République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Sâad DAHLAB de Blida
Faculté des sciences
Département d'informatique

MEMOIRE DE FIN D'ETUDE

En vue de l'obtention du Diplôme de Master en Informatique

OPTION : Génie des Systèmes Informatique *(GSI)*

Thème :

Contribution à la Conception d'un Crypto-Système Hybride RSA-AES Embarqué sur un circuit programmable FPGA

Soutenu le : 10 Septembre 2013.

Présenté par :

M^lles : **BENACHOUR Fida** et **GOUIZI Khadidja**

MA-004-159-1

Devant le jury composé de :

Dr : **BENNOUAR Djamal, Maître de Conférences A, USDB** Président

Mr : **SIDOUMOU Mohamed Réda, Maître Assistant A, USDB** Examinateur

Mr : **BAOUYA Abd Elhakim, Maître Assistante B, USDB** Examinateur

Dr : **ANANE Mohamed, Maître de Conférences B, ESI** Promoteur

Organisme d'accueil :

CDTA – Centre de Développement des Technologies Avancées
Baba Hassen, Alger

Remerciements

Tout d'abord, nous tenons à rendre grâce à DIEU tout puissant pour nous avoir donné le courage et la détermination nécessaire pour finaliser ce travail.

Nous tenons à remercier avec gratitude notre promoteur Docteur ***Mohamed ANANE*** *pour l'honneur qu'il nous a fait d'avoir accepté de nous encadrer et qui a endossé son rôle de la meilleure façon qui soit. Nous retiendrons sa disponibilité, son aide indéfectible, ses conseils avisés et ses idées riches ainsi que sa sympathie et ses encouragements. Nous voulons le remercier aussi de nous avoir fait bénéficier de son savoir et de son expérience tout au long de la période de notre stage. Que vous trouvez ici l'expression de notre plus profond respect et nos sincères reconnaissances.*

Toute notre gratitude est exprimée aussi à notre chef d'option Docteur ***Djamel BENNOUAR*** *pour l'honneur qu'il nous a fait d'avoir accepté l'accès dans son option «* ***GSI*** *» et qui a contribué activement à notre formation pendant notre cursus universitaire. Que vous trouvez ici l'expression de tous nos remerciements et reconnaissances.*

Nos sincères remerciements et notre profonde reconnaissance sont adressées à notre encadreur Mme ***N.ANANE*** *pour le temps qu'il a bien voulu nous consacré.*

Nous remercions aussi Mme ***Nacera CHERID*** *directrice des études à l'ESI pour l'honneur qu'il nous a fait d'avoir facilité l'accès dans établissement et la mise à disposition des ressources nécessaires.*

Aussi, nous exprimons nos sincères remerciements et notre profonde reconnaissance à Monsieur ***Sami HEBIB*** *pour son aide très précieuse et pour le temps qu'il a bien voulu nous consacré.*

Nos sincères remerciements sont adressés aux membres du jury :

Dr : ***(Djamel BENNOUAR)*** *qui nous a fait l'honneur de présider ce jury*

Mr : ***(Mohamed Réda SIDOUMOU)*** *et Mr :* ***(A.H. BAOUYA)*** *pour le temps que vous nous avez accordé pour l'examinassions de ce travail afin de le bien valoriser.*

Un grand merci à tout le corps enseignant du département d'informatique de l'université Saâd DAHLAB de Blida qui nous a guidés durant toutes nos années d'étude.

Que ceux qui se sentent oubliés mais par ses aides : par une idée, une prière, une sourie trouvent ici notre profonde gratitude et nos chaleureux remerciements pour leur concours dans l'accomplissement de ce travail.

Dédicaces

Je dédie ce modeste travail :

A Mes Très Chers Parents

Tous les mots du monde ne sauraient exprimer l'immense amour que je vous porte, ni la profonde gratitude que je vous témoigne pour tous les efforts et les sacrifices que vous n'avez jamais cessé de consentir pour mon instruction et mon bien-être.

C'est à travers vos encouragements que j'ai opté pour cette noble profession, et c'est à travers vos critiques que je me suis réalisée.

J'espère avoir répondu aux espoirs que vous avez fondés en moi.

Que Dieu tout puissant vous garde et vous procure santé, bonheur et longue vie pour que vous demeuriez le flambeau illuminant le chemin de vos enfants

À mes Grand Parents

Puisse Dieu vous protéger du mal, vous procurer une longe vie pleine de bonheur

A mes très chers frères

. Puisse Dieu tout puissant jouir votre vie, vous combler d avantage, t'apporter bonheur, et t'aider à réaliser tous tes vux

A Toute ma grande famille :oncles, tantes, cousins et cousines.

A toutes mes amies

Merci pour les bons moments qu'on a passé ensemble, de votre soutien et de votre serviabilité. Sousou, Yasmine, Nihad ,Mohammed, Chouaib

A ma chère binôme Khadidja ainsi, que toute sa famille. Que Dieu vous la garde.

A tous mes collègues de la promotion 2012/2013, ainsi que mes collègues dans le projet au sein de CDTA et ESI.

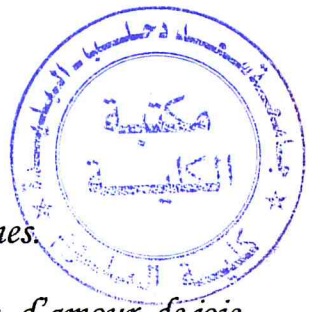
Fida

Dédicaces

Je dédie ce modeste travail :

A mes très chers parents en témoignage de ma profonde gratitude et mon incontestable reconnaissance, pour leurs sacrifices, la confiance qu'ils m'accordent, leurs soutiens permanents et tout l'amour dont ils m'entourent.

Que Dieu vous me garde au près de moi, et que m'aide de vous satisfaire d'avantage par ce modeste travail.



A mes très chers grands parents décédés : paix à vos âmes.

A mes très chers frères. Que Dieu va garantir une vie pleine de succès, d'amour, de joie et de bonheur.

A Toute ma grande famille : oncles, tantes, cousins et cousines.

A toutes mes amies de l'enfance à l'université surtout :

Latifa, Nassima, Khadidja, F.Zohra, Amina M et Amina T, ainsi

qu'A mes amies de classe en master GSI : Khadidja, Nadjah, Batoul, Hadjer, farah et bessma et tous mes amis que je ne peux pas, malheureusement, les cités tous.

A mon binôme Fida ainsi, que toute sa famille. Que Dieu vous la garde.

A tous mes collègues de la promotion 2012/2013, ainsi que mes collègues dans le projet au sein de CDTA et ESI.

Khadidja

Résumé

Résumé

La notion de sécurité dans les systèmes embarqués sur puce (System-on-Chip "SoC") est une exigence primordiale pour la majorité des applications. Ceci peut être assuré par un protocole de chiffrement implémenté sur matériel en prenant en compte diverses contraintes telles que la vitesse de traitement, la surface occupée et l'énergie limitée.

Notre projet de fin d'études, intitulé : « Contribution à la conception d'un crypto-système hybride RSA-AES embarqué sur circuit programmable FPGA», consiste à étudier la conception d'un système hybride RSA-AES, dans le but de l'intégrer dans une plate-forme de chiffrement reconfigurable visant à protéger les applications s'exécutant dans un système embarqué sur puce. Pour l'implémentation de ces deux algorithmes RSA et AES, le partitionnement de ces derniers sur les deux ressources logicielles et matérielles s'effectue généralement en tenant compte des performances à atteindre comme la surface occupée et le temps d'exécution afin d'acquérir une bonne conception.

L'implémentation matérielle de l'AES a été réalisée par trois architectures en utilisant les circuits FPGA. Les performances obtenues par ces implémentations ont été satisfaisantes en termes de surface, vitesse et fréquence de fonctionnement où l'architecture pipeline offre de meilleurs résultats comparés à l'architecture série.

L'implémentation du RSA a été faite par logiciel à l'aide du langage C, où la clé secrète de l'AES a été chiffrée par RSA.

Mots clés

Systèmes embarqués, (System-on-Chip "SoC"), chiffrement, RSA, AES, FPGA, système hybride.

Summary

The concept of security in the on chip (System-one-Chip "Ploughshare") embedded systems is a paramount requirement for the majority of the applications. This can be ensured by a protocol of coding implemented on material while taking into account various constraints such as the speed of treatment, occupied surface and limited energy.

Our project of end of studies, heading:"Contribution to the design of a hybrid crypto-system RSA-AES embedded on a programmable circuit FPGA", consists in studying the design of a hybrid system RSA-AES, with an aim of integrating it in a platform of reconfigurable coding aiming at protecting the applications being carried out in a system embedded on chip. For the implementation of these two algorithms RSA and AES, the partitioning of the two resources software and hardware is generally carried out by taking account of the performances to reach as occupied surface and time of the execution in order to acquire a good design.

The hardware implementation of the AES was effected by three architectures by using circuits FPGA. The performances obtained by these implementations were satisfactory in terms of surface, speed and frequency of operation where the pipelined architecture offers better results compared with serials.

The implementation of the RSA was made by software using the language C, or the secret key of the AES was coded by RSA.

Key words

Embedded systems, (System-one-Chip "SoC"), coding, RSA, AES, FPGA, hybrid system.

Liste des acronymes

AES	*Advanced Encryptions Standard*
API	*Application Programming Interface*
ASIC	*Applicatrion-Specific Integrated Circuit*
ATM	*Asynchronous Transfert Mode.*
BRAM	*Block Random access memory*
CBC	*Cipher Block Chaining.*
CLB	*configurable logic bloc*
CLK	*Clock*
CPLD	*Complex programmable logic device*
DES	*Data Encryptions Standard.*
DMA	*Direct Memory Access*
ECB	*Electronic Code Book*
EDK	*Embedded Development Kit*
EPROM	*Erasable Programmable Read Only Memory*
EEPROM	*Electrically-erasable programmable read-only memory*
FPGA	*Field Programmable Gate Array*
FPU	*Floating Point Unit*
FSL	*Fast Simplex Link*
FIFO	*First In First Out*
GF	*Galois Field*
FIPSP	*FIPSP*
IEEE	*Institute of Electrical and Electronics Engineers*
IHM	*Interface homme machine*

Liste des acronymes

InvMixColumns	*Inverse Mix columns*
InvSubByte	*Inverse Substitution Byte.*
InvShiftrow	*Inverse Shiftrow*
I/O	*Input/output*
IOB	*Input Output block*
IP	*Intelectuel Proprety*
ISE	*Integrated Software Environment*
LCA	*logic cells arrays*
LCD	*Liquid Crystal Display*
LED	*Light Emitter Diode*
LMB	*Local Memory Bus*
LUT	*Look up table*
MHS	*Micropocessor Hardware Spécification*
MSS	*Micropocessor Software Spécification*
MOS	*Metal Oxide Semiconductor*
Mod	*est une opération mathématique donne le reste de division*
NSA	*National Security Agency*
NIST	*National Institute of Standards and Technology.*
OPB	*On-Chip Peripheral Bus*
PDA	*Personal Digital Assistan*
PROM	*Programmable Read Only Memory*
RAM	*Random Access Memory*
Rcon	*Round Constant*
ROM	*Read Only Memory*
RSA	*Rivest Shamir Adelman.*

Liste des acronymes

S-Box	*Substitution Box.*
SHA	*Secure Hash Algorithm*
SRAM	*Static Random Access Memory*
SOC	*System-on-Chip*
SONET	*Synchronous Optical NETwork.*
SoPC	*System on Programmable Chip*
SubByte	*Substitution Byte*
UART	*Universal asynchronous receiver/transmitter*
VHDL	*Very High Speed Integrated Circuits Hardware Description Language*
VPN	*Virtual Privat Network (Réseaux privés virtuels).*
XCL	*Xilinx Cache Link*
XPS	*Xilinx Platform Studio*

Liste des Figures

Liste des Tableaux

Table des Matières

Chapitre 03 : Système Embarqué Et Sécurité Cryptographique

مكتبة الكلية

Chapitre 04 : Conception

Chapitre 05 : Simulation et Résultats

Annexes

Annexe A: Galois Field GF (2^8).

Annexe B : Détail de la S-Box et son Inverse.

Annexe C : Architecture du microblaze.

Annexe B : Architecture de FPGA.

Références Bibliographiques

Introduction Générale

Introduction Générale

1. Contexte du Travail

Durant cette dernière décennie, l'évolution technologique a conduit à l'augmentation de la densité d'intégration et de la fréquence de fonctionnement des circuits intégrés [ZEC 05].Ceci permet aujourd'hui d'assembler sur une même puce un système numérique complet, appelé système sur puce ou SoC (System-on-Chip).

Ces systèmes font de plus en plus partie de notre quotidien et se retrouvent dans des domaines comme la télécommunication, l'avionique, l'automobile, la télévision numérique, etc., et par conséquent ils sont aussi de plus en plus susceptibles à être agressés par différentes formes d'attaques.

C'est pourquoi la notion de sécurité des données dans les SoCs est de plus en plus préoccupante pour les concepteurs des circuits intégrés et a également fait l'objet de nombreux thèmes de recherche.

Afin de préserver la confidentialité des données lors d'une transmission, on implante dans certains circuits intégrés des algorithmes cryptographiques qui permettent de garantir une sécurité dans la communication et le transfert des données [Cryp 00].

Sans compter ses utilisations dans la communication secrète des militaires et du gouvernement, la cryptographie est également utilisée pour protéger beaucoup de genres de systèmes civils tels que le commerce d'Internet, les réseaux mobiles, les transactions de guichet automatique et beaucoup plus d'autres.

Le chiffrement des données est réalisé par un algorithme et une clé de chiffrement, qui permettent de chiffrer le contenu d'un message dans le but de le rendre indescriptible et seuls ceux qui ont la clé de décryptage puissent le lire.

Un algorithme de chiffrement fournit aussi la confidentialité, l'authentification, l'intégrité et le non-reniement des données.

2. Problématique et Motivations

Dans la plupart des protocoles de sécurité, on peut diviser la cryptographie en deux grandes catégories qui dépendent de la phase d'initialisation du protocole et de la phase «de protection de données».

La première phase est généralement basée sur le chiffrement asymétrique (clés publique/privée) alors que la deuxième est basée sur un chiffrement symétrique (clé secrète). L'utilisation des primitives de chiffrement asymétrique permet de résoudre les problèmes de gestion des clés dans des réseaux de grande taille.

Chaque utilisateur génère (ou reçoit de l'administrateur du système) une clé privée et sa clé publique est diffusée.

Pour que deux personnes puissent communiquer, il n'est donc plus nécessaire qu'elles aient préalablement échangé un secret, contrairement aux primitives à clé secrète. Cependant, les mécanismes mis en œuvre sont moins efficaces en termes de débit que la cryptographie symétrique.

Pour les algorithmes de chiffrement symétrique, une attaque exhaustive est hors de portée dès que l'espace des clés est suffisamment grand (minimum 64 bits).

Pour les algorithmes de chiffrement asymétrique, leurs sécurités reposent principalement sur la factorisation d'un entier (N) d'une grande taille (plus de 1024 bits).Il est donc souvent avantageux de faire appel au chiffrement hybride qui combine les avantages de ces deux modes de chiffrement.

