accéder au monde extérieur [SAL 15].

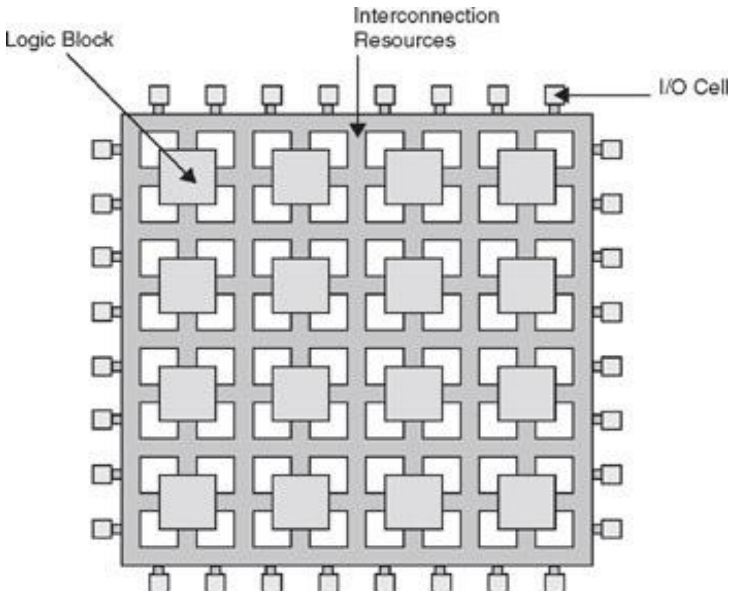


Figure4.1. Structure de base d'un FPGA [SAL 15].

Les blocs logiques configurables (CLB) constituent l'unité logique élémentaire d'un FPGA. Parfois appelés "tranches" ou "cellules logiques", les blocs logiques se composent des bascules (flip-flops), des multiplexeurs, des portes logiques disposées sous forme matricielle et des tables de correspondance (LuT).

Une look-up table ou bien une table de correspondance est une liste d'association de valeurs. Elle se comporte exactement comme une table de vérité avec une seule sortie. Il s'agit donc d'une structure de données stockée en mémoire, employée pour remplacer un calcul par une fonction paramétrée par des opérations plus simples telles que (and, or, xor, …). Le gain de vitesse est significatif, car rechercher une valeur en mémoire est souvent plus rapide qu'effectuer un calcul. Il existe plusieurs tables, dépendantes variables d'entrées. Par exemple, une Lut5 est une mémoire de 2^5 bits, adressée par 5 entrées et une seule sortie. C'est donc une table de vérité 5x1. La figure 4.2 montre le schéma interne d'une Lut6 utilisant deux Lut5 suivi d'un multiplexeur 2 :1 [AZZ 14].

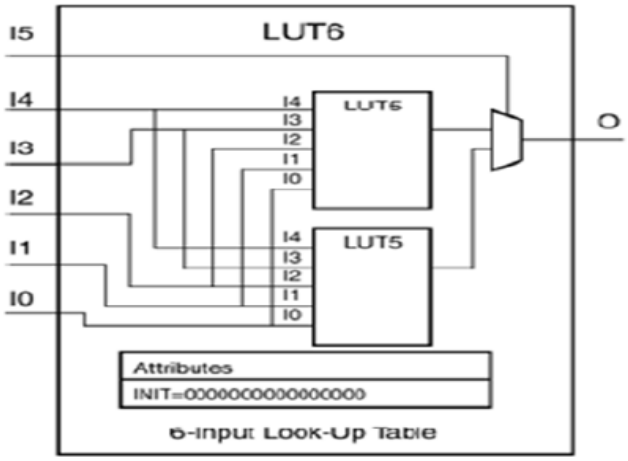


Figure 4.2. LuT6 [AZZ 14].

En plus des composants de base cités, les FPGA comprennent d'autres éléments comme des blocs mémoires BRAMs, des multiplieurs, des blocs DSP et des processeurs embarqués [PAP 12].

IV.3. Description de l'environnement ISE

ISE (Integrated Synthesis Environment) est le logiciel de programmation des produits Xilinx (CPLD, FPGA Spartan et Virtex…). Cet outil permet de créer des projets comportant plusieurs types de fichiers (HDL, schématique, UCF, EDIF, etc.), de compiler, de créer des contraintes d'implémentation avec des contraintes de timings sur les horloges, de déterminer l'emplacement des broches et de créer des fichiers de stimuli.

Le Navigateur de projet ISE offre un environnement de conception et regroupe tous les outils nécessaires à la conception, la simulation et à l'implémentation d'un projet. Il dispose de :

- Un éditeur de textes, de schémas et diagrammes d'états.
- Un compilateur VHDL et Verilog.
- Un simulateur.
- Des outils pour la gestion des contraintes temporelles.
- Des outils pour la synthèse.
- Des outils pour la vérification.
- Des outils pour l'implémentation sur FPGA et CPLD.

IV.4. Étapes d'implémentation sur un circuit FPGA

a. Spécification : la spécification HDL regroupe les trois modes de création d'un circuit (schematic, diagrammes d'états ou HDL). Elle est synthétisée pour générer un fichier appelé NETLIST qui décrit les interconnexions entre les registres.

b. Vérification : la vérification du design est une étape parallèle où le concepteur observe le comportement du code et vérifie s'il se comporte tel qu'il est supposé. Un simulateur simule le circuit par l'utilisation des vecteurs de test. Les vecteurs de test peuvent se présenter sous plusieurs formes, la plus courante est les **TESTBENCHS** écrits dans un langage de description matériel comme le VHDL pour entrer les instructions au simulateur en appliquant les vecteurs de test sur le code pour que le simulateur fournit les sorties du circuit.

c. Implémentation : une fois la vérification est terminée, le circuit est implémenté sur le composant en spécifiant les références exactes de celui-ci à savoir : le circuit utilisé, la fréquence de travail et les autres options spécifiques à chaque composant. Cette étape se termine par un rapport de tous les sous-programmes exécutés (les erreurs, les I/O utilisées et des données qui permettent de savoir si le composant choisi est le mieux adapté pour l'application ciblée) [ATT 13].

IV.5. Langage de description matérielle VHDL

Le VHDL (**V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage**)** est un langage de spécification, vérification et description de composants électroniques. On peut décrire et modéliser autant un système complet qu'une simple porte logique [BOU 15]. Le VHDL est un langage de description matérielle HDL portable et synthétisable structuré de la façon suivante [ATT 13] : *Entity* et *Architecture*. L'entity est la partie déclarative d'un circuit où les différentes entrées / sorties doivent être déclarées. L'architecture décrit le comportement du circuit où les composants, les signaux internes et le programme principal vont être rédigés [AZZ 14].