3. Cadre du Projet

Ce projet de fin d'études, proposé au sein de l'équipe AC2 de la division Architecture des Systèmes et Multimédia (ASM) du Centre de Développement des Technologies Avancées (CDTA) et intitulé ***" Contribution à la conception d'un crypto-système hybride RSA-AES embarqué sur circuit FPGA"*** s'intéresse à la sécurité dans une certaine classe d'applications où les contraintes sont fortes (sécurité, systèmes embarqués etc.) et les exigences sont importantes (débit élevé, temps réel, etc.).

Les solutions dans ce cas de figure ne peuvent être que hardware ou mixte hardware/software.

4. Objectif du travail

L'objectif de notre travail est double : il s'agit, d'une part, de comprendre le fonctionnement des crypto-systèmes symétrique AES (Advanced Encryption Standard) et asymétrique RSA (Rivest, Shamir et Adleman) afin de les combiner et de maitriser la technologie FPGA pour implémenter notre crypto-système hybride RSA-AES sur circuit FPGA. Et d'autre part, implémenter un IP (Intellectual Property) matériel pour le chiffrement/déchiffrement des données sensibles pour les systèmes embarqués en utilisant l'AES qui sera géré par une application VHDL.

Pour ce faire, l'un des objectifs est d'étudier l'influence de la taille du chemin de données et le type d'exécution utilisé sur les performances d'exécution de notre IP-AES et d'implémenter une autre plate-forme de chiffrement/déchiffrement utilisant l'algorithme RSA pour la protection de la clé de l'IP- AES, qui sera géré par une application C.

5. Organisation du Mémoire

Nous avons organisé notre travail en cinq chapitres répartis comme suit :

Le chapitre 1 a été consacré aux généralités sur la cryptographie où ses deux types, à savoir symétrique et asymétrique ont été présentés et comparés avec ses avantages et ses inconvénients.

Dans le chapitre 2, le crypto-système hybride RSA-AES a été décrit où les crypto-systèmes symétrique AES et asymétrique RSA ont été détaillés ainsi que les architectures matérielle de l'AES pour son implémentation sur circuit FPGA et ses techniques d'optimisation pour une implémentation efficace.

Le chapitre 3, a porté sur la sécurité des systèmes embarqués et des SoCs pour intégrer notre crypto-système RSA-AES.

Le chapitre 4 a concerné trois architectures proposées pour l'IP AES à savoir Série-Série, Parallèle-Série et Parallèle pipeline qui ont été décrites. Ainsi, que l'implémentation d'une nouvelle version de la méthode SubBye dans le corps Galois Field GF (2^8) et l'implémentation software en C de la génération des clés de l'AES par RSA.

Dans le chapitre 5, sont exposés les résultats d'implémentation des architectures de l'AES sur circuit FPGA de la famille Virtex-5, sous l'environnement ISE de Xilinx, en

termes de performances temporelles et surface occupée pour chacune de ces architectures et ses différents composants.

Nous terminons notre travail par une conclusion générale et quelques perspectives.

Chapitre 01

Généralités sur la Cryptographie

1.1. Introduction

La sécurité constitue une composante cruciale de la société de l'information. Elle est à la base de l'instauration de la confiance et un des leviers de son essor. La cryptographie déclinée en algorithmes et en protocoles, permet d'assurer la sécurité dans l'univers de l'information numérique.

L'objectif de ce chapitre est de donner un aperçu des techniques de la cryptographie moderne et de ces deux types, à savoir cryptographies symétrique et asymétrique ainsi que leurs avantages et inconvénients pour les combiner en crypto-système hybride.

1.2. Définitions de la cryptologie

La cryptologie : est une science mathématique qui comporte la cryptographie et la cryptanalyse.

1. **La cryptographie :** le mot "Cryptographie" est composé des mots grecques : CRYPTO = caché et GRAPHY = écrire.

C'est donc l'art de l'écriture secrète. C'est une science permettant de préserver la confidentialité des échanges.

La cryptographie : est la science qui étudie les principes et les méthodes mathématiques appliquées à la sécurité de l'information dans des buts tels que la confidentialité, l'intégrité des données, l'identification d'entités (personnes ou machines), et l'authentification de l'origine des données. Elle tend à développer des techniques permettant de stocker des informations sensibles et de les transmettre via des réseaux non sécurisés (comme Internet) de telle sorte que ces données ne puissent être lues ou modifiées que par les personnes autorisées. *Le but traditionnel de la cryptographie est d'élaborer des méthodes permettant de transmettre des données de manière confidentielle selon les principes énoncés par Kerckhoffs* [KER 83].

2. **La cryptanalyse :** est l'étude des procédés cryptographiques dont le but est de trouver des faiblesses et, en particulier de pouvoir décrypter des textes chiffrés, c'est l'action qui permet de retrouver le texte chiffré sans connaître la clé de chiffrement.

Le cryptage ou le chiffrement : est une transformation qui consiste à remplacer un texte clair et explicite par un autre incompréhensible et indéchiffrable de manière à ce que seuls ceux qui ont la clé de décryptage (déchiffrement) puissent le lire.

Le décryptage ou déchiffrement : c'est la procédure inverse du chiffrement qui consiste à retrouver le message clair à partir du message chiffré en connaissant la clé de déchiffrement.

La clé cryptographique : est une donnée utilisée par un algorithme cryptographique et donne le moyen de chiffrer ou déchiffrer, de sceller ou de signer une autre donnée que l'on souhaite protéger ou authentifier. C'est une suite de 0 et de 1 dont la longueur peut aller de quelques dizaines de bits à quelques milliers de bits, elle dépend essentiellement de l'algorithme qui l'emploie et de l'usage à laquelle on la destine [LAB 01] et [IBR10].

1.3. Types de Crypto-Systèmes

Deux familles de cryptographie coexistent depuis les années 1970 [BER 98]. Elles se distinguent en fonction du type de clés utilisées. La cryptographie symétrique nécessite que les systèmes de chiffrement et de déchiffrement disposent de la même clé cryptographique tandis que la cryptographie asymétrique ou à clé publique considère deux clés complémentaires "publique et privée" réalisant indifféremment l'une le chiffrement et l'autre le déchiffrement.

Dans ce qui suit, ces deux familles seront détaillées avec quelques exemples d'algorithmes couramment utilisés de nos jours, ainsi que leurs avantages et inconvénients.

1.3.1. Crypto-système symétrique : appelé aussi à clé privée ou à clé secrète, se fonde sur une même clé pour chiffrer et déchiffrer un message. Il est très rapide car il utilise des clés de petites tailles et nécessite des opérations simples : addition et décalage simple à implémenter. Le problème de ce système est que la clé qui doit rester totalement confidentielle, doit être transmise au correspondant via un canal sûr non sécurisé.
Il existe plusieurs algorithmes qui fonctionnent sur ce principe le plus connu est l'**AES**.

L'AES est un algorithme inventé par Joan Daemen et Vincent Rijmen et appelé «Rijndael». C'est un sous-ensemble de Rijndael puisque il ne fonctionne qu'avec des blocs de 128 bits alors que Rijndael offre des tailles de blocs et de clés qui sont des multiples de 32. Il a été également approuvé par la NSA (National Security Agency) pour les informations top secret [IBR 10].

Enfin, le choix des clés dépend du niveau de protection que l'on souhaite attribuer aux documents ; c'est ainsi que la NSA recommande l'emploi des clés de 192 ou 256 bits pour des documents top-secrets.

De plus, son utilisation est très pratique car il consomme peu de mémoire et n'étant pas basé sur des schémas de Feistel, sa complexité est moindre et il est plus facile à implémenter.

Le système symétrique possède deux modes de cryptage par blocs et par flot.

1.3.1.1. Modes de chiffrement par blocs

Les algorithmes de chiffrement par blocs peuvent être utilisés suivant différents modes, dont les plus connues sont le mode ***ECB*** (*Electronic Code Book*) et le mode ***CBC*** (Cipher *Block Chaining).*

a. Mode *ECB*

Le mode «*ECB* » est l'algorithme de chiffrement par blocs le plus simple où un bloc de texte en clair se chiffre, indépendamment du reste des blocs composant un message et ces blocs peuvent être chiffrés en parallèle. L'inconvénient de ce mode est qu'un même bloc de texte en clair sera toujours chiffré en un même bloc de texte chiffré. Ceci permet à un attaquant actif de manipuler facilement les messages chiffrés en retirant, répétant ou inter changeant des blocs. Un autre inconvénient est l'amplification d'erreur.

b. Mode *CBC*

La solution aux problèmes posés par le mode ***ECB*** est d'utiliser une technique dite *de chainage*, dans laquelle chaque bloc du message crypté (ou cryptogramme) dépend non seulement du bloc de texte en clair correspondant, mais aussi de tous les blocs précédents, ce qui assure une certaine résistance à l'égard des attaques standards.

En mode **CBC**, chaque bloc de texte en clair est combiné par un *XOR* avec le bloc chiffré précédent, avant d'être chiffré. Le premier bloc du texte en clair est combiné avec un bloc appelé « ***vecteur d'initialisation*** ».

L'utilisation d'un vecteur d'initialisation diffère pour chaque message et permet de s'assurer que deux messages identiques (ou dont les premiers blocs sont identiques) donneront des cryptogrammes totalement différents. Le gros avantage du mode ***CBC*** réside dans le fait que la structure du texte en clair est obscurcie par le chainage. Un attaquant ne peut plus manipuler le cryptogramme, excepté en retirant des blocs au début ou à la fin. Un inconvénient est qu'il n'est plus possible de paralléliser le chiffrement des différents blocs. Ce mode pourrait entrainer une propagation d'erreur importante. De ce fait, une erreur d'un bit sur le texte en clair affectera tous les blocs chiffrés suivants. Par contre, si un bit de

texte chiffré est modifié au cours du transfert, seul le bloc de texte en clair correspondant et un bit du bloc de texte en clair suivant seront endommagés : le mode ***CBC*** est dit aussi « *auto-réparateur* ».

c. Chiffrement par blocs avec itération

Un algorithme de chiffrement par blocs avec itération est un algorithme qui chiffre les blocs par un processus comportant plusieurs rondes. Dans chaque ronde, la même transformation est appliquée au bloc, en utilisant une sous-clé dérivée de la clé de chiffrement. En général, un nombre de rondes élevé garantit une meilleure sécurité au détriment bien évidemment des performances.

Un cas particulier d'algorithmes de chiffrement par blocs avec itérations est la famille des «*Chiffres de Feistel*».

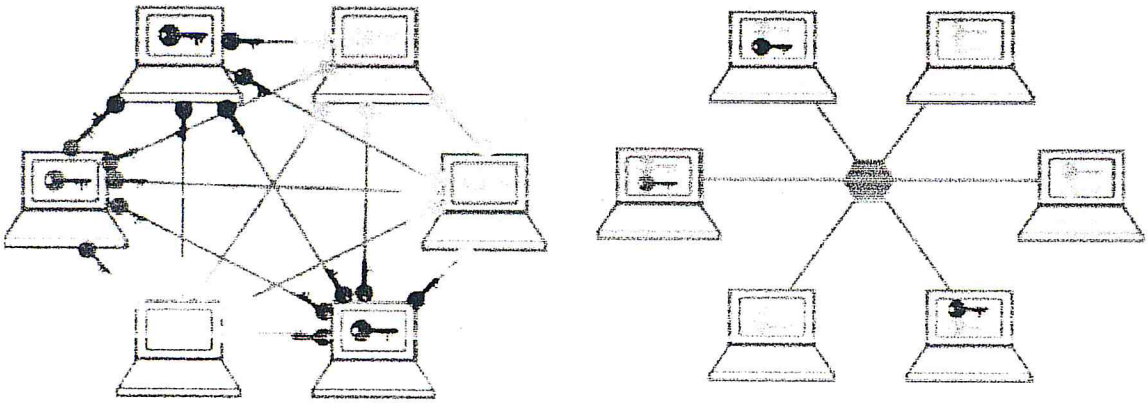
Dans un *chiffre de Feistel*, un bloc de texte en clair est découpé en deux moitiés, la transformation de ronde est appliquée à l'une de ces deux moitiés et le résultat est combiné avec l'autre moitié par un l'opération *XOR*. Les deux moitiés sont alors inversées pour l'application de la ronde suivante. Un avantage de ce type d'algorithme est que le chiffrement et le déchiffrement sont structurellement identiques.

1.3.2. Crypto-Système asymétrique

Il utilise une paire de clés indépendantes, une clé publique connue par tous pour le cryptage et une clé secrète pour le décryptage. Avec ce mode tout le monde peut chiffrer un message, mais seul le propriétaire de la clé secrète pourra le déchiffrer. Il permet d'assurer la *confidentialité* et la *signature* des messages.

Le principe de ce système a été introduit en 1976 par *Diffie* et *Hellman* qui l'a conçu pour créer une **paire de clés** dans le but de résoudre le problème de distribution des clés posé par la cryptographie à clé secrète. De nombreux algorithmes permettant de réaliser un crypto système à clé publique ont été proposés. Ils sont le plus souvent basés sur des problèmes mathématiques difficiles à résoudre [DIH 76], [LAU 10], [IBR 10] et [TAB 10].

Le RSA nommé d'après les noms de ses inventeurs (RIVEST, SHAMIR et ADELMAN), est le premier algorithme asymétrique publié en 1977. Sa sécurité 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.



Symétrique asymétrique

Figure 1.1. Gestion des clés dans les chiffrements symétrique et asymétrique.

Le tableau 1.1 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 [FE 173].

Dans la plupart des applications actuelles, la meilleure solution consiste à utiliser un système hybride où les cryptographies symétrique et asymétrique sont combinées et on exploite leurs avantages du mieux que possible pour rendre les systèmes performants.

Les clés publique et privée sont une fonction d'un couple de grands nombres premiers de 1024 bits ou plus. 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 nombres 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.

1.4. Comparaison entre crypto-systèmes symétrique et asymétrique

Les crypto-systèmes symétriques et asymétriques présentent chacun des avantages et des inconvénients.

La principale difficulté des algorithmes de chiffrement symétrique ou à clé secrète 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 [IBR 10].

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.

Pour calculer le nombre de clés pour un groupe de 10 personnes qui veulent communiquer, il leur faut 10 couples de clés soit 20 clés alors qu'il faut 45 clés dans le cas du cryptage symétrique. Ce processus est représenté sur la figure 1.1

La cryptographie symétrique sert à chiffrer les messages puisque cette fonction est très rapide et la cryptographie asymétrique sert à échanger les clés entre deux utilisateurs puisqu'il faut leur donner des clés identiques et de manière sûre pour qu'ils chiffrent et déchiffrent leur message. Comme l'échange de clés ne se fait qu'une seule fois, le système n'en est que très peu ralenti [CAN 05].

مكتبة الكلية

1.5. Fonctionnement du crypto-système hybride

Un crypto-système hybride consiste à utiliser les avantages des chiffrements symétrique et asymétrique tels que:

- La rapidité d'un système symétrique qui grâce à une clé secrète valide le temps du transfert de l'information, ou le temps d'une session.
- La possibilité de transmettre la clé secrète par un crypto-système asymétrique.

Il utilise le cryptage asymétrique avec une paire de clés publique et privée pour échanger une clé secrète symétrique ou clé de session qui servira à crypter le message.

La plupart des systèmes hybrides procèdent de la manière suivante : une clé aléatoire est générée pour l'algorithme symétrique. La taille de cette clé est entre 128 et 512 bits selon les algorithmes. L'algorithme de chiffrement symétrique est ensuite utilisé pour chiffrer le message.

La clé aléatoire quant à elle, se voit chiffrée grâce à la clé publique du destinataire, c'est ici qu'intervient la cryptographie asymétrique tels que le RSA. Comme la clé est courte, ce cryptage prend peu de temps. Crypter l'ensemble du message avec un algorithme asymétrique serait bien plus lourd, c'est pourquoi on préfère passer par un algorithme symétrique.

Il suffit ensuite d'envoyer le message chiffré avec l'algorithme symétrique, accompagné par la clé chiffrée correspondante.

Le destinataire déchiffre la clé symétrique avec sa clé privée et via un déchiffrement symétrique, retrouve le message [IBR 10].

Dans notre travail nous nous intéressons à un crypto système hybride qui combinant l'AES pour chiffrer le message et le RSA pour chiffrer la clé secrète de l'AES.

1.6. Conclusion

Dans ce chapitre, nous avons présenté une introduction générale à la cryptographie où les différents types de la cryptographie ont été exposés avec leurs avantages et inconvénients. Une conclusion est faite que les protocoles cryptographiques à clé publique présentent l'avantage d'échanger des messages de manière sûre sans échange préalable de secret et les protocoles symétriques sont rapides et simples à implémenter.

Les avantages des chiffrements symétrique et asymétrique ont été combinés pour décrire le principe de fonctionnement d'un crypto-système hybride basé sur le RSA et l'AES. Ce dernier fera l'objet de notre étude dans le prochain chapitre où nous présenterons son principe de fonctionnement et les différentes architectures qui ont été proposées dans la littérature pour son implémentation, ainsi que les techniques d'optimisation pour une éventuelle implémentation de l'AES sur un circuit FPGA.

Chapitre 02

Protocole De Chiffrement Hybride RSA-AES

$\mathbf{d \equiv e^{-1} mod ((p - 1) \times (q - 1))}$.

- Le couple **(e, N)** est la clé publique et **(d, N)** est la clé privée.

2.2.2. Chiffrement d'un message

Pour chiffrer un message clair **m,** on le découpe en messages **M** tel que M< N et on calcule l'entier chiffré **C** en utilisant la clé publique (e, N) : $C = M^e \bmod N$.

Le message chiffré **C** peut alors être transmis.

2.2.3. Déchiffrement d'un message

Pour déchiffrer un message chiffré **C** (retrouver le message clair **M**), il suffit de calculer : $M = C^d \bmod N$, tel que : e x d = 1 mod [(p-1) x (q-1)] [ENG 06].

2.3. L'Advanced Encryption Standard

L'AES (Advanced Encryption Standard) est un algorithme de chiffrement symétrique. Il remporta en octobre 2000 le concours AES, lancé en 1997 par le NIST et devint le nouveau standard de chiffrement pour les organisations du gouvernement des États-Unis. Il a été également approuvé par la NSA (National Security Agency) pour les informations top secrètes.

Le choix de l'algorithme AES répond à de nombreux critères tels que : sa résistance aux attaques connues, sa facilité de calcul, ses besoins faibles en ressources et en mémoire, sa flexibilité d'implémentation en termes de tailles de clés et de blocs, sa simplicité dans le design et la possibilité de son implémentation logicielle et matérielle [ROL 02] et [IBR 10].

L'AES est aussi connu sous le nom de Rijndael, le pseudonyme de ses deux concepteurs Joan Daemen et Vincent Rijmen. Il s'exécute en blocs de taille fixe de 128 bits où un certain nombre de transformations exprimées en rounds convertissent le texte en clair en un texte chiffré en utilisant des sous clés générées à partir de la clé de chiffrement.

Chaque round se compose de plusieurs étapes de traitement, à savoir AddRoundKey, SubBytes, ShiftRows et MixColumns [DAR 99], [ROL 02].

Un ensemble de rounds inverses sont appliqués pour retrouver le texte original en utilisant la même clé de chiffrement. La figure 2.1 montre les étapes de l'AES-128.

des attaques basées sur de simples propriétés algébriques, la « ***S-Box*** » est construite en combinant cette fonction inverse avec une transformation affine inversible f. On a donc :

$$S\text{-}Box[a] = f(g(a)),\ \forall\ a \in GF[2^8]$$

Les concepteurs ont également fait en sorte que cette application ***S-Box*** n'admette pas de point fixe, ni de point fixe opposé:

Enfin, la fonction affine f : $GF(2^8) \rightarrow GF(2^8)$ est définie par : [ROL 02].

$$S\text{-}Box[a] + a \neq 00,\ \forall\ a \in F_{256}$$

$$S\text{-}Box[a] + a \neq FF,\ \forall a \in F_{256}$$

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

L'opération inverse de *SubByte* est notée *InvSubByte et* consiste à effectuer la même transformation que la SubByte mais à partir de la ***S-Boxinverse***, notée ***InvS-Box ou S-box^{-1}***.

Remarque: mathématiquement, comme la fonction g est son propre inverse,

on a :$InvS\text{-}Box(a) = S\text{-}Box^{-1}(a) = g^{-1}(f^{-1}(a)) = g(f^{-1}(a)),\ \forall\ a \in GF[2^8]$.

La fonction affine inverse f^{-1}est définie par la figure suivante :

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

2.4.3. Etape *ShiftRows*

L'étape *ShiftRows* opère sur les lignes de la matrice ***State*** et permet d'effectuer un décalage cyclique vers la gauche des lignes de State selon différents cas. La ligne 0 n'est pas décalée, la ligne 1 l'est de 1 octet, la ligne 2 de *2* octets, et la ligne 3 de *3* octets, comme illustré sur la figure 2.5.

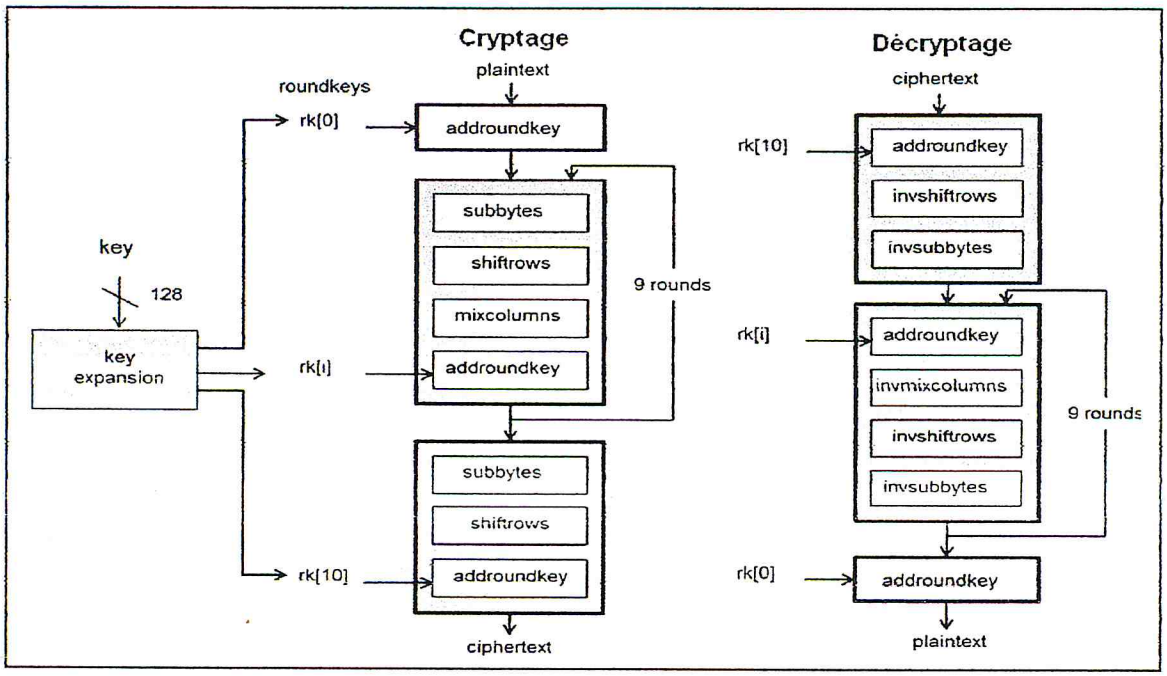


Figure 2.1. Opérations de l'algorithme AES [ROL 02].

Chaque fonction de chiffrement ou de déchiffrement se divise en trois blocs:

- Un "Round initial" avec la clé générale qui compte qu'une seule opération "AddRoundKey".
- Une série de rounds réguliers, "Etape des N Rounds", avec N le nombre d'itérations.
- Un "Round Final" qui est identique à l'un des N tours, mais l'opération MixColumns n'est pas effectuée lors de ce dernier round.

2.3.1. Entrées / Sorties de l'AES

Les entrées/sorties de l'AES consistent en des séquences de **128 bits**. L'AES nécessite une clé secrète de chiffrement de 128, 192 ou 256 bits. Les différentes transformations opèrent sur le résultat intermédiaire du chiffrement, appelé State [DAR 99] et [ROL 02].

Les entrées state et les sorties de l'AES sont représentées par des matrices rectangulaires de $4 \times N_b$ octets, ($N_b = 4$, reflète le nombre de mots de 32 bits dans un tel bloc) comme le montre la figure 2.2. Cette matrice comporte quatre lignes et quatre colonnes.

in_0	in_4	in_8	in_{12}	$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	out_0	out_4	out_8	out_{12}
in_1	in_5	in_9	in_{13}	$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	out_1	out_5	out_9	out_{13}
in_2	in_6	in_{10}	in_{14}	$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	out_2	out_6	out_{10}	out_{14}
in_3	in_7	in_{11}	in_{15}	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	out_3	out_7	out_{11}	out_{15}

Figure 2.2. Entrée, Etat intermédiaire (State) et Sortie de l'AES [DUT 11].

De même la clé de chiffrement est représentée par une matrice rectangulaire de quatre lignes, le nombre de colonnes noté $\mathbf{N_k = 4, 6 ou 8}$ ($\mathbf{N_k}$: reflète la longueur de la clé divisée par 32). Cette représentation est illustrée par la figure 2.3 [DAR 99].

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Figure 2.3. Exemple de clé de chiffrement (cas de NK = 4)

Un bloc peut être représenté comme une matrice carrée d'octets **4×4 (*State*).** Les octets lus-en entrée y sont copiés colonne après colonne (chaque colonne représente un mot lu). $\{s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j}\}$représente un mot wj, $j=0,1,2$ ou 3 ; plus précisément :

$$\mathbf{w0} = \begin{Bmatrix} s_{0,0} \\ s_{1,0} \\ s_{2,0} \\ s_{3,0} \end{Bmatrix} \quad ; \quad \mathbf{w1} = \begin{Bmatrix} s_{0,1} \\ s_{1,1} \\ s_{2,1} \\ s_{3,1} \end{Bmatrix}$$

$$\mathbf{w2} = \begin{Bmatrix} s_{0,2} \\ s_{1,2} \\ s_{2,2} \\ s_{3,2} \end{Bmatrix} \quad ; \quad \mathbf{w3} = \begin{Bmatrix} s_{0,3} \\ s_{1,3} \\ s_{2,3} \\ s_{3,3} \end{Bmatrix}$$

L'AES exécute une séquence des rounds qui seront détaillés par la suite. On note **Nr** le nombre de rounds qui doivent être effectués. Ce nombre dépend des valeurs **Nb** et **Nk**. Les différentes configurations possibles sont détaillées dans le tableau 2.1.

Nr	Nb= 4
Nk= 4	10
Nk= 6	12
Nk= 8	14

Tableau 2.1. Nombre des rounds de l'AES.

2.3.2. L'Arithmétique de l'AES

L'**AES** est un algorithme de chiffrement orienté « **byte** », un **byte** est la concaténation de 8 bits **{b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0}**. Les valeurs de ces bytes peuvent être interprétées comme des éléments dans le corps Galois Field GF (2^8), et peuvent être représentés algébriquement sous forme de polynômes de degrés ≤ 7[DAR 99], [ROL 02] et [DUT 11].

2.3.3. Le corps de Galois (2^8)

Les éléments du corps GF (Galois Field) (2^8) peuvent être représentés de plusieurs façons qui sont tous isomorphes.

Un octet **b** (byte), est représenté par les bits: b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0. Il peut être aussi considéré comme un polynôme ayant des coefficients dans {0,1} représenté par:

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

L'information peut être représentée sous trois formes

1. **Binaire :** avec {0, 1}
2. **Polynômiale :** avec des coefficients dans {0,1}.
3. **Hexadécimal :** avec des éléments dans {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.

2.4. Fonctionnement de l'AES

L'*AES* opère sur une matrice 4 x **Nb** d'éléments de ***GF(2^8)***, notée « ***State*** ». Le chiffrement ***AES*** consiste en une addition initiale de state avec la clé, notée *AddRoundKey*, suivie par (**Nr-1**) rounds, chacun de ces rounds est constitué de quatre transformations :

1. ***SubByte***: il s'agit d'une substitution non-linéaire où chaque octet « byte » est remplacé par un autre octet choisi dans une table particulière (***S-Box***).
2. ***ShiftRows :*** est une étape de transposition ou chaque élément de la matrice State est décalé cycliquement à gauche d'un certain nombre de colonnes.
3. ***MixColumns :*** effectue un produit matriciel en opérant sur chaque colonne (vu alors comme un vecteur) de la matrice.
4. ***AddRoundKey :*** qui combine par addition de chaque octet avec l'octet correspondant dans une clé de round obtenue par l'opération de la diversification de la clé de chiffrement.

Enfin, un round final ***Final Round*** est appliqué (c'est le round où l'étape *MixColumns* est omise).

La clé de round pour la transformation AddRoundKey sera notée *RoundKeys [i]*. La dérivation de la clé de chiffrement **K** est notée *Key Expansion*.

Les sections suivantes détaillent chacune de ces opérations [ARM 11].

2.4.1. Etape d'Initialisation

On part du premier bloc de message et de la clé de chiffrement après qu'on les découpe en blocs de 128 bits et on les transforme en hexadécimal.

Bloc de message **la clé de chiffrement**

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \quad \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix}$$

On les additionne par un ou exclusif « XOR ».

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \oplus \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix} = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix}$$

Le résultat est la matrice « *State* » où : $s_{i,j} = a_{i,j} \oplus k_{i,j}$; $0 \leq i, j \leq 3$

2.4.2. Etape *SubByte*

L'étape SubByte correspond à une transformation non-linéaire de l'algorithme AES. Dans cette étape, chaque élément de la matrice « *State* » est substitué selon une table de substitution inversible notée « ***S-Box*** » comme le montre la figure 2.4.

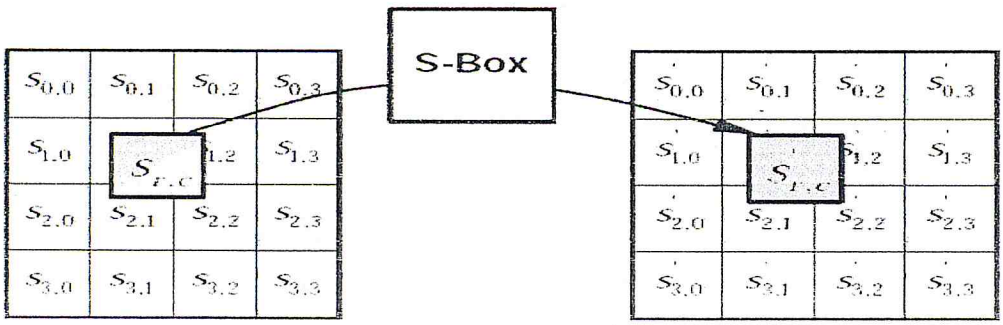


Figure 2.4. L'opération SubByte [DUT 11].