IV.6. Résultats d'implémentation

IV.6.1. Résultats de Simulation fonctionnelle

Tout d'abord, nous commençons par vérifier le bon fonctionnement de tous les blocs de l'architecture, par des simulations que nous avons réalisé grâce à l'outil ISIM qui est inclus dans l'environnement ISE.

Pour ce faire, nous avons utilisé les mêmes exemples publiés sur le site du NIST, pour vérifier l'exactitude des résultats délivrés par notre architecture.

Nous allons donner dans ce qui suit quelques exemples de simulations des différents blocs constituant notre architecture.

IV.6.1.1. *ShiftRows*

La figure 4.3 montre le résultat de simulation de la première opération *ShiftRows*. Cette dernière est implémentée en utilisant un registre à décalage.

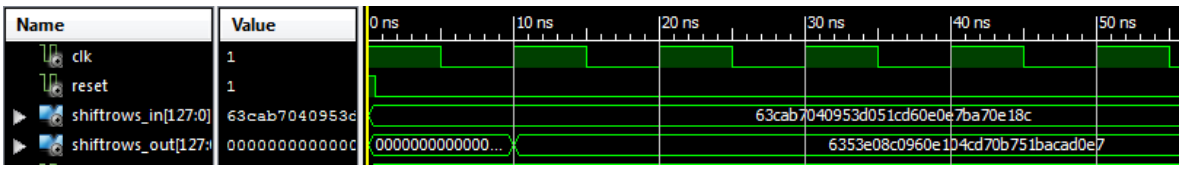


Figure4.3. Simulation de l'*ShiftRows*.

IV.6.1.2. *InvShiftRows*

Nous avons pris le résultat précédent de l'opération *ShiftRows et* vérifier si *l'InvShiftRows* retourne la même valeur d'entrée, comme le montre la figure 4.4.

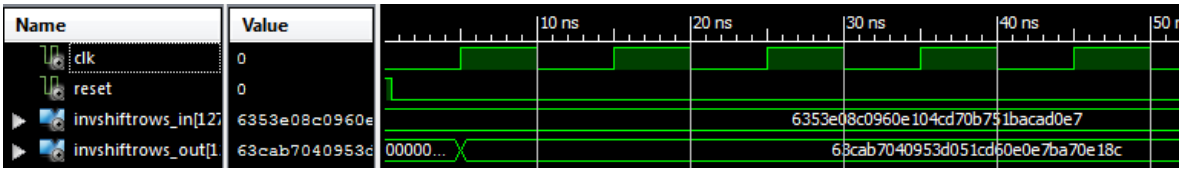


Figure4.4. Simulation de l'*InvShiftRows*.

IV.6.1.3. *SubByte*

La figure 4.5 montre le résultat de simulation de quatre SBox implémentés en parallèle.

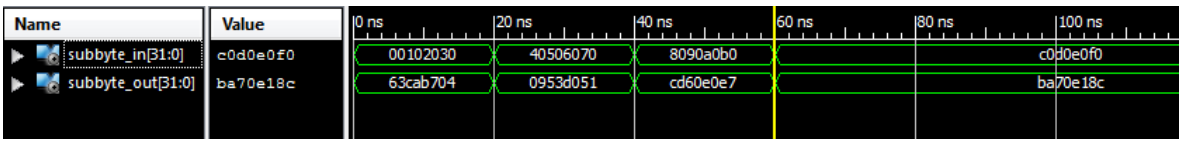


Figure4.5. Simulation de *SubByte*.

IV.6.1.4. InvSubByte

Pour la simulation de *l'InvSubByte*, l'entrée du vecteur de test est le résultat obtenu par la transformation *SubByte*. La figure 4.6 montre les résultats obtenus.

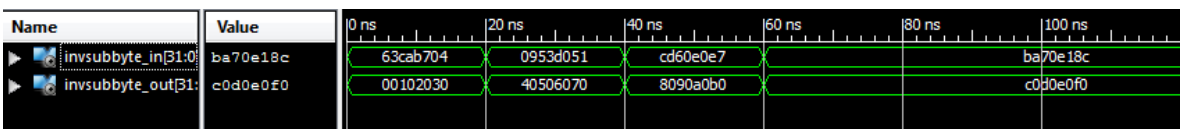


Figure 4.6. Simulation de l'*InvSubByte*.

IV.6.1.5. *MixColomns/AddRoundKey*

Puisque la transformation AddRoundKey n'est rien d'autre que de simples XOR bit par bit entre la donnée et la clé et comme nous avons optimisé l'opération Mixcolumn en n'utilisant que des opérations XOR, nous avons alors pensé à rassembler les deux opérations MixColumn et AddRoundKey dans une seule fonction logique.

Cette transformation prend en entrée 4 octets E_0, E_1, E_2, E_3 et un vecteur de 32 bits avec K représentant la clé et les résultats sont sauvegardés dans 4 octets S_0, S_1, S_2, S_3. La figure 4.7 montre le résultat de simulation de cette opération.

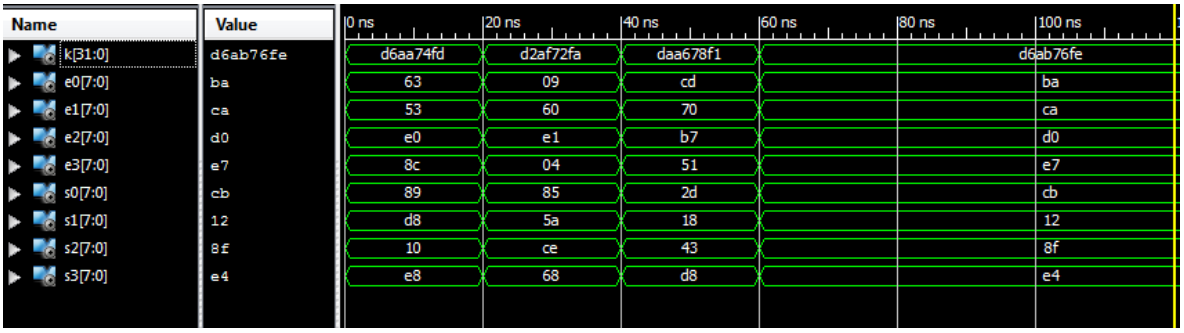


Figure 4.7. Simulation de *MixColomn/AddRoundKey*.