La table *S-Box* dérive de la fonction : $g : a \in GF(2^8) \rightarrow a^{-1} \in GF(2^8)$, *0* étant son propre inverse. Cette fonction est connue pour ses bonnes propriétés de non-linéarité. Afin d'éviter

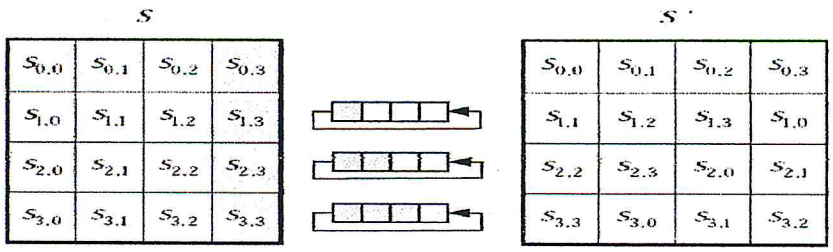
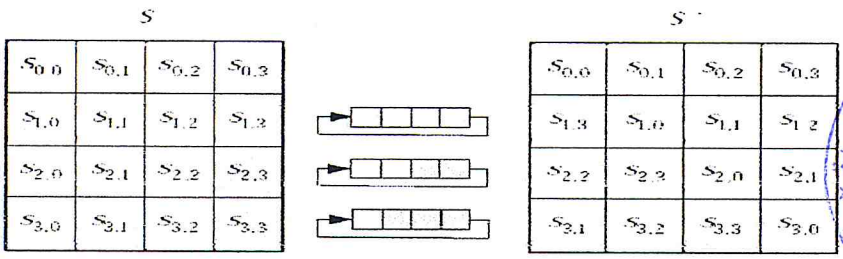


Figure 2.5. L'opération ShiftRows [DUT 11].

L'opération inverse de *ShiftRows* est notée *InvShiftRows*, consiste à effectuer au niveau des lignes un décalage cyclique à droite de ***Ci*** positions comme le montre la figure 2.6 [DAR 99].



جامعة سعد دحلب - البليدة
مكتبة الكلية
كلية العلوم

Figure 2.6. L'opération InvShiftRows [DUT 11].

2.4.4. Etape MixColumns

La transformation *MixColumns* opère sur les colonnes de la matrice ***State*** comme le montre la figure 2.7.

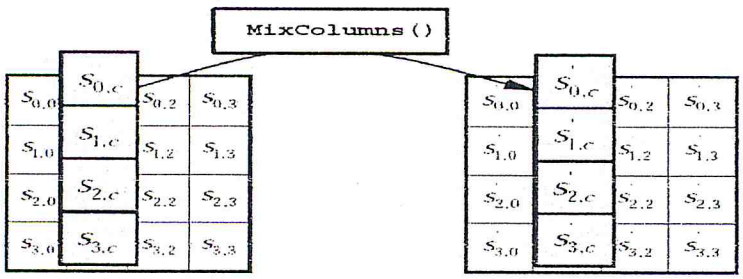


Figure 2.7. L'opération MixColumns [DUT 11].

Elle est traitée comme un polynôme $a(x)$ de degré 3 à coefficients dans le $GF(2^8)$.La transformation *MixColumns* consiste alors à effectuer pour chaque colonne de State une multiplication par un polynôme ***c(x)*** fixé par l'AES, suivie d'une réduction modulo le polynôme $p(x) = x^4+1$.

Dans *MixColumns*, on réalise donc l'opération :$(03\ x^3 + x^2 + 02) \times a(x) \mod (x^4+1)$ dans le corps GF (2^8).Cette étape revient à effectuer le calcul suivant :

$$\begin{pmatrix} S'_0 \\ S'_1 \\ S'_2 \\ S'_3 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 02\ 01\ 01 \end{pmatrix} \otimes \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix}$$

✓ Le résultat d'une colonne est donné par:

$S'_{0,c} = ((02) \cdot S_{0,c}) + ((03) \cdot S_{1,c}) + S_{2,c} + S_{3,c}$

$S'_{1,c} = S0,c + ((02) \cdot S_{1,c}) + ((03) \cdot S_{1,c}) + S_{3,c}$

$S'_{2,c} = S0,c + S1,c + ((02) \cdot S_{2,c}) + ((03) \cdot S_{3,c})$

$S'_{3,c} = ((03) \cdot S_{0,c}) + ((02) \cdot S_{1,c}) + S_{2,c} + S_{3,c}$

→ 2 opérations de multiplication et 3 opérations XORs pour claque élément de la colonne.

L'opération inverse de *MixColumns* est notée *InvMixColumns* et consiste à effectuer la même opération mais à partir d'une multiplication par un autre polynôme $d(x) = c^{-1}(x)$ donné par la relation : $(03\ x^3 + x^2 + x + 02) * d(x) = 01 \bmod (x^4 + 1)$.

On obtient ainsi :

$d(x) = 0B\ x^3 + 0D\ x^2 + 09\ x + 0E$. Matriciellement, cette étape revient à effectuer le calcul suivant :

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

2.4.5. Etape AddRoundKey

Lors de l'étape *AddRoundKey*, la matrice *State* est modifiée en l'additionnant (au sens de l'addition termes à termes dans le corps composé $GF(2^8)$) avec une clé de round [DAR 99], [ARM 11] et [DUT 11].

2.4.6. Diversification de la clé dans l'*AES*

Cette étape, notée ***Key Expansion***, permet de diversifier la clé de chiffrement K (de 4×Nk octets) dans une clé étendue W de 4×Nb× (Nr + 1) octets. Cette opération prend en entrée 4 words (16 bytes) et produit en sortie 44 words (176 bytes).

Au début, la clé est copiée dans les quatre premiers words de ***Key Expansion***. Chaque word W[i] dépend du word précédent W [i-1] et son correspondent dans les quatre words précédents W [i-4].

Pour le word dont la position est un multiple de 4, une fonction complexe g est utilisée.

1- **RotWord** : consiste à exécuter un décalage vers le haut d'un byte.
2- **SubByte** : consiste à substituer chaque byte, en utilisant la table S-box.
3- Le résultat des étapes 1 et 2 exécute un XOR avec un élément correspondant de la table RCon[j], (où j reflète le numéro de tour), qui est définie récursivement par : [ARM 11].

$$\mathbf{Rcon\ [i] = [x^{i-1}, 00, 00, 00]\ ,\forall\ i \geq 1.}$$

2.4.7. Propriétés Algorithmiques

Il est possible d'inverser l'ordre des transformations SubByte et ShiftRows et également pour InvSubByte et InvShiftRows.

Les transformations MixColumns et InvMixColumns sont linéaires vis-à-vis de la colonne d'entrée, c'est-à-dire : InvMixColumns (state XOR RoundKey) = InvMixColumns (state) XOR InvMixColumns (RoundKey) [DUT 11] et [DAR 99].

2.5. Architectures matérielles de l'AES dans la littérature

L'implémentation matérielle de l'algorithme AES varie selon la taille du bus de données (DataPath) et la technologie utilisée. Plusieurs conceptions ont été proposé dans la littérature utilisant la technologie ASIC tandis que d'autre utilisent la technologie FPGA. Mais avec l'avènement des nouvelles technologies dans le domaine des FPGAs, ils sont devenus de plus en plus préférés qu'aux ASICs. Les FPGAs sont particulièrement rapides en termes de débit lorsque les désignes sont implémentés en utilisant le principe de pipeline. La vitesse qu'elle peut atteindre est fixée par le plus long chemin dans la conception [SNS 12].

Vu les avantages innombrables d'employer une plate-forme de FPGA par rapport aux autres plateformes comme il est signalé dans le chapitre précédent, nous avons choisi de passer en revue par des implémentations matérielles de l'AES exclusivement sur FPGA.

Les implémentations de l'AES peuvent être devisées en trois types principaux en fonction de la taille de la largeur de chemin de données. Le premier type s'exécute avec un DataPath

de 8 bits, le deuxième type est procédé avec un DataPath de 32 bits et le troisième type de ces implémentations est une architecture avec un DataPath 128 bits. Les sections suivantes présentent les travaux précédents pour chacun de ces types.

2.5.1. Architecture 8 bits

La plus part des conceptions d'AES avec 8 bits utilisent des architectures itératives. Ce type d'architectures utilise un seul étage de chiffrement/déchiffrement de l'algorithme AES avec un retour à la fin. Cette architecture est implémentée en [GOB 06], dont le but est de conserver en maximum la surface d'implémentation, avec un débit faible par rapport aux autres types. Un autre travail « ***a compact 8-bit AES crypto processor for low-cost and low power applications*** » a été présenté en [HAS 10], supporte le processus de cryptage et de décryptage pour un coût minimal du matériel utilisé.

2.5.2. Architecture 32 bits

Avec l'avènement des nouvelles technologies dans le domaine des FPGAs, ils sont devenus de plus en plus performent (riche en termes de ressources). Avec cette tendance la mise en œuvre de l'AES avec bloc de 32-bit a commencé à être présenter après 2003 pour les applications à faible débit et peu de surface. Tels que les PDA, réseau sans fil et systèmes embarqués [HAS 10]. Des implémentations avec 32 bits sont devenues plus utilisées et demandées afin d'atteindre un meilleur débit avec une consommation moyenne de la surface. Ce type d'architecture est implémenté par [CHO 03], [RSQ 04] et [CHC 08]. Pour assurer l'intégrité des données une architecture proposée en [SSG 05] basée sur un chaînage de message, ce travail propose l'implémentation d'un protocole de cryptage sécurisé sur FPGA avec une évaluation de performance et de coût, elle opère sur une colonne de données de 32 bits, 41 cycles d'horloge sont nécessaires pour compléter le chiffrement d'un bloc de 128 bits, dans ce travail l'implémentation de la table SBox est basée sur des RAMs et garantie une haute performance. L'implémentation de cette architecture alloue moins d'espace mémoire avec une fréquence élevée.

Récemment en 2013 ce type d'architecture est proposé par [CHO 13] et [KAK 13], dont le but est de garantir une optimisation dans les ressources d'implémentation avec un débit moyen.

2.5.3. Architecture 128 bits

Ce type d'architecture est spécifié pour les applications à fort débit [HAS 10], pour cela la plupart de ces architectures utilisent le principe du pipeline. Et ces contributions ne conviennent généralement que la partie la plus critique dans l'algorithme AES qui est la S-Box [SNS 12].

Une architecture non pipeline de l'algorithme AES 128 bits est présentée dans [HDH 05] et [ELR 04], supporte les deux modes de fonctionnement sans et avec retour avec un débit maximal atteint 3,84 Gbps suivi par une fréquence de 330 MHZ. Des autres travaux dans [GVM 10], [HAM 10] et dans [HOV 04] proposent l'implémentation de l'AES avec un DataPath de 128 bits afin d'atteindre un débit élevé. Le travail présenté dans [ALI 10] est une architecture pipeline complet implémentée sur les circuits FPGAs, atteint un débit maximal de 21,54 Gbps. Une autre architecture a été proposée dans [SKS 05] présente l'implémentation de l'AES 128 bits en mode pipeline pour la transformation SubByte afin de réduire l'occupation des ressources.

Récemment, en 2013 deux architectures de l'AES 128 bits ont été proposé par les travaux dans [BAR 13] et[GAF 13], une de ces architectures suit une implémentation entièrement pipeline avec une implémentation de la SubByte par la technique de calcule dans le corps Galois Field dans la base 8 avec quatre étages pipelines, pour une implémentation matérielle moins complexe et moins d'occupation des ressources par rapport à une implémentation avec des LUTs, cette architecture devise les transformations des ronds en deux étages, l'autre architecture est destinée au cryptage des signaux audio, elle est d'un même DataPath mais avec un sous pipeline entre les opérations pour augmenter en plus le débit avec une augmentation dans les ressources utilisées.

Donc selon les besoins et l'utilité des architectures, nous choisissons la taille des bus de données. Le bus de 8 bits est le bon choix pour une faible consommation des ressources avec une simple S-Box pour les applications où la fiable occupation de surface et dissipation de puissance sont importantes. En outre les architectures d'AES de 128 bits avec 16 S-Boxs pour la transformation SubByte d'un bloc de données de 128 bits peuvent atteindre un débit élevé. Enfin les architectures de 32 bits avec 4 S-Boxs constituent un bon compromis entre les deux 8 bits et 128 bits, elles tiennent compte d'un rendement beaucoup plus élevé que des architectures de 8 bits mais exigent seulement peu de surface d'implémentations par rapport une architecture de 128-bits.

2.6. Approches pour une implémentation matérielle efficace d'AES

Les techniques d'optimisation pour l'implémentation matérielle de l'algorithme AES peuvent être classées en deux catégories:[SNS 12].

2.6.1. Optimisation Architecturelle

Deux types d'architectures peuvent être utilisés pour augmenter la vitesse de cryptage et de décryptage en dupliquant le matériel pour l'implémentation de chaque round. Ces architectures sont basées sur la force de pipeline et de sous-pipeline.

2.6.1.1. Pipeline

Le pipeline est le mode d'implémentation où plusieurs blocs de données seront sur le processeur en même temps, ce qui donne une vitesse de cryptage/décryptage élevée. Il est construit par l'insertion des lignes de registres entre la logique combinatoire. Les rounds entre deux registres consécutifs forment une structure en pipeline d'un seul étage. A chaque cycle d'horloge les données traitées passent à l'étape suivante et sa position est occupée par le bloc de données suivant [SNS 12].

2.6.1.2. Sous-Pipeline

En cas de traitement sous-pipeline, les registres pipelines sont insérés à l'intérieur des rounds de l'algorithme de chiffrement AES. Le nombre des registres pipelines internes est répété K fois, où le nombre de ronds K dépend de la valeur maximale de la surface ou du débit maximal requis.

Ce mode, offre un débit nettement plus élevé. Toutefois, la surface de circuit est assez grande [SNS 12].

2.6.2. Optimisation Algorithmique

Un round particulier dans l'AES exécute les opérations ShiftRows/InvShiftRows, AddRoundKey, SubByte/InvSubByte et MixColumns/InvMixColumns, l'optimisation algorithmique manipule les techniques d'optimisation de ces opérations, mais l'opération ShiftRows/InvShiftRows n'entraine pas des portes logiques, aussi l'opération AddRoundKey requit seulement une opération XOR, donc aucune optimisation ne peut être réalisée sur cette opération. Cependant l'optimisation peut être opérée sur les deux transformations SubByte/InvSubByte et MixColumnls/InvMixColumns de cryptage/décryptage [SNS 12].

2.6.2.1. Implémentation de SubByte/InvSubByte (S-Box/S-Box^{-1})

La transformation SubByte/InvSubByte est généralement implémentée par l'utilisation des Look Up Tables (LUTs). Cependant cette implémentation consomme un espace énorme de surface lorsque plusieurs tours sont implémentés en même temps. Une autre implémentation utilisant le principe du corps GF(2^8) est adoptée pour les architectures où la surface du circuit est importante, où le bon choix est de transformer l'opération arithmétique dans le GF(2^8) vers l'opération dans le GF ((2^4)^2) pour le calcul des valeurs de la table S-Box. Cette transformation nécessite peux de ressources pour une implémentation sur les Look Up Tables (LUTs). Néanmoins, celle-ci engendre un long délai. Dans le corps GF(2^8) chaque bit peut considérer comme étant le coefficient de puissance correspondante dans le polynôme GF(2^8) [SNS 12].