IV.6.1.6. *InvMixColomn/AddRoundKey*

La simulation de l'opération InvMixColomn/AddRoundKey est illustrée par la figure 4.8.

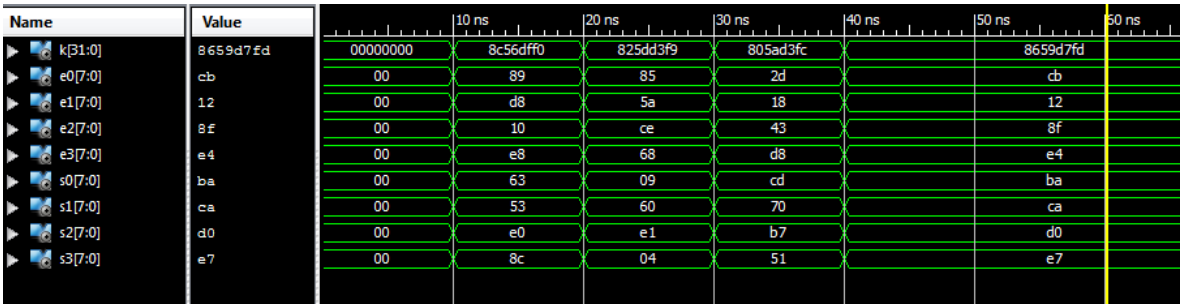


Figure 4.8. Simulation de *l'InvMixColomn/AddRoundKey*.

IV.6.1.7. *KeyExpansion*

Le processus de génération des sous clés est un module qui a deux entrées cle_in et RC. Puisque les valeurs de RC sont sauvegardées dans une ROM, nous avons relié l'entrée de la ROM avec un compteur qui compte de 0 à 9. La figure 4.9 montre le résultat de simulation de la ROM.

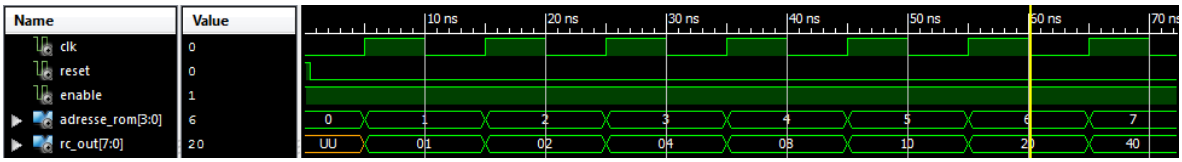


Figure 4.9. Simulation du composant ROM.

La figure 4.10 montre le résultat de simulation de génération des sous clés. Bien sûr, les résultats obtenus ont été vérifiés et comparés par les résultats du NIST.

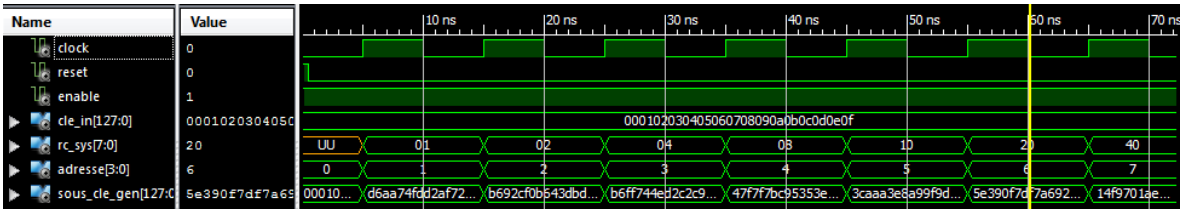


Figure 4.10. Simulation de *KeyExpansion*.

IV.6.1.8. Chiffrement

Après la vérification du fonctionnement de chaque composant individuellement, nous simulons le crypto système en entier. La figure 4.11 représente le résultat de la simulation du processus de chiffrement.

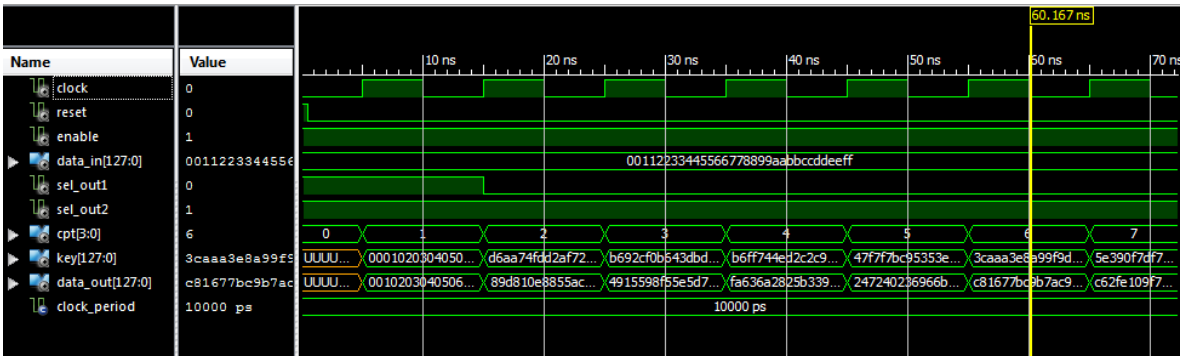


Figure 4.11. Résultat de simulation de chiffrement.

IV.6.1.9. Déchiffrement

Cette transformation est l'opération inverse du chiffrement et la même architecture que celle du chiffrement a été appliquée. La figure 4.12 montre la simulation fonctionnelle du déchiffrement.

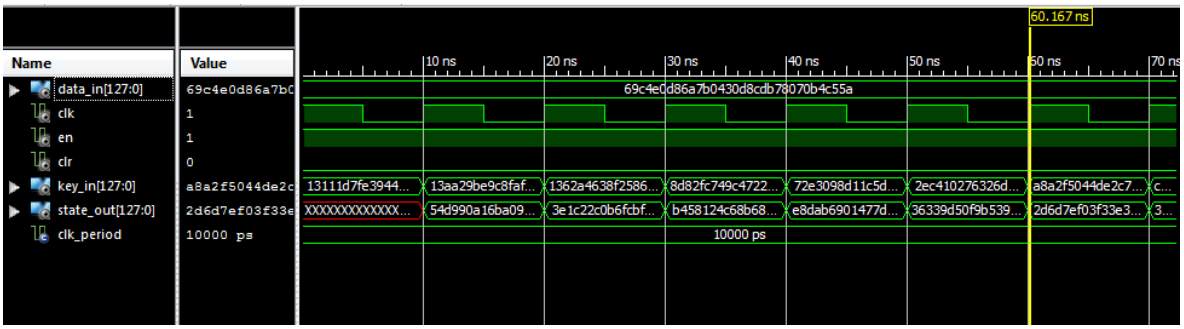


Figure 4.12. Résultat de simulation de déchiffrement

IV.6.2. Synthèse et implémentation

Après la vérification du fonctionnement des différents blocs développés, nous présentons l'ensemble des rapports de synthèse et d'implémentation de notre IP en précisant le temps d'exécution et la surface occupée pour chaque transformation.