2.6.2.2. Implémentation de MixColumns/ InvMixColumns

2.6.2.2.1. Implémentation de MixColumns

Dans la transformation MC les constantes multiplicatives (02) et (03) dans le GF(2^8) peuvent être implémentées par l'utilisation de la fonction Xtime, où le chemin critique est celui de quatre portes Xors [SNS 12].

2.6.2.2.2. Implémentation d'InvMixColumns (IMC)

La transformation IMC utilisée dans le décryptage est plus compliquée. Les constantes multiplicatives dans l'IMC peuvent être exprimées par :

- {0b}X= {08}X+{02}X+ X,
- {0d}X= {08}X+{04}X+ X,
- {09}X= {08}X+ X,
- {0e}X= {08}X+{04}X+ {02} X [SNS 12].

2.7.Conclusion

Dans ce chapitre, nous avons d'abord présenté le principe de fonctionnement des deux algorithmes qui constituent notre crypto-système hybride AES et RSA ainsi que leurs concepts de bases, leurs points forts et leurs applications.

Nous avons axés ensuite sur les architectures matérielles de l'AES proposés dans la littérature avec des différents modes d'exécution : séquentiel, pipeline et sous-pipeline pour une implémentation matérielle efficace de l'AES optimisée en termes de surface et vitesse.

Dans le but d'embarquer notre crypto-système hybride RSA-AES sur circuit FPGA, le chapitre suivant est consacré à la présentation des systèmes sur puce. En particulier les systèmes sur puce implémentés sur circuits FPGA SoPC (system on programmable chip).

Chapitre 03

Système Embarqué Et Sécurité Cryptographique

3.2.2. Caractéristiques principales d'un système embarqué

Les principales caractéristiques d'un système embarqué sont les suivantes :

- C'est un système principalement numérique.
- Utilise généralement un processeur.
- Exécute un logiciel dédié pour réaliser une fonctionnalité bien précise.
- Il n'a pas réellement de clavier standard.
- L'affichage est limité à un écran LCD ou n'existe pas du tout.
- Ce n'est pas un PC.
- N'exécute pas une application scientifique ou commerciale traditionnelle.

De cette initiative, nous pouvons dire: qu'un système embarqué n'exécute qu'une application dédiée alors qu'un PC standard peut exécuter tout type d'applications.

Le système embarqué est généralement composé d'un système contrôleur (le système informatique) qui agit sur un système contrôlé (l'environnement physique de l'application).

3.2.3. Types de système embarqué

3.2.3.1. Informatique générale (*General Computing*) : dans les applications empaquetés dans un système embarqué, jeu vidéo et set-top box).

3.2.3.2. Système de contrôle (*Control Systems*) : pour le contrôle des systèmes en temps réel, moteur d'automobile, processus chimique, processus nucléaire, système de navigation aérien et le contrôle des fonctions d'un satellite de télécommunication.

3.2.3.3. Traitement de signal *(Signal Processing)* : pour le calcul de grosses quantités de données, radar, sonar, compression vidéo et traitement du son et de la parole.

3.2.3.4. Communication et réseau (*Communication & Networking*) : pour la transmission d'information et commutation, téléphone, internet et les routeurs, commutateur, modems...

3.3. Système embarqué sur puce SoC

Le terme SoC (Système on Chip) ou système sur puce désigne une classe de circuits intégrés actuels. Un Système sur puce est un système embarqué sur une puce, pouvant intégrer sur un même substrat de silicium un ou plusieurs microprocesseurs, de la mémoire

(statique, dynamique, flash, ROM, PROM, EPROM, EEPROM), des périphériques d'interface, ou tout autre composant nécessaire à la réalisation de la fonction attendue.

Les intérêts sont multiples (miniaturisation accrue, meilleures performances etc...) et ont donc largement contribué à pousser dans cette voie la majorité des fabricants de semi-conducteurs.

Pour la conception d'un SoC, il est primordial de sélectionner la plate-forme matérielle d'implémentation. Cette sélection est basée sur la réponse à la question suivante : *Comment atteindre le fonctionnement correct d'un système avec un faible coût de réalisation et en respectant des contraintes supplémentaires ?*

En effet, pour les systèmes ne nécessitant pas de grandes capacités de traitement de données, les microcontrôleurs ou les microprocesseurs bas de gamme peuvent être un bon choix.

Si les besoins en calcul sont plus importants, les microprocesseurs plus puissants ou les DSPs peuvent être considérés. Ce type de solution est très flexible puisqu'il est basé principalement sur une écriture de programmes.

Pour obtenir des performances élevées, il est nécessaire de choisir des circuits spécifiques en utilisant des circuits programmable de type FPGAs. Ces derniers sont généralement recommandés pour le prototypage et la production en faible série. On parle alors de système SoPC (System on Programmable Chip).

Pour une large série de fabrication, les circuits ASICs (Applicatrion-Specific Integrated Circuit)sont utilisés[DBS 06].

Les différents types de plateforme matérielle d'implémentation sont montrés sur la figure 3.1.

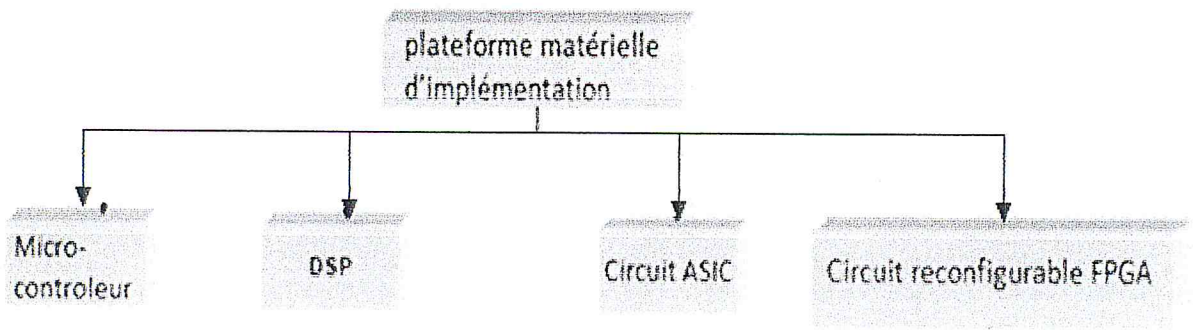


Figure 3.1. Plate-formes matérielles d'implémentation des SoCs [SCH 10].

Avec l'intégration d'une panoplie de composants sur une même puce, celle-ci est souvent considérée comme étant une architecture hétérogène. On peut référencer plusieurs types de

composants qui constituent l'architecture d'un SoC suivant le degré de personnalisation possible.

En effet, les Soft-cores sont des IPs (Intellectuel Property) décrits dans un langage de description matériel synthétisable (VHDL, Verilog,...), très souples pour l'utilisation.

Les « Hard-cores » sont des IPs optimisés en surface et en vitesse. Ces derniers sont implémentés directement sur silicium et font appel à des composants élémentaires d'une librairie générique. Ces composants sont placés et routés entre eux.

3.4. Système embarqué sur les circuits FPGA (SoPC)

L'implantation se fait sur une plate-forme reconfigurable. On utilise des composants configurables tels que les FPGAs pour réaliser des systèmes sur des puces programmables (System on Programmable Chip) ou SoPC.

De nos jours, les composants FPGA haute densité et les IPs permettent l'intégration des systèmes complexes sur la même puce qui demande à être développées pour répondre aux besoins et aux nouvelles contraintes. Il est évident que les FPGAs sont aujourd'hui utilisés dans un large éventail d'applications. D'ailleurs, intégrer un système embarqué dans un circuit FPGA est la nouvelle approche pour profiter des avantages offerts par le FPGA tels que la souplesse ou la simplicité d'intégration [BUT 01].

En effet, la plupart des systèmes sur puce intègrent désormais un microprocesseur. Ce n'est pas une coïncidence si l'une des dernières nouveautés en matière d'IP sur FPGA est le microprocesseur 32 bits embarqué.

Puiseurs raisons justifient l'intégration de fonctionnalités de traitement embarquées sur un FPGA, bien que certaines soient moins évidentes que d'autres.

En premier lieu, il n'y a aucune implémentation d'architecture fixe et donc aucune obligation que les fonctions soient réalisées par matériel plutôt que par logiciel. Il existe un vaste choix de solutions envisageables pour chaque application.

En deuxième lieu, une fois que la conception est adaptée de manière optimale sur un circuit FPGA, il n'y a aucune raison pour conserver cette conception éternellement sur le circuit. Pour de nombreuses applications, on peut utiliser une plate-forme de prototypage plus générale pour les premières étapes de définition, avec l'intention de la remplacer un jour par une plate-forme moins onéreuse, plus performante ou d'un facteur de forme différent.

3.4.1 Processeurs embarquées sur FPGAs

Lorsque l'on conçoit un système numérique complexe, on met en œuvre généralement un processeur embarqué.

Ce processeur embarqué est :

- Soit un bloc IP et on parle ainsi de processeur soft-core.
- Soit il est implémenté sur silicium et on parle de processeur hard-core. Celui-ci est généralement plus performant que le processeur soft-core.

Le choix d'un processeur pour le SoPC peut se faire selon différents critères :

- Processeur *hard-core* : pour ses performances au détriment de la flexibilité.
- Processeur *soft-core* : pour sa flexibilité de mise à jour au détriment des performances moindres que le précédent.

Généralement, on privilégie les processeurs soft-core pour pouvoir bénéficier facilement des évolutions apportées en refaisant une synthèse. La description d'un processeur soft-core est souvent effectuée en utilisant un langage de description matérielle (VHDL, Verilog) et il est lié à un fondeur particulier de circuit FPGA (comme Altera ou Xilinx). De ce fait, son code source ne peut être implanté que dans les circuits FPGAs du fondeur approprié.

3.4.2 Système embarqué à base du processeur Microblaze

La conception et la réalisation des systèmes embarqués sur puce nécessitent un cycle de développement assez long. Pour cette raison, il est recommandé de procéder en premier lieu au développement d'un prototype dans lequel les circuits FPGAs sont souvent utilisés pour leur reconfigurabilité.

En effet, les fondeurs de ce type de circuits mettent généralement à la disposition des concepteurs, des cartes de prototypage à base de ces circuits. Ceci est dans le but de réaliser des implémentations en un cycle de temps relativement faible.

L'implémentation du processeur Microblaze en tant que contrôleur principal du SoPC sur le circuit FPGA, nécessite selon les besoins du concepteur, l'ajout sur le même circuit des périphériques qui jouent les rôles d'interfaces entre le processeur et les autres composants de la carte. Ses périphériques sont fournis par Xilinx sous forme d'IPs en langage VHDL. Ils seront connectés autour de Microblaze sur le bus système OPB. Parmi ces périphériques, on peut retrouver un UART (Universal Asynchronous Receiver/Transmitter), un contrôleur de mémoire externe (memCntlr), un Timer, un contrôleur d'interruption (INTC), un module pour

le débogage (MDM),...etc. Il est à noter que d'autres périphériques sont nécessaires au fonctionnement du processeur et qu'il est indispensable de les inclure. Ces derniers sont une mémoire BRAM et ses contrôleurs, DCM (Digital Clock Manager).

3.5. Flot de Conception d'un SoPC

Pour faciliter la conception de ces systèmes, plusieurs logiciels permettent la description et la simulation des circuits à différents niveaux d'abstraction comme l'outil EDK de Xilinx. De plus, la complexité accrue des systèmes a naturellement tendance à augmenter le temps de conception et donc l'automatisation de certaines phases permet de limiter le "time to market".

D'une manière générale, le flot de conception d'un système sur puce nécessite le développement de deux ressources à savoir, une ressource logicielle et une autre matérielle comme le montre la figure 3.2. La partie logicielle est utilisée pour rendre les applications à implémenter plus flexibles. La partie matérielle est développée pour accélérer les chemins sensibles à la contrainte temps réel des architectures associées aux applications.

Le partitionnement d'une application sur les deux ressources s'effectue généralement en tenant compte des performances à atteindre, c'est-à-dire la surface occupée, le temps d'exécution, la puissance consommée et le coût de développement.

En effet, la migration des tâches vers le matériel (HW) est toujours possible jusqu'à ce que les contraintes de performances soit atteintes, comme elles peuvent être basculées vers le logiciel (SW) [SCH 10].

La modélisation de la partie logicielle est réalisée par l'utilisation d'un langage de description machine en l'occurrence, le langage C, ou C++ ou encore l'assembleur et sera exécuté par un microprocesseur/microcontrôleur.

Pour la partie matérielle, celle-ci peut être décrite par un langage de description matérielle (HDL, VHDL,...).

Une fois l'étape de spécification de l'architecture du système est achevée, une synthèse logique est exécutée sur cette dernière. Cette étape n'est rien d'autre qu'une transformation de l'architecture en un ensemble d'équations booléennes. On parle ainsi du niveau Transfert Registre Level ou RTL.

Le résultat obtenu de la synthèse sous forme de netlist est optimisé et transformé en une autre netlist de blocs logiques qui sera placée/routée sur la plateforme utilisée.

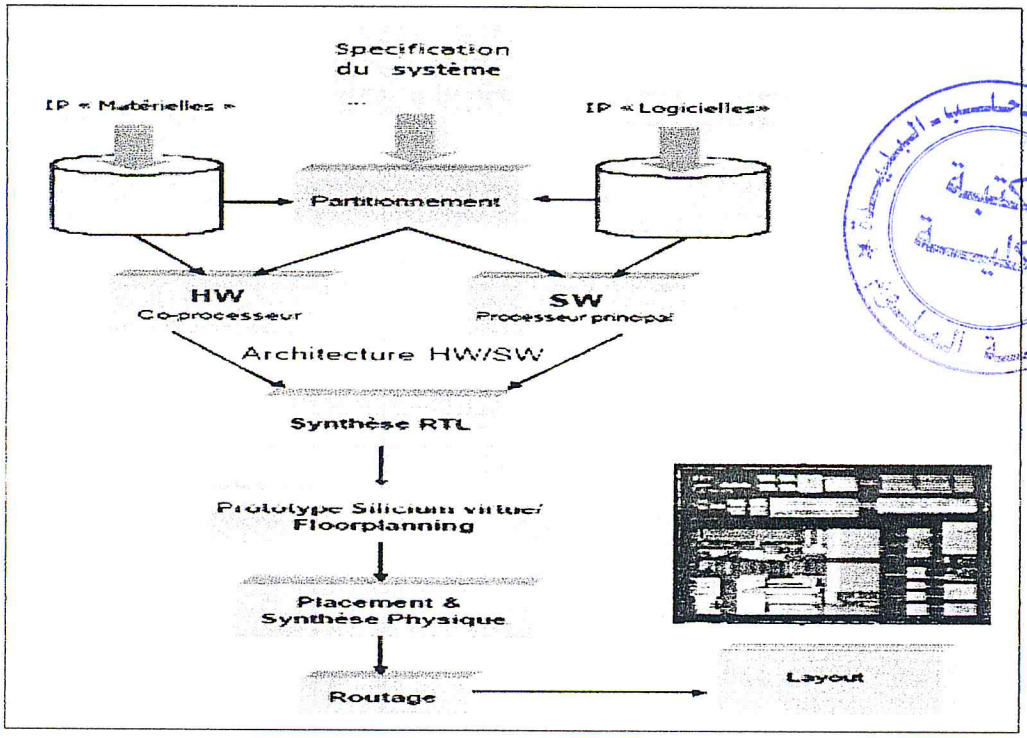


Figure 3.2. Flot de conception d'un système sur puce.