Les tableaux 4.1 et 4.2 montrent les résultats d'implémentation des transformations *ShiftRows* et *InvShiftRows.*

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	728,400	0%
Number of Slice LUTs	0	364,200	0%
Number of occupied Slices	0	91,050	0%
Number of LUT Flip Flop pairs used	0		
Number of bonded IOBs	258	600	43%
IOB Flip Flops	128		

Tableau 4.1. Résultats d'implémentation *de la transformation ShiftRows*.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	728,400	0%
Number of Slice LUTs	0	364,200	0%
Number of occupied Slices	0	91,050	0%
Number of LUT Flip Flop pairs used	0		
Number of bonded IOBs	258	600	43%
IOB Flip Flops	128		

Tableau 4.2. Résultats d'implémentation *de l'InvShiftRows*.

Ces résultats montrent que l'implémentation de *ShiftRows* et de son inverse ne nécessitent aucune LUT.

Les résultats de synthèse de la *SubByte* sont résumés dans le tableau 4.3.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	728,400	0%
Number of Slice LUTs	40	364,200	1%
Number used as logic	40	364,200	1%
Number using O6 output only	40		

Tableau 4.3. Résultats de synthèse de *SubByte*.

Les résultats de synthèse de l'*InvSubByte* sont donnés par le tableau 4.4.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	728,400	0%
Number of Slice LUTs	40	364,200	1%
Number used as logic	40	364,200	1%
Number using O6 output only	40		

Tableau 4.4. Résultats de synthèse de l'*InvSubByte*.

Pour la *SubByte* et son inverse, les ressources occupées sont exactement les mêmes, car elles sont deux tables de même taille.

L'implémentation de *MixColomn*/*AddRoundKey* et son inverse utilisant des portes XOR sont représentées par les tableaux 4.5 et 4.6.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	728,400	0%
Number of Slice LUTs	39	364,200	1%
Number used as logic	39	364,200	1%
Number using O6 output only	34		
Number using O5 output only	0		
Number using O5 and O6	5		
Number used as ROM	0		
Number used as Memory	0	111,000	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	26	91,050	1%

Tableau 4.5. Résultats de synthèse de *MixColomn*/*AddRoundKey* avec des XOR.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	728,400	0%
Number of Slice LUTs	74	364,200	1%
Number used as logic	74	364,200	1%
Number using O6 output only	66		
Number using O5 output only	0		
Number using O5 and O6	8		
Number used as ROM	0		
Number used as Memory	0	111,000	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	32	91,050	1%

Tableau 4.6. Résultats de synthèse d'*InvMixColomn*/*AddRoundKey* avec des XOR.

Les résultats d'implémentation des deux transformations précédentes avec l'utilisation des luts sont représentés dans les tableaux 4.7 et 4.8.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	728,400	0%
Number of Slice LUTs	38	364,200	1%
Number used as logic	38	364,200	1%
Number using O6 output only	32		
Number using O5 output only	0		
Number using O5 and O6	6		
Number used as ROM	0		
Number used as Memory	0	111,000	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	21	91,050	1%

Tableau 4.7. Résultats de synthèse de *MixColomn*/*AddRoundKey* avec des LuTs.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	728,400	0%
Number of Slice LUTs	76	364,200	1%
Number used as logic	76	364,200	1%
Number using O6 output only	72		
Number using O5 output only	0		
Number using O5 and O6	4		
Number used as ROM	0		
Number used as Memory	0	111,000	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	40	91,050	1%

Tableau 4.8. Résultats de synthèse d'*InvMixColomn*/*AddRoundKey* avec des LuTs.

Après avoir vu les résultats d'implémentation de l'étape *MixColumns* et de son inverse, nous confirmons que l'étape *InvMixComumns* est plus complexe que la *MixColumns* et plus coûteuse vue la différence du volume des ressources occupées dans les deux architectures.

Nous pouvons remarquer aussi que l'implémentation de la *MixColumns*, au niveau LuT nécessite moins de ressources comparée à l'implémentation par l'opérateur XOR qui vérifie le contraire pour son inverse.

Les tableaux 4.9 et 4.10 montrent les ressources occupées pour la partie chiffrement et celle de déchiffrement.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	258	728,400	1%
Number used as Flip Flops	258		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	1,034	364,200	1%
Number used as logic	1,034	364,200	1%
Number using O6 output only	922		
Number using O5 output only	0		
Number using O5 and O6	112		
Number used as ROM	0		
Number used as Memory	0	111,000	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	415	91,050	1%

Tableau 4.9. Résultats d'implémentation du chiffrement AES.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	256	728,400	1%
Number used as Flip Flops	256		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	1,413	364,200	1%
Number used as logic	1,413	364,200	1%
Number using O6 output only	1,253		
Number using O5 output only	0		
Number using O5 and O6	160		
Number used as ROM	0		
Number used as Memory	0	111,000	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	665	91,050	1%

Tableau 4.10. Résultats d'implémentation du déchiffrement AES.

Le résultat de synthèse de l'architecture *KeyExpansion* est représenté par le tableau 4.11.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	142	728,400	1%
Number used as Flip Flops	142		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	327	364,200	1%
Number used as logic	327	364,200	1%
Number using O6 output only	313		
Number using O5 output only	0		
Number using O5 and O6	14		
Number used as ROM	0		
Number used as Memory	0	111,000	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	130	91,050	1%

Tableau 4.11. Résultats d'implémentation de la partie *KeyExpansion*.

Puisque l'outil de synthèse ne fournit qu'une estimation du temps d'exécution, nous avons utilisé l'outil Analyze *Post-Place & Route Static Timing* afin de trouver le chemin critique. Ceci nous permet de relever les délais des différents blocs constituant notre architecture. Dans le tableau 4.12, nous montrons les délais des blocs utilisés dans le chiffrement pour les deux approches par le portes XOR et par les Luts. Le tableau 4.13 montre la même chose pour le déchiffrement.

	Temps (ns)	Logique (ns)	Routage (ns)
SubByte	1.492	0.423	1.069
MixColomn/AddRoundKey (***XOR***)	1.482	0.282	1.200
MixColomn/AddRoundKey(**LuTs**)	1.093	0.266	0.827
Chiffrement	2.948	0.628	2.320

Tableau 4.12. Résultats de l'analyse temporelle des différents composants de chiffrement.

	Temps (ns)	Logique (ns)	Routage (ns)
InvSubByte	1.492	0.423	1.069
InvMixColomn/*AddRoundKey*(**XOR**)	1.404	0.278	1.126
InvMixColomn/*AddRoundKey*(**Luts**)	1.349	0.370	0.979
Déchiffrement	3.658	0.561	3.097

Tableau4.13. Résultats de l'analyse temporelle des différents composants de déchiffrement.

Le délai du processus keyExpansion est donné dans le tableau 4.14.

	Temps (ns)	Logique (ns)	Routage (ns)
keyExpansion	2.369	0.468	1.901

Tableau4.14. Résultats de l'analyse temporelle de *keyExpansion*.

Le tableau ci-dessous montre les ressources du circuit programmable VIRTEX-7 XC7V585T-3-FFG1157 occupées lors de l'implémentation de notre crypto-système :

Ressources de VIRTEX-7	Utilisés	Disponibles	Utilisation
Nombre des Registres dans le « SLICE »	408	728400	1%
Nombres de « LUTs » dans le « SLICE »	2662	364200	1%
Nombre de « SLICE » occupés	968	91050	1%
Nombre d'entrées/sorties	388	600	64%
Number of Block RAM/FIFO	2	795	1%

Tableau4.15. Résultats de l'implémentation du crypto-système AES.

La figure 4.11 représente la surface occupée par l'architecutre l'AES lors de son implémentation avec l'outil ***FPGA EDITOR*** de ISE de XILINX, l'espace bleu décrit les ressources et les interconnexions utilisés :

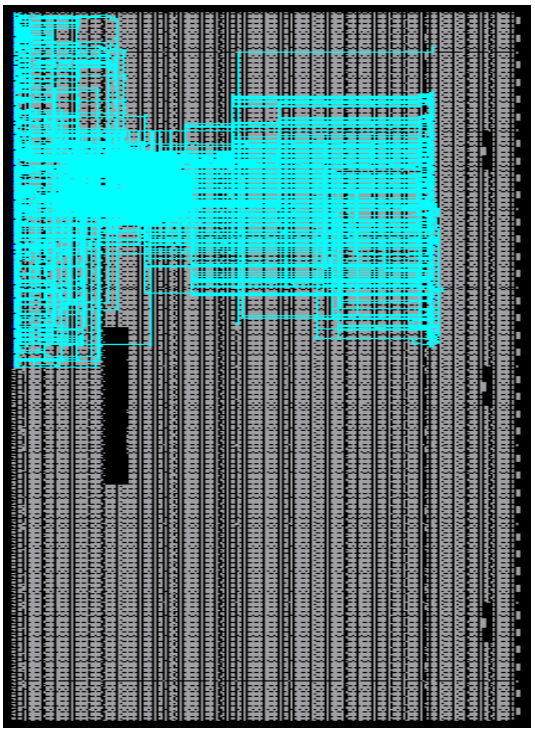


Figure 4.13. Surface occupée par le crypto système AES.

IV.6.2. Discussion des résultats

Notre travail est consacré aux applications forts débits qui nécessitent un chiffrement d'une grande masse d'information. Le premier objectif est de diminuer le temps de chiffrement et de déchiffrement, c'est pour cela qu'on est passé vers la programmation bas niveau en utilisant les LuTs. En effet, nous avons implémenté la *SubByte* en LuT. Ceci nous a permis d'avoir un temps d'exécution satisfaisant. De même, pour la MixColumn et l'InvMixColumn en utilisant les LuTs, nous remarquons d'après les tableaux 4.10 et 4.11 que le temps d'exécution est bien réduit.

Le deuxième objectif est de réduire l'écart de temps d'exécution entre le chiffrement et le déchiffrement, qui se produit à cause de la complexité de l'*InvMixColumns* par rapport au MixColumns. Pour ce faire, nous avons utilisé une autre technique d'optimisation. Celle-ci a été détaillée dans le chapitre 3. Par le biais de cette dernière, nous avons pu réduire cet écart. Comme nous pouvons le constater sur les deux tableaux 4.10 et 4.11.

IV.7. Conclusion

Nous avons introduit ce chapitre par une brève description des différents outils nécessaires pour le développement de notre crypto-système en présentant les différentes étapes d'implémentation sur un circuit programmable, et en décrivant le langage de description matérielle utilisé.

Par la suite, nous avons exposé nos résultats de simulations de chaque bloc fonctionnel pour attester leurs bons fonctionnements, en les validant par un exemple publié par le NIST.

A la fin, nous avons implémenté notre crypto-système sur un circuit Virtex-7, en précisant le temps d'exécution et les ressources occupées.

Conclusion & perspectives

Conclusion et Perspectives

Le thème de notre projet étant la conception d'une architecture matérielle pour l'AES et implémentation sur circuit FPGA, d'ou nous avons adopté une double démarche pour le traiter. Une démarche théorique qui consiste à travers une série de synthèses d'asseoir une approche globale de la cryptologie en général, et du protocole cryptographique AES en particulier. Ensuite vient la démarche pratique : dans cette dernière, nous avons pu réaliser un IP pour le crypto-système AES et de l'implémenter sur un circuit FPGA.

Tout au long de la première démarche, nous avons étudié les opérations mathématiques requises par les rounds de l'AES, à savoir *SubBytes*, *ShiftRows*, *MixColumn* et *AddRoundKey* puis nous avons examiné les architectures possibles de l'AES, à savoir Série/Série, Parallèle/Série où Parallèle/Pipeline pour choisir celle qui offre les meilleures performances dans le but de l'implémenter sur circuit FPGA. Le choix s'est fixé sur l'architecture Parallèle/Série qui donne un bon compromis entre le temps d'exécution et les ressources matérielles occupées.