3.6. Conclusion

Dans ce chapitre, nous avons introduit des notions générales sur les systèmes embarqués et plus précisément les systèmes sur puce ou SoC puis SoPC.

D'un point de vue matériel, la réalisation d'un système sur puce, aujourd'hui, est plus accessible grâce au FPGA de plus en plus performant alliés aux blocs IPs réutilisables qui sont très répandus et qui permettent de maîtriser la complexité croissante des SoCs [RAB 00].

La conception sur circuit FPGA de notre crypto-système hybride RSA-AES sera détaillée dans le prochain chapitre, où l'implémentation du RSA, qui servira à chiffrer la clé secrète de l'AES, se fait par software par le langage C et l'implémentation de l'AES se fait par hardware par le langage VHDL.

Chapitre 04

Conception

4.1. Introduction

Après avoir donné un aperçu général sur les fonctionnements du RSA et de l'AES ainsi que les différentes implémentations matérielles de l'AES, nous présenterons dans ce chapitre trois architectures matérielles pour les trois cas d'exécution des rounds de l'AES: série-série, parallèle-série et parallèle-pipeline qui seront implémentés sur FPGA, puis nous présentons les implémentations software du RSA et de la génération des clés de l'AES par une application en C.

4.2. Architecture du crypto-système RSA-AES

La figure 4.1 montre l'architecture de notre crypto-système Hybride RSA-AES. Elle est constituée de cinq blocs (Entité1, AES, Génération des clés, RSA et Entité 2).

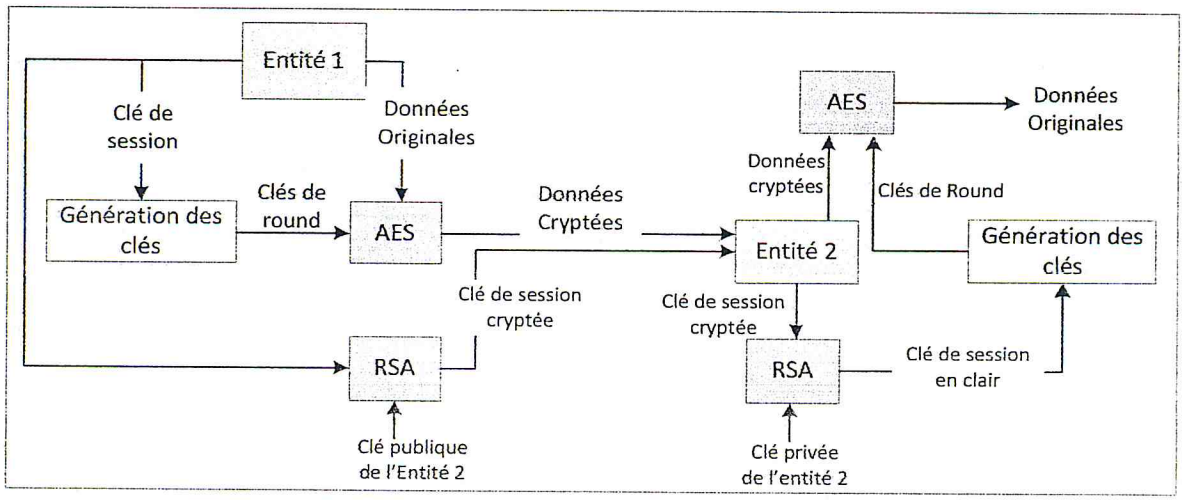


Figure 4.1 Architecture du système hybride RSA-AES.

Le fonctionnement de cette architecture, pour que l'**Entité1** envoie des messages à l'**Entité2** via un canal non sécurisé, se fait comme suit.

En premier lieu, **Entité1** choisit une clé de session pour crypter ses données par le bloc AES. Cette clé passe par le bloc Génération des clés, qui permet de générer des clés de rounds utilisées dans le chiffrement AES, et le bloc RSA afin d'être cryptée en utilisant la clé publique de l'**Entité2**. À la sortie, le bloc RSA fournit la clé de session cryptée et le bloc AES fournit les données cryptées. Enfin, l'**Entité1** envoie son message composé de la clé de session et des données cryptées par les deux blocs RSA et AES vers l'**Entité2** via un canal non sécurisé. À la réception, ce message crypté passe par les mêmes blocs AES, Génération des clés et RSA pour exécuter l'opération inverse de ces blocs.

D'abord, la clé de session passe par le bloc RSA afin d'être décryptée en utilisant la clé privée, ce dernier fournit en sortie la clé de session en clair. Ensuite, la clé de session passe par le bloc Génération des clés pour exécuter la même procédure de génération des clés et fournit en sortie des clés utilisées par le bloc AES afin de décrypter les données du message reçu. Enfin, ce dernier bloc fournit en sortie des données en clair sous forme des données originales.

4.3. Eléments de l'IP AES

Notre IP est basé sur l'AES qui opère sur des blocs de message de 128 bits avec une clé secrète pour chiffrer un texte en clair ou déchiffrer un texte crypté avec une masse de données. L'AES s'exécute en rounds où neuf de ces rounds incluent les quatre opérations (ShiftRows, SubByte, MixColumns et AddRoundKey) et le dixième round se caractérise par l'absence des opérations MixColumns et InvMixColumns pour les processus de chiffrement et de déchiffrement respectivement, plus un round initial qui exécute l'opération XOR entre le bloc de données de 128 bits et la clé de chiffrement qui est de même taille que les données.

Le schéma de l'architecture de l'AES est illustré sur la figure 4.2. Elle est constituée de quatre modules : un module principal de sécurité (AES_Core), un bloc Key_Ram pour le stockage des clés et un Controller. Toute la sécurité du système se base sur le module AES_Core. Ce dernier permet de crypter et décrypter les informations contenues dans chaque message reçu.

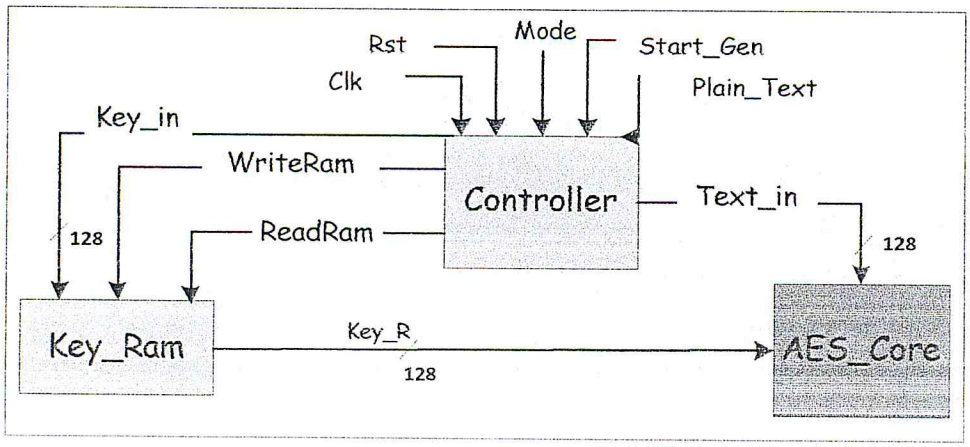


Figure 4.2. Architecture de l'IP AES.

4.4. Fonctionnement de l'AES_Core

Dans notre conception de l'IP-AES, les données à traiter et les clés des rounds seront stockées dans deux tables d'octets avec quatre lignes et quatre colonnes nommées State et RoundKeys respectivement représenté sur la figure 4.3.Le i^eme round d'un bloc de données de 128 bits sera défini par :

State^{i} = $S_{15}^{i}S_{14}^{i}S_{13}^{i}S_{12}^{i}S_{11}^{i}S_{10}^{i}S_{9}^{i}S_{8}^{i}S_{7}^{i}S_{6}^{i}S_{5}^{i}S_{4}^{i}S_{3}^{i}S_{2}^{i}S_{1}^{i}S_{0}^{i}$, où l'élément S_{15}^{i} représente l'octet le plus significatif dans le **State**^{i}.

$$\begin{pmatrix} S_{15}^{i} & S_{11}^{i} & S_{7}^{i} & S_{3}^{i} \\ S_{14}^{i} & S_{10}^{i} & S_{6}^{i} & S_{2}^{i} \\ S_{13}^{i} & S_{9}^{i} & S_{5}^{i} & S_{1}^{i} \\ S_{12}^{i} & S_{8}^{i} & S_{4}^{i} & S_{0}^{i} \end{pmatrix}$$

Figure 4.3. Représentation matricielle de State de chiffrement pour le i^eme round.

L'exécution d'un round dans l'AES a besoin de 16 Sboxs pour la transformation SubByte et quatre opérations de 32 bits pour la transformation MixColumns travaillant avec des données indépendantes, avec plusieurs possibilités du parallélisme et seulement la transformation ShiftRows qui a besoin de traiter le bloc de 128 en entier.

Nous avons opté pour le choix de l'architecture 32 bits pour des implémentations avec une exécution des rounds série-série, parallèle-série et parallèle-pipeline pour choisir la structure la plus fiable pour notre système.

1. Série-Série, où les blocs de 32 bits constituant le message d'un round sont exécutés en série et le tout en séquentiel.
2. Parallèle-Série, où les blocs de 32 bits constituant le message d'un round sont exécutés en parallèle et le tout en séquentiel.
3. Parallèle-Pipeline, où les blocs de 32 bits constituant le message d'un round sont exécutés en parallèle et le tout en pipeline.

Par la suite, nous allons détailler l'implémentation du processus de chiffrement et de déchiffrement de chacune de ces architectures.

Il est à noter que toutes les implémentations de l'architecture de l'AES sont avec une clé de 128 bits.

4.4.1.Processus de chiffrement

4.4.1.1. Exécution Série-Série

C'est une architecture séquentielle de l'AES où seulement 32 bits sont traités à la fois et les rounds sont exécutés séquentiellement sur le même matériel. Ceci permet de réduire la surface d'un facteur de quatre, mais augmente le temps nécessaire d'un round à **quatre cycles.**

Ce type d'architecture a pour objectif de garantir une optimisation dans les ressources d'implémentation avec un débit faible. Son implémentation se fait pour un seul round, suivi par une exécution itérative des données de 32 bits à travers ce round, jusqu'à chiffrer ou déchiffrer toutes les données. Le schéma de cette architecture est donné par la figure 4.4. Elle est constituée de trois modules : un module principal de sécurité (**AES_BLOC**), deux registres **Latch** de 128 bits chacun, deux **Aiguilleurs** de 128 bits chacun et une matrice **State** d'octets de **4×4**.Toute la sécurité du système se base sur le module **AES_BLOC**. Ce dernier permet de crypter les informations contenues dans chaque bloc reçu par l'exécution des différentes opérations qu'il inclue (SUB_BYTE, MixColumns et AddRK).

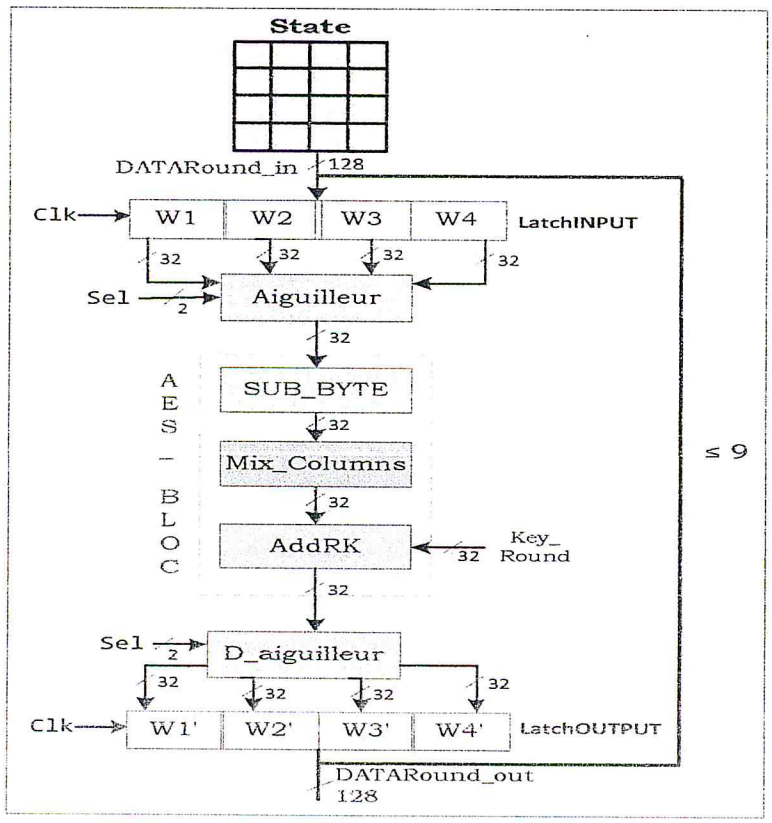


Figure 4.4. Exécution Série-Série pour le fonctionnement des rounds 1 à 9.

Dans cette conception, la manière d'accéder aux données de State^{i} est procédée par la diagonale suivant le principe de l'opération ShiftRows comme il est représenté par la figure 4.3. On commence à lire le premier quart du round avec le Word (W1) qui est constitué des quatre octets suivants: $S_{15}^{i}, S_{10}^{i}, S_{5}^{i}$ *et* S_{0}^{i}en parallèle, puis on exécute les trois transformations (SUB_BYTE,MixColumns et AddRK) comme suit :

- **SUB_BYTE (W1)** = (SUB_BYTE (S_{15}^{i}), SUB_BYTE (S_{10}^{i}), SUB_BYTEe (S_{5}^{i}), SUB_BYTE (S_{0}^{i})).
- **MixColumns (W1)** = SUB_BYTE (**W1**) × M(x) =

$$\textbf{State}^{i+1} = \begin{pmatrix} S_{15}^{i+1} & S_{11}^{i+1} & S_{7}^{i+1} & S_{3}^{i+1} \\ S_{14}^{i+1} & S_{10}^{i+1} & S_{6}^{i+1} & S_{2}^{+1i} \\ S_{13}^{i+1} & S_{9}^{+1i} & S_{5}^{i+1} & S_{1}^{i+1} \\ S_{12}^{i+1} & S_{8}^{i+1} & S_{4}^{i+1} & S_{0}^{i+1} \end{pmatrix}$$

Figure 4.5. Résultat intermédiaire de chiffrement de l'i^{eme} round.

$$\begin{pmatrix} SUB\_BYTE\ (S_{15}^{i}) \\ SUB\_BYTE(S_{10}^{i}) \\ SUB\_BYTE\ (S_{5}^{i}) \\ SUB\_BYTE\ (S_{0}^{i}) \end{pmatrix} \times \begin{pmatrix} \textbf{'02'} & \textbf{'03'} & \textbf{'01'} & \textbf{'01'} \\ \text{'01'} & \text{'03'} & \text{'02'} & \text{'01'} \\ \text{'01'} & \text{'01'} & \text{'02'} & \text{'03'} \\ \text{'03'} & \text{'02'} & \text{'01'} & \text{'01'} \end{pmatrix}$$

- **AddRK (W1)** = MixColumns (W1) **XOR**RoundKey^{i}comme suit:

$$\text{AddRK(W1)} = \begin{pmatrix} MixColumns\ (S_{15}^{i}) \\ MixColumns(S_{10}^{i}) \\ MixColumns\ (S_{5}^{i}) \\ MixColumns\ (S_{0}^{i}) \end{pmatrix} \oplus \begin{pmatrix} Rk_{15}^{i} & Rk_{11}^{i} & Rk_{7}^{i} & Rk_{3}^{i} \\ Rk_{14}^{i} & Rk_{10}^{i} & Rk_{6}^{i} & Rk_{2}^{i} \\ Rk_{13}^{i} & Rk_{9}^{i} & Rk_{5}^{i} & Rk_{1}^{i} \\ Rk_{12}^{i} & Rk_{8}^{i} & Rk_{4}^{i} & Rk_{0}^{i} \end{pmatrix}$$

Puis, on réécrit les résultats dans la matrice **State**$^{i+1}$ en word

W1'= $S_{15}^{i+1}, S_{14}^{i+1}, S_{13}^{i+1}$ *et* S_{12}^{i+1}comme le représente la figure 4.5.

Remarque

Les trois autres words (W2, W3 et W4) sont traités de la même manière que W1 avec W2 =(S_{11}^{i}, S_{6}^{i}, S_{1}^{i} *et* S_{12}^{i}), W3 = (S_{7}^{i}, S_{2}^{i}, S_{13}^{i} *et* S_{8}^{i}) et W4 = (S_{3}^{i+1}, $S_{14}^{i+1}, S_{9}^{i+1}$ *et* S_{4}^{i+1}) suivant le principe de l'opération ShiftRow.

Le tableau 4.1, décrit les signaux d'entrées/sorties de cette architecture.

Signal	Description
Clk	Horloge du système.
Sel[0. .1]	Commande de sélection sur 2 bits indiquant les données entrantes.
Key_Round[0..31]	Clé de round de 32 bits.
W1[0..31]	Données sur 32 bits.
W2[0..31]	Données sur 32 bits.
W3[0..31]	Données sur 32 bits.
W4[0..31]	Données sur 32 bits.
W1'[0..31]	Données sur 32 bits.
W2'[0..31]	Données sur 32 bits.
W3'[0..31]	Données sur 32 bits.
W4'[0..31]	Données sur 32 bits.
Data_Round_in[0..127]	Texte à crypter/décrypter entrant sur 128 bis.
Data_Round_out[0..127]	Texte à décrypter/crypter sortant sur 128 bis.
State	Matrice carrée d'octets de **4×4,** représentant les résultats intermédiaires.

Tableau 4.1. Description des signaux d'E/S de l'architecture Série-série.

Dans ce qui suit, nous décrirons chacun des modules de l'architecture Série-série.

- **LatchINPUT** : ce composant est responsable de la surveillance de l'opération de synchronisation du cryptage/décryptage. Il fait une mémorisation temporaire des données de 128 bits à partir de la source State, et les transforme à travers le bus de communication de l'architecture en quatre Words de 32 bits (W1, W2, W3 et W4).Le fonctionnement de ce composant est géré par un processus et activé par le signal *Clk*. Pour chaque front montant d'horloge *Clk*, il y a une lecture d'un word des données (W1 ,W2 ,W3ouW4). Après 1 cycle d'horloge, la transmission de données est achevée.

- **LatchOUTPUT** : ce composant effectue la même opération que le module LatchINPUT. Pour chaque front montant d'horloge *Clk*, il y a une lecture des quatre words de 32 bits reçus à partir du module D_aiguilleur. Après un cycle d'horloge, la transmission de données est achevée.

- **Aiguilleur** : ce composant sélectionne un word Wi parmi les quatre du LatchInput. Cet aiguillage se fait selon une commande de sélection Sel de deux bits. Le fonctionnement

de cet aiguilleur est géré par un processus et prend en entrée quatre **Words** (W1, W2, W3et W4) où chacun est codé sur 32 bits et fournit en sortie un bloc OUTPUT codé sur 32 bits sélectionné selon la valeur du signal *Sel*(codé sur 2 bits).

Pour déclencher le processus de sélection des words, on doit effectuer un test sur le signal *Sel*, tel que : si *Sel* = 00 alors le Word (W1)est sélectionné, si *Sel* = 01 c'est le Word (W2) qui est sélectionné, si *Sel* = 10 c'est le Word (W3) qui est sélectionné et enfin si *Sel* =11 c'est le dernier Word (W4) qui est sélectionné.

Pour chaque front montant d'horloge *Clk*, il y a une lecture des quatre words de 32 bits. Après 1 cycle d'horloge, le regroupement des words des données en un bloc de 128 bits est achevé et suivant la valeur de signal *Sel* les données sont transmises.

- **D_aiguilleur** : ce composant permet d'effectuer la transformation inverse du module Aiguilleur. En effet, il segmente un bloc de données de 128 bits, arrivées en quatre itérations à partir de module AES_BLOC en quatre words de données de 32 bits chacun afin de les transmettre vers le module destination LatchOutput. Si le signal *Clk* est actif à la valeur 1, il y'a une lecture d'un bloc de données de 128 bits. Dans ce cas ces données seront subdivisées en quatre words de 32 bits chacun. En plus, il a besoin d'un signal *Sel* codé sur 2 bits pour sélectionner les words (Wi pour i = 1 à 4) fournis en sortie.
- **AES_BLOC** :c'est le module principal de l'IP de sécurité. Ce module se base sur le crypto-système symétrique de l'algorithme de chiffrement AES. Son architecture est constituée des trois transformations (SUB_BYTE, MixColumns et AddRK) représenté par les composants suivants :

a. **Composant AddRK** : c'est le composant qui réalise l'opération AddRoundKey dans l'AES, il a besoin d'un bloc de données et d'une clé de chiffrement comme où les données à chiffrer et la clé de chiffrement sont codées sur 32 bits.

b. **Composant SUB_BYTE** : c'est le composant qui réalise l'opération SubByte dans l'AES. Cette transformation a été implémentée par l'utilisation de quatre Sboxs enregistrées dans des blocs Rams avec (256×8×4 = **8 Kbits**) pour substituer le bloc de données de 32 bits.

c. **Composant MixColumns** : c'est le composant qui réalise l'opération MixColumns de l'AES. Cette transformation est implémenté en utilisant la fonction Xtime La figure 4.6représente l'architecture réalisant cette opération.

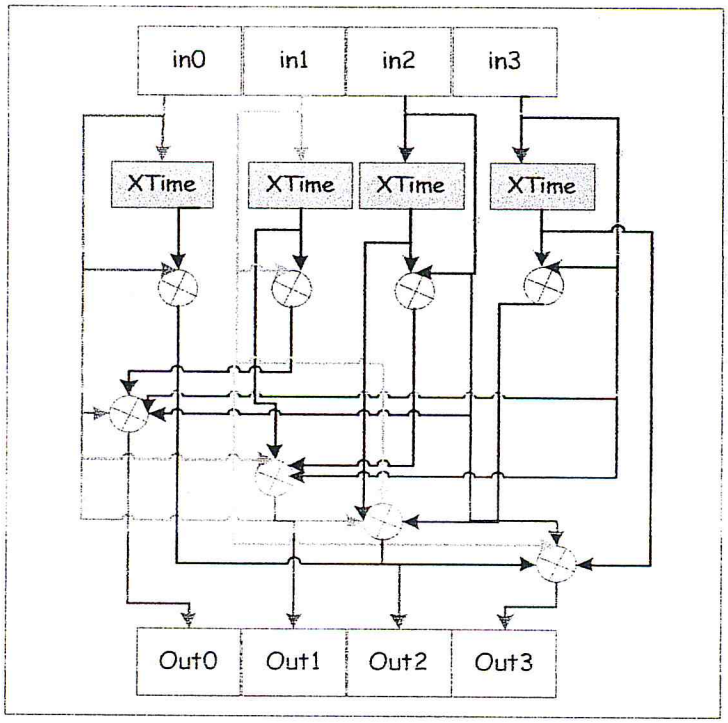


Figure 4.6. Architecture de l'opération MixColumns avec la fonction XTime.

✓ Dans cette architecture, l'implémentation d'une colonne nécessite quatre portes Xors pour le calcul de la fonction Xtime plus quatre portes XORs pour le calcul des éléments d'une colonne. (4× 8 + 4) = **36 Portes XORs**.

Le tableau 4.2 décrit les caractéristiques d'implémentation d'un message de 128 bits avec une architecture Série-série pour un round et pour 10 rounds.

Caractéristiques / Nbre de round	**SBoxs utilisées**	**Nombre d'itérations**	**Temps nécessaire**
Un Round	4 Sboxs (8 K bits)	4 itérations.	**4 Tops d'horloge**
10 Rounds	4 Sboxs (8 K bits)	4×10=**40 itérations**	4×10 = **40 Tops d'horloge**

Tableau 4.2. Caractéristiques d'implémentation de l'architecture Série-Série.

Remarque : dans cette architecture Série-Série, l'implémentation de l'opération ShiftRows ne nécessite aucun matériel et elle est implémentée seulement avec un décalage câblé.

➢ D'après le tableau 4.2, cette architecture est caractérisée par **un débit très faible avec un temps = 40 Tops d'Horloge** pour chiffrer seulement un message de 128 bits, avec peu de ressources (8 Kbits et 36 portes XORs).

➢ Ce type d'architecture n'est pas supporté pour les applications qui nécessitent un débit de traitement élevé comme on le souhaite.

Pour améliorer le temps d'exécution nous avons proposé l'architecture suivante.

4.4.1.2. Exécution Parallèle-Série

Le parallélisme offert dans l'exécution d'un round dans l'AES permet d'améliorer le temps d'exécution de l'architecture Série-Série qui sera remplacée par l'architecture Parallèle-Série représenté sur la figure 4.7.

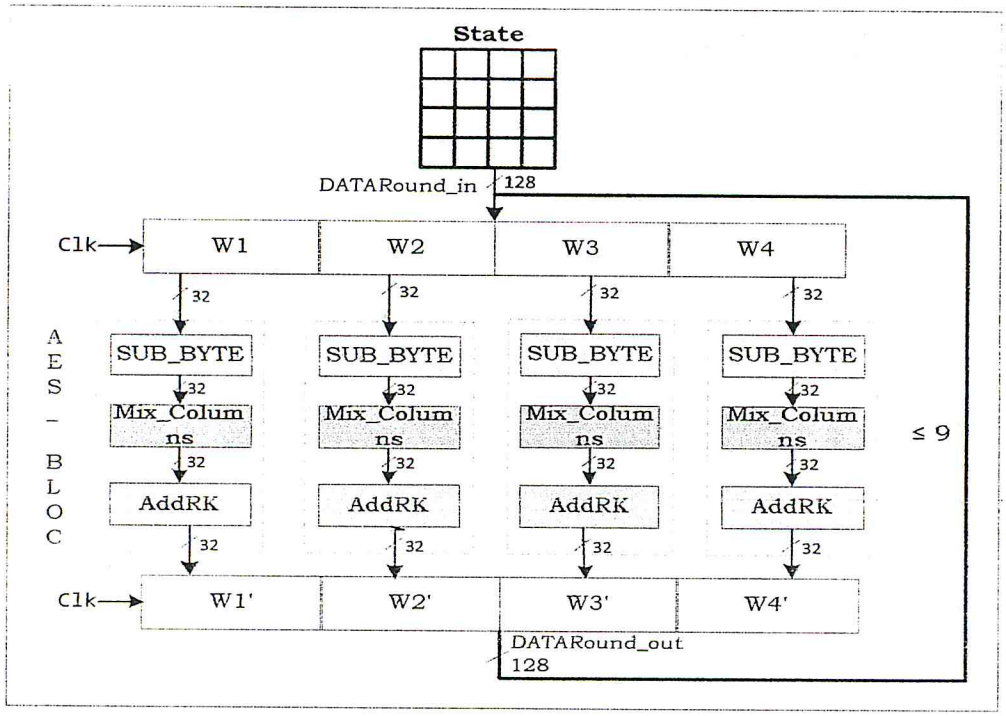


Figure 4.7. Architecture Parallèle-Série pour le fonctionnement des rounds 1 à 9.

Cette architecture a les mêmes composants que l'architecture Série-Série avec l'absence des deux composants Aiguilleur et D_aiguilleuret un dédoublement par quatre du module AES_BLOC. C'est une architecture séquentielle avec un parallélisme dans les blocs de 32 bits, dont le but est de diminuer le nombre d'itérations pour augmenter le débit.

Dans cette architecture l'implémentation d'un round pour la transformation SubByte requiert 16 Sboxs et quatre opérations de 32 bits pour l'implémentation de la transformation MixColumns avec **144 Ports XORs**.

Le tableau 4.3décrit les caractéristiques de cette architecture pour l'exécution d'un round et de 10 rounds

Caractéristiques / Nbre de round	**SBoxs utilisées**	**Nombre d'itérations**	**Temps nécessaire**
Un Round	16 Sboxs(+32 Kbits)	Une itération	1 Top d'horloge.
10 Rounds	16 Sboxs(+32K bits)	1×10 =10 itérations	1×10 =**10 Tops d'horloge**

Tableau 4.3. Caractéristiques de l'implémentation de l'architecture Parallèle-Série.

➢ Dans cette architecture le nombre d'itérations est réduit **par quatre**, alors que les ressources sont **multipliées par quatre** avec un **débit moyen multiplié par** quatre par rapport à l'architecture Série-Série.

Pour améliorer d'avantage les temps et débit de traitement, d'autres modifications peuvent être apportées à cette architecture en introduisant le pipelining.

4.4.1.3. Exécution Parallèle-Pipeline

L'architecture Parallèle-Pipeline permet aux dix blocs de données de 128 bits d'être traités en même temps, ce qui donne un débit de cryptage/décryptage plus élevée. Le schéma de cette architecture est représenté sur la figure 4.8.

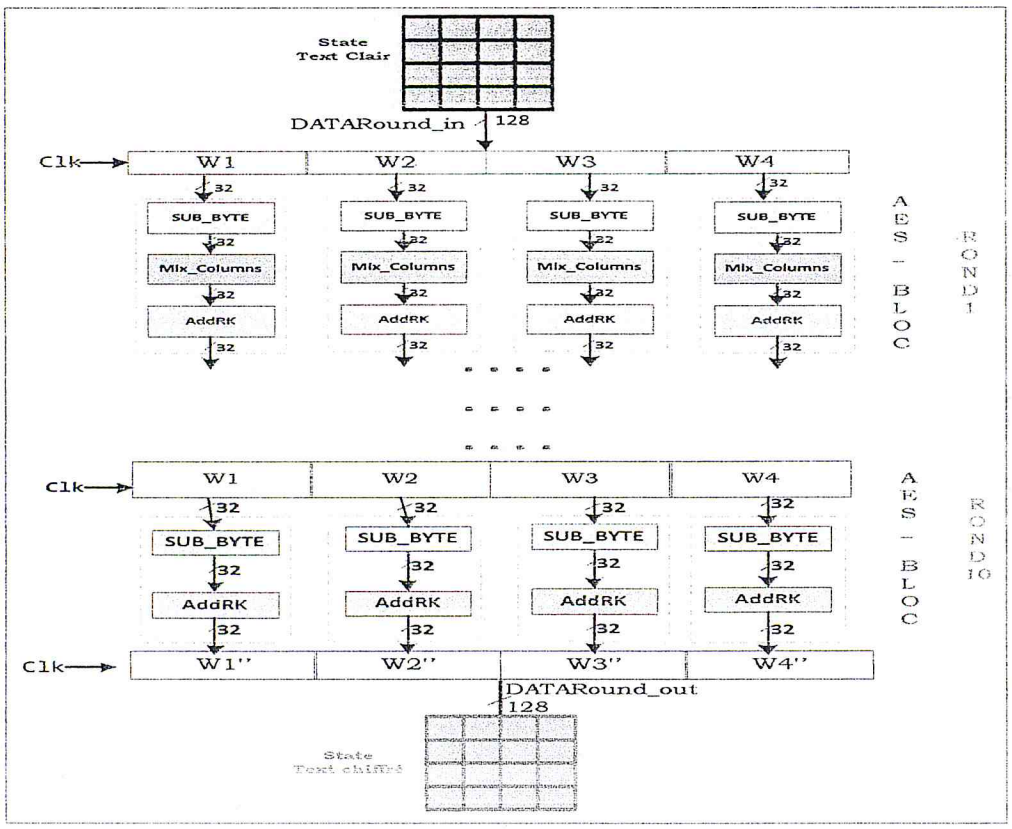


Figure 4.8. Exécution Parallèle-Pipeline pour le fonctionnement des rounds 1 à 9.