Durant la deuxième démarche, la conception de notre IP a été réalisée en programmant au niveau le plus bas des circuits FPGA à savoir sur les « *Look up Tables* » ou LuT, qui nous a permis d'augmenter les performances de chiffrement et de déchiffrement en diminuant les délais de routage. Cette approche nous a permis de rendre le temps d'une lecture mémoire équivalent au temps de transition d'une LuT par rapport à la transformation *SubByte*. Dans un deuxième temps nous avons utilisé une technique d'optimisation par rapport à la transformation *InvMixColumns* qui est l'opération la plus complexe. Le fruit de cette optimisation était de réduire l'écart entre le processus de chiffrement et de déchiffrement. Nous avons fini par intégrer l'étape *AddRoundKey* avec la *MixColumns* et son inverse en profitant d'avantage des entrées des LuTs non utilisées.

Comme perspective à ce travail, une architecture Parallèle/Pipeline permettra d'augmenter d'avantage le débit. Ainsi que l'utilisation des modes de chiffrement va aider à augmenter les performances du système sans pénaliser son efficacité.

Bibliographie

Bibliographie

[ABD 09] I. Abd-ElGhafar, A. Rohiem, A. Diaa, F. Mohammed, Generation of AES Key Dependent S-Boxes using RC4 Algorithm,International Conference on *AEROSPACE SCIENCES & AVIATION TECHNOLOGY*, Military Technical College, KobryElkobbah, Cairo, Egypt, 26 mai 2009.

[ANA 15] M. Anane, N. Anane, Efficient Implementation of AES S-box in LUT-6 FPGAs, International Conference on Electrical Engineering, IGEE, Boumerdes, Algerie 13 - 15 Decembre 2015.

[ATT 13] B. M. Attar, A. Amini, Contribution à La Conception d'un Système d'authentification Cryptographique Embarqué sur FPGA, Mémoire de Master en INFOTRONIQUE, Université M'Hamed BOUGARA Boumerdes, Algérie, 2013.

[AZZ 14] O. Azzouzi, Système embarqué flexible pour un chiffrement hybride symétrique / asymétrique, Mémoire de Magistère en Informatique, Ecole Nationale Supérieure, Oued Smar, Algérie, 2014.

[BAB 12] M.Z. Baba-Ahmed, F. Z. Benmansour, M.Anane, Conception d'un crypto système pour les transmissions de données chiffrées, International Conference on Software Engineering and New Technologies, IGEE, Boumerdes, Algérie, Decembre 2012.

[BOU 15] H. Boumeridja**,** Introduction aux FPGAS Concepts de Base, Architecture et applications, support de cours, Université De Boumerdes, Algérie, Mai 2015.

[CAN 06] A. Canteaut, La cryptologie moderne, INRIA Projet CODES BP 105 78153 Le Chesnay Cedex, 2006.

[DAE 01] J. Daemen, V. Rijmen, The Design of Rijndael, AES : The Advanced Encryption Standard, ISBN 3540425802, 2001.
URL https://autonome-antifa.org/IMG/pdf/Rijndael.pdf

[FOU 03] P.A. Fouque, Cryptographie appliquée, Techniques de l'Ingénieur, traité Sécurité des systèmes d'information, 2003.

[FIG 00] C. Figlarek, l'utilisation de la cryptographie dans les échanges de données médico-sociale, mémoire de l'école national de la santé publique, Rennes, France, 2000.

[GUR 06] F. K. Gürkaynak, ALS System Design : Side Channel Attack Secure Cryptographic Accelerators, Cryptographic Accelerators Istanbul, Turquie, 20 décembre 2006.

[MES 06] D.G.Mesquita, Architectures reconfigurables et cryptographie : une analyse de robustesse et contremesures face aux attaques par canaux cachés , thèse de Doctorat de l'université de Montpellier II, France, 2006.

[PAA 10] C. Paar, J. Pelzl, Understanding Cryptography, Springer-Verlag Berlin Heidelberg, ISBN 978-3-642-04100-6, 2010.

[RAM 09] E. Ramaraj, S. Karthikeyan and M. Hemalatha, A Design of Security Protocol using Hybrid Encryption Technique AES- Rijndael and RSA, International Journal of the Computer, the Internet and Management Vol. 17, N°1, janvier-avril, 2009 pp 78-86.

[SAL 15] A. Salhi, K. Bakiri, FPGA-Based Cryptosystem, Mémoire de Master en Electricité et Electronique, Université M'Hamed BOUGARA, Boumerdes, Algérie, 2015.

[SAM 02] S. Samanta B.E, FPGA Implementation of AES Encryption and Decryption, Electronics & Communication Engg, Sardar Vallabhbhai National Institute of Technology, surat, Inde, 2002.

[SCH 03] J. Schiltz, Les modes opératoires de la cryptographie symétrique, livre électronique, 2003.
URL https://orbilu.uni.lu/bitstream/10993/8992/1/2003_1%20Cryptographie%20symétrique.pdf

[VAR 06] S. VARRETTE, Introduction à la cryptologie, Université du Luxembourg – Laboratoire LACS, Luxembourg CNRS/INPG/INRIA/UJF-Laboratoire LIG-IMAG, 2006.
URL http://www.apprendre-en-ligne.net/crypto/bibliotheque/PDF/brisson.pdf

Webographie

[PAP 12] URL http://www.ni.com/white-paper/6983/fr/ , NATIONAL INSTRUMENTS, dernière visite le ; 18/06/2016.

Annexes

Annexe A

Arithmétique de l'AES « Corps de Galois (2^8) »

Dans l'AES, le corps de Galois contient 256 éléments et est noté GF (2^8). Ce corps a été choisi parce que chacun de ces éléments peut être représenté par un octet. L'arithmétique de Galois Field est utilisée dans la plupart des transformations de l'AES. Nous verrons ci-après que les éléments de GF (2^8) peuvent être représentés comme des polynômes, et que le calcul est réalisé en effectuant un certain type d'arithmétique polynomiale. Les polynômes ont un degré maximum de 7, de sorte qu'il y a 8 coefficients au total pour chaque élément donc chaque élément A ∈ GF (2^8) est représenté comme suit :

$$A(x) = a_7x^7 + \cdots + a_1x + a_0,\ a_i \in GF(2) = \{0,1\}.$$

1. Addition et Soustraction dans GF (2^8)

La transformation *AddRoundKey* utilise l'addition. Elle est simplement obtenue en effectuant une addition polynômiale dans Galois Field (2) : on additionne simplement les coefficients avec des puissances égales de x. l'addition dans Galois Field (2) correspond à un simple XOR bit à bit.

Par exemple :

$$\begin{array}{l} A(x) = x^7 + x^6 + x^4 + \quad 1 \\ B(x) = \quad\quad x^4 + x^2 + 1 \\ \hline C(x) = x^7 + x^6 \quad + x^2 \end{array}$$

La soustraction est la même que l'addition.

2. Multiplication dans GF (2^8)

La multiplication dans GF (2^8) est l'opération de base de la transformation *Mixcolumns*. Dans une première étape, deux éléments (représentés par leurs polynômes) de GF (2^8) sont multipliés en utilisant la règle de multiplication polynomiale. En général, le produit polynomial avec un degré supérieur à 7 doit être réduit. Sa réduction est le reste de sa division polynomiale par un polynôme irréductible,

Pour l'AES, le polynôme irréductible utilisé est :

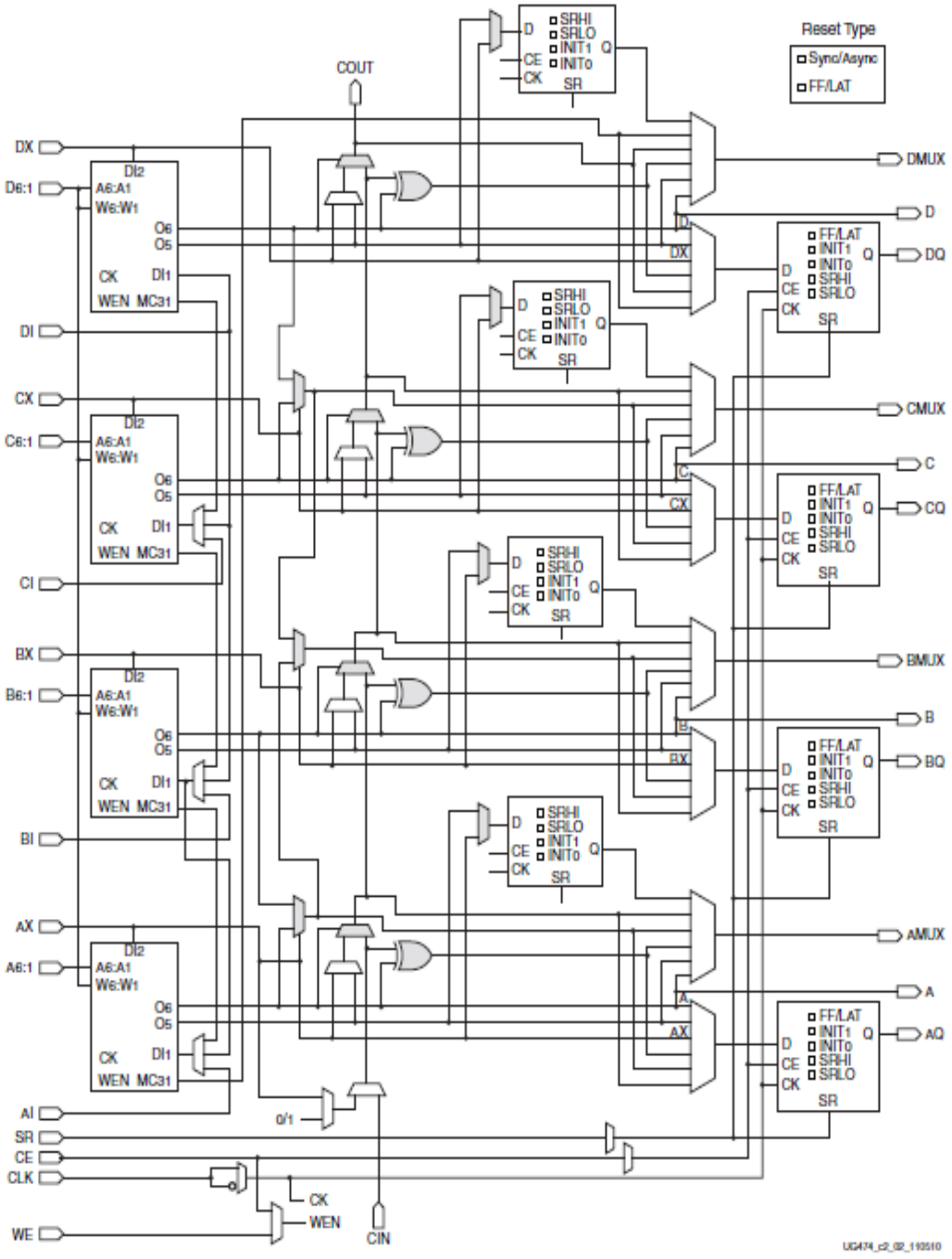
$$P(x) = x^8 + x^4 + x^3 + x + 1$$

Soit A (x), B (x) ∈ GF (2^8) :

$$C(x) \equiv A(x) \cdot B(x) \bmod P(x).$$

Annexe B

L'architecture d'un slice de Virtex7 est illustrée dans la figure suivante :



Architecture d'un slice de Virtex 7

Annexe C

Cette annexe représente un exemple publié par NIST, que nous avons utilisé pour vérifier le bon fonctionnement de notre architecture.

PLAINTEXT :00112233445566778899aabbccddeeff

KEY : 000102030405060708090a0b0c0d0e0f

CIPHER (ENCRYPT):

Round [0].input 00112233445566778899aabbccddeeff

Round [0].k_sch 000102030405060708090a0b0c0d0e0f

Round [1].start 00102030405060708090a0b0c0d0e0f0

Round [1].s_box 63cab7040953d051cd60e0e7ba70e18c

Round [1].s_row 6353e08c0960e104cd70b751bacad0e7

Round [1].m_col 5f72641557f5bc92f7be3b291db9f91a

Round [1].k_sch d6aa74fdd2af72fadaa678f1d6ab76fe

Round [2].start 89d810e8855ace682d1843d8cb128fe4

Round [2].s_box a761ca9b97be8b45d8ad1a611fc97369

Round [2].s_row a7be1a6997ad739bd8c9ca451f618b61

Round [2].m_col ff87968431d86a51645151fa773ad009

Round [2].k_sch b692cf0b643dbdf1be9bc5006830b3fe

Round [3].start 4915598f55e5d7a0daca94fa1f0a63f7

Round [3].s_box 3b59cb73fcd90ee05774222dc067fb68

Round [3].s_row 3bd92268fc74fb735767cbe0c0590e2d

Round [3].m_col 4c9c1e66f771f0762c3f868e534df256

Round [3].k_sch b6ff744ed2c2c9bf6c590cbf0469bf41

Round [4].start fa636a2825b339c940668a3157244d17

Round [4].s_box 2dfb02343f6d12dd09337ec75b36e3f0

Round [4].s_row 2d6d7ef03f33e334093602dd5bfb12c7

Round [4].m_col 6385b79ffc538df997be478e7547d691

Round [4].k_sch 47f7f7bc95353e03f96c32bcfd058dfd

Round [5].start 247240236966b3fa6ed2753288425b6c

Round [5].s_box 36400926f9336d2d9fb59d23c42c3950

Round [5].s_row 36339d50f9b539269f2c092dc4406d23

Round [5].m_col f4bcd45432e554d075f1d6c51dd03b3c

Round [5].k_sch 3caaa3e8a99f9deb50f3af57adf622aa

Round [6].start c81677bc9b7ac93b25027992b0261996

Round [6].s_box e847f56514dadde23f77b64fe7f7d490

Round [6].s_row e8dab6901477d4653ff7f5e2e747dd4f

Round [6].m_col 9816ee7400f87f556b2c049c8e5ad036

Round [6].k_sch 5e390f7df7a69296a7553dc10aa31f6b

Round [7].start c62fe109f75eedc3cc79395d84f9cf5d

Round [7].s_box b415f8016858552e4bb6124c5f998a4c

Round [7].s_row b458124c68b68a014b99f82e5f15554c

Round [7].m_col c57e1c159a9bd286f05f4be098c63439

Round [7].k_sch 14f9701ae35fe28c440adf4d4ea9c026

Round [8].start d1876c0f79c4300ab45594add66ff41f

Round [8].s_box 3e175076b61c04678dfc2295f6a8bfc0

Round [8].s_row 3e1c22c0b6fcbf768da85067f6170495

Round [8].m_col baa03de7a1f9b56ed5512cba5f414d23

Round [8].k_sch 47438735a41c65b9e016baf4aebf7ad2

Round [9].start fde3bad205e5d0d73547964ef1fe37f1

Round [9].s_box 5411f4b56bd9700e96a0902fa1bb9aa1

Round [9].s_row 54d990a16ba09ab596bbf40ea111702f

Round [9].m_col e9f74eec023020f61bf2ccf2353c21c7

Round [9].k_sch 549932d1f08557681093ed9cbe2c974e

Round [10].start bd6e7c3df2b5779e0b61216e8b10b689

Round [10].s_box 7a9f102789d5f50b2beffd9f3dca4ea7

Round [10].s_row 7ad5fda789ef4e272bca100b3d9ff59f

Round [10].k_sch 13111d7fe3944a17f307a78b4d2b30c5

Round [10].output 69c4e0d86a7b0430d8cdb78070b4c55a

INVERSE CIPHER (DECRYPT)

Round [0].iinput 69c4e0d86a7b0430d8cdb78070b4c55a

Round [0].ik_sch 13111d7fe3944a17f307a78b4d2b30c5

Round [1].istart 7ad5fda789ef4e272bca100b3d9ff59f

Round [1].is_row 7a9f102789d5f50b2beffd9f3dca4ea7

Round [1].is_box bd6e7c3df2b5779e0b61216e8b10b689

Round [1].ik_sch 549932d1f08557681093ed9cbe2c974e

Round [1].ik_add e9f74eec023020f61bf2ccf2353c21c7

Round [2].istart 54d990a16ba09ab596bbf40ea111702f

Round [2].is_row 5411f4b56bd9700e96a0902fa1bb9aa1

Round [2].is_box fde3bad205e5d0d73547964ef1fe37f1

Round [2].ik_sch 47438735a41c65b9e016baf4aebf7ad2

Round [2].ik_add baa03de7a1f9b56ed5512cba5f414d23

Round [3].istart 3e1c22c0b6fcbf768da85067f6170495

Round [3].is_row 3e175076b61c04678dfc2295f6a8bfc0

Round [3].is_box d1876c0f79c4300ab45594add66ff41f

Round [3].ik_sch 14f9701ae35fe28c440adf4d4ea9c026

Round [3].ik_add c57e1c159a9bd286f05f4be098c63439

Round [4].istart b458124c68b68a014b99f82e5f15554c

Round [4].is_row b415f8016858552e4bb6124c5f998a4c

Round [4].is_box c62fe109f75eedc3cc79395d84f9cf5d

Round [4].ik_sch 5e390f7df7a69296a7553dc10aa31f6b

Round [4].ik_add 9816ee7400f87f556b2c049c8e5ad036

Round [5].istart e8dab6901477d4653ff7f5e2e747dd4f

Round [5].is_row e847f56514dadde23f77b64fe7f7d490

Round [5].is_box c81677bc9b7ac93b25027992b0261996

Round [5].ik_sch 3caaa3e8a99f9deb50f3af57adf622aa

Round [5].ik_add f4bcd45432e554d075f1d6c51dd03b3c

Round [6].istart 36339d50f9b539269f2c092dc4406d23

Round [6].is_row 36400926f9336d2d9fb59d23c42c3950

Round [6].is_box 247240236966b3fa6ed2753288425b6c

Round [6].ik_sch 47f7f7bc95353e03f96c32bcfd058dfd

Round [6].ik_add 6385b79ffc538df997be478e7547d691

Round [7].istart 2d6d7ef03f33e334093602dd5bfb12c7

Round [7].is_row 2dfb02343f6d12dd09337ec75b36e3f0

Round [7].is_box fa636a2825b339c940668a3157244d17

Round [7].ik_sch b6ff744ed2c2c9bf6c590cbf0469bf41

Round [7].ik_add 4c9c1e66f771f0762c3f868e534df256

Round [8].istart 3bd92268fc74fb735767cbe0c0590e2d

Round [8].is_row 3b59cb73fcd90ee05774222dc067fb68

Round [8].is_box 4915598f55e5d7a0daca94fa1f0a63f7

Round [8].ik_sch b692cf0b643dbdf1be9bc5006830b3fe

Round [8].ik_add ff87968431d86a51645151fa773ad009

Round [9].istart a7be1a6997ad739bd8c9ca451f618b61

Round [9].is_row a761ca9b97be8b45d8ad1a611fc97369

Round [9].is_box 89d810e8855ace682d1843d8cb128fe4

Round [9].ik_sch d6aa74fdd2af72fadaa678f1d6ab76fe

Round [9].ik_add 5f72641557f5bc92f7be3b291db9f91a

Round [10].istart 6353e08c0960e104cd70b751bacad0e7

Round [10].is_row 63cab7040953d051cd60e0e7ba70e18c

Round [10].is_box 00102030405060708090a0b0c0d0e0f0

Round [10].ik_sch 000102030405060708090a0b0c0d0e0f

Round [10].ioutput 00112233445566778899aabbccddeeff