Le pipeline est construit par l'insertion des lignes de registres de 128 bits entre les blocs AES_BLOC. Les tours entre deux registres consécutifs forment une structure en pipeline

d'un seul étage. A chaque cycle d'horloge les données traitées passent à l'étape suivante et sa position est occupée par le bloc de données suivant.

Le tableau 4.4 décrit les caractéristiques d'implémentation d'un message de 128 bits avec cette architecture pour un round et pour 10 rounds.

Caractéristiques / Nbre de round	**SBoxs utilisées**	**Nombre d'itérations**	**Temps nécessaire**
Un Round	16 Sboxs(+32 K bits).	Une itération.	1 Top d'horloge.
10 Rounds	16 ×10 Sboxs(+327K bits).	1×10 = 10 itérations	1 × 10 = **10 Tops d'horloge.**

Tableau 4.4. Caractéristiques d'implémentation de l'architecture Parallèle-Pipeline.

- Dans cette architecture il y'a une multiplication du matériel **par 40** par rapport à l'architecture Série-Série et **par 10** par rapport à l'architecture parallèle-série.
- Latence = 10 tops d'horloge.
- Ce type d'architecture est spécifié pour les applications à fort débit, pour cela la plupart des applications utilisent ce mode d'architecture.

Vu les ressources énormes occupées par cette architecture lors de l'implémentation de l'opération SubByte avec des valeurs pré-calculées qui ont été stockées dans une table de consultation basée sur une ROM, mais cette méthode reste coûteuse en termes de matériel.

4.4.1.3.1. Méthodologie de construction de S-Box

Cette section illustre les étapes pour la construction de module d'inverse multiplicatif pour la S-Box en utilisant l'arithmétique dans le corps composé Galois Field. La manière est de calculer l'inverse multiplicatif dans GF (2^8) d'abord, puis exécuter la transformation affine pour le calcul des valeurs de la table S-Box et de calculer l'inverse de la transformation affine suivie par le calcul de l'inverse multiplicative dans le corps GF (2^8) pour le calcul de la ($S\text{-}Box^{-1}$).Un élément dans le GF(2^8) peut être représenté par bx+ c, où b représente la moitié la plus significative et c représente la moitié la moins significative. De là, le calcul de l'inverse multiplicative peut être donné par l'utilisation de l'équation suivante :

$$(bx + c)^{-1} = b(b^2B + bcA + c^2)^{-1} x + (c + bA)(b^2B + bcA + c^2)^{-1}.$$

De [JAG 09], le polynôme sélectionné est donné par :$x2 + x + \lambda$. *Donc A= 1 et B= λ*, d'où l'équation précédente de forme simplifiée est:

$$(bx + c)^{-1} = b\,(b^2\lambda + c(b+c))^{-1} x + (c + b)(b^2\lambda + c(b + c))^{-1}.$$

A partir de cette équation les opérations indiquées sont la multiplication, l'addition, le calcul du carré et le calcul de l'inverse multiplicatif dans le corps Galois Field GF(2^4). Chacune de ces opérations peut être transformée en des blocs individuels lors de la construction du circuit pour le calcul de l'inverse multiplicatif. A partir de l'équation simplifiée, l'architecture de la S-Box à base cette équation est donnée par la figure 4.9 :

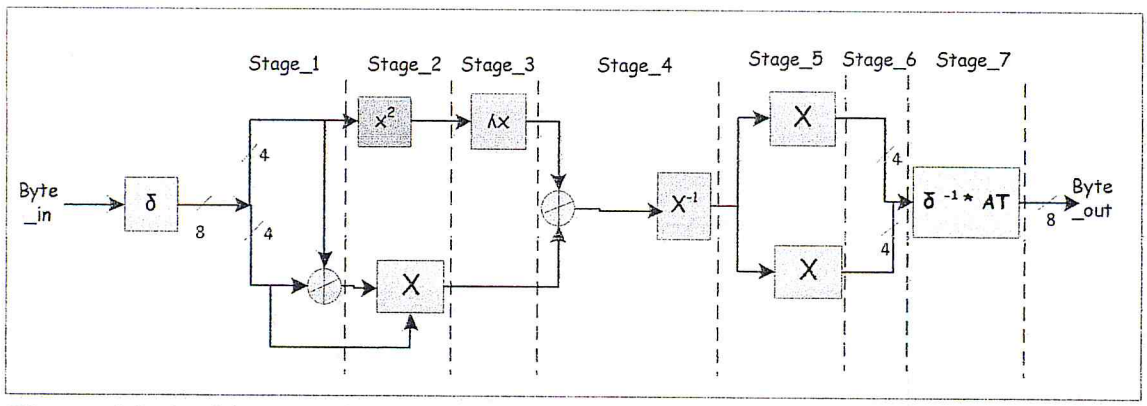


Figure 4.9. Architecture de la table S-Box avec l'utilisation de l'arithmétique dans le corps GF (2^8) avec 7 étages pipeline.

Les légendes des blocs de ce circuit peuvent être données par le tableau 4.5.

Légende	Description
δ	Mappage isomorphe dans le corps GF (2^8).
x^2	Le carré dans le corps GF (2^4).
⊕	L'addition dans le corps GF (2^4).
X	La multiplication dans le corps GF (2^4).
λx	La multiplication par la constante λ dans le corps GF (2^4).
X^{-1}	L'inverse multiplicatif dans le corps GF (2^4).
$\delta^{-1} * AT$	L'inverse du mappage isomorphe dans le corps GF (2^8) suivie par la transformation affine (AT).

Tableau 4.5. Légendes des blocs du circuit de la table S-Box par l'utilisation de l'arithmétique dans le corps GF (2^8).

a). Mappage isomorphe et son inverse

Le calcul de l'inverse multiplicatif se fait en décomposant le corps composé $GF(2^8)$ à des corps moins complexes d'ordre $GF(2^1)$, $GF(2^2)$ et $GF((2)^2)^2)$.Pour effectuer ces calculs, les polynômes irréductibles suivants sont employés :

1- $GF(2^2) \rightarrow GF(2)$: x^2+x+1.

2- $GF((2^2)^2) \rightarrow GF((2)^2)$: $x^2+x+\varphi$. Avec : $\varphi = \{10\}_2$.

3- $GF(((2^2)^2)^2) \rightarrow GF(((2)^2)^2)$: $x^2+x+\lambda$. Avec : $\lambda = \{1100\}_2$.

Le calcul de l'inverse multiplicatif dans les corps composés ne peut être directement appliqué à un élément qui est basé sur $GF(2^8)$.Cet élément doit être mappé de sa représentation dans le corps composé par l'intermédiaire d'une fonction isomorphe δ. De même, après l'exécution de l'inverse multiplicatif, le résultat devra également être mappé sa représentation dans le corps composé à son équivalent dans $GF(2^8)$ par l'intermédiaire de la fonction inverse δ^{-1}.δ et δ^{-1}peuvent être représentées comme une matrice de 8×8.

Soit *q un* élément dans $GF(2^8)$, d'où le mappage et son inverse peuvent être donnés par $\delta \times q$ et $\delta^{-1} \times q$ respectivement, qui est un cas d'une multiplication matricielle comme montré ci-dessous, où $q_7$représente le bit le plus significatif et $q_0$est le bit moins significatif.

$$\delta \times q = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} q_7 \\ q_6 \\ q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix} \qquad \delta^{-1} \times q = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} q_7 \\ q_6 \\ q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix}$$

Cette multiplication matricielle peut être transformée en une opération logique avec des portes XORs, comme indiqué dans la représentation suivante :

$$\delta \times q = \begin{bmatrix} q_7 \oplus q_6 \\ q_7 \oplus q_6 \oplus q_5 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_0 \\ q_7 \oplus q_5 \oplus q_4 \oplus q_2 \\ q_7 \oplus q_6 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_7 \oplus q_6 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_6 \oplus q_5 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_6 \oplus q_3 \oplus q_2 \\ q_5 \oplus q_4 \oplus q_1 \oplus q_0 \end{bmatrix} \qquad \delta^{-1} \times q = \begin{bmatrix} q_7 \oplus q_6 \oplus q_5 \oplus q_1 \\ q_6 \oplus q_2 \\ q_5 \oplus q_4 \oplus q_1 \\ q_6 \oplus q_5 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_7 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_7 \oplus q_6 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_5 \oplus q_4 \\ q_6 \oplus q_5 \oplus q_4 \oplus q_2 \oplus q_0 \end{bmatrix}$$

b). Opérations Arithmétiques du Corps Composé

Aussi, de [ISS 10] et [CHS 08], un nombre binaire q dans le corps Galois Fields peut être représenté par :$q_h x + q_l$,où q_h et q_l peuvent aussi être décomposés. Suivant cette idée les équations logiques pour l'addition, la multiplication et l'inverse peuvent être dérivées en :

b.1. L'Addition dans le GF(2^4)

L'addition de deux éléments dans le corps GF (2^4) peut être transformée en des simples opérations XORs entre ces deux éléments.

b.2. Carré dans GF (2^4)

Soit $k = q^2$, où k et q sont des éléments dans le corps GF (2^4) représentés par les nombres binaires :$\{k_3k_2k_1k_0\}_2$ et $\{q_3q_2q_1q_0\}_2$respectivement.

$K = k_h x + k_l = (q_h x + q_l)^2$. Avec $k_h = k_3k_2$ et $k_l = k_1k_0$.

Dans cette équation, la réduction dex^2se fait par l'utilisation d'un polynôme irréductible :$x^2+x+\varphi$.

Avec l'utilisation de $x^2 = x + \varphi$, la nouvelle expression serait : $k = q_h^2 + (x + \varphi) + q_l^2$.

Cette expression sera aussi décomposée de GF (2^2) en GF (2). La décomposition de q_h et de q_l dans le GF (2) est la suivante :

$k_h = q_h^2 = (q_3q_2)^2 = (q_3x + q_2)^2$.

Avec l'utilisation d'un polynôme irréductible :$x^2 + x + 1$.La nouvelle expression de la précédente est donnée par : $k_h = q_3(x + 1) + q_2$... (1).

De la même manière, ql est décomposé en : $Kl = q_3x^3 + q_2x + q_1x^2 + q_0$..... (2).

De l'équation (1) et (2), la formule pour calculer le carré dans le GF (2^4) est donnée par :

$$\begin{cases} K_3 = q_3. \\ K_2 = q_3 \oplus q_2. \\ K_1 = q_2 \oplus q_1. \\ K_0 = q_3 \oplus q_1 \oplus q_0. \end{cases}$$

La représentation matérielle de cette équation est indiquée par la figure 4.10

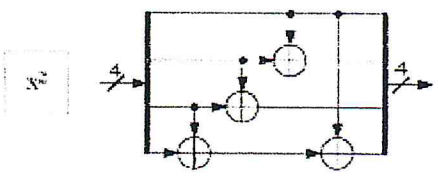


Figure 4.10. Architecture matérielle du carré dans le corps GF (2^4).

b.3. Multiplication par la constante λ

Soit $k = q\lambda$, où $k = \{k_3k_2k_1k_0\}_2$, $q = \{q_3q_2q_1q_0\}_2$ et $\lambda = \{1100\}_2$ sont des éléments dans le corps GF(2^4).$K = k_hx + k_l = (q_hx + q_l)(1100)$.

Après la simplification de cette équation on aura : $K = (q_hx + q_l)(\lambda_hx + \lambda_l)$, λ_l peut être éliminé car :$\lambda_l = \{00\}_2$, d'où $k = q_h\lambda_h + q_l\lambda_hx$.

Avec l'utilisation d'un polynôme, x2 = x + φ, la nouvelle expression est donnée par :

$$\begin{cases} K_3 = q_2 \oplus q_0. \\ K_2 = q_3 \oplus q_2 \oplus q_1 \oplus q_0. \\ K_1 = q_3. \\ K_0 = q_2. \end{cases}$$

Cette équation est transformée en une représentation matérielle donnée par la figure 4.11:

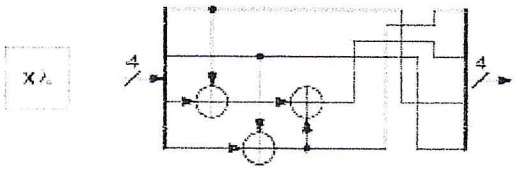


Figure 4.11. Architecture matérielle de la multiplication par λ.

b.4. Multiplication dans GF (2^4)

Soit $k = qw$, où $k = \{k_3k_2k_1k_0\}_2$, $q = \{q_3q_2q_1q_0\}_2$ et $w = \{w_3w_2w_1w_0\}_2$ sont des éléments dans le corps GF(2^4).

$K = k_hx + k_l = (q_hx + q_l)(w_hx + w_l)$.

Après la simplification de cette équation, la substitution de x^2 par $x + \varphi$ donne la nouvelle expression suivante : $K = k_hx + k_l = (q_hw_h + q_hw_l + q_lw_h)x + k_hw_h\varphi + k_lw_l$, (GF (2^2).

Cette équation est donnée dans le GF (2^2). Il est remarquable que les opérations existantes soient l'addition et la multiplication dans le GF (2^2). La multiplication dans le GF (2^2) nécessite une décomposition vers le corps GF (2) pour une implémentation matérielle. En outre, l'expression serait trop complexe si cette équation devait être décomposée en GF(2). La représentation matérielle de cette multiplication est donnée par la figure 4.12

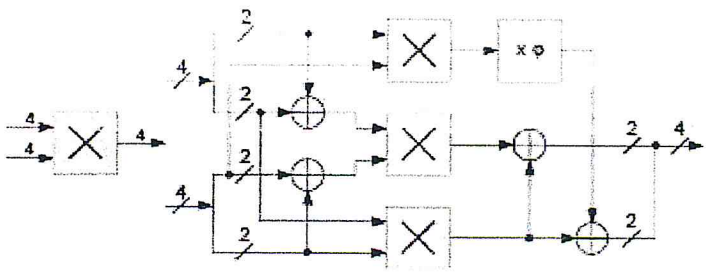


Figure 4.12. Architecture matérielle de la multiplication dans le corps GF (2^4).

b.5. Multiplication dans le GF (2^2)

Soit $k = qw$, où $k = \{k_1k_0\}_2$, $q = \{q_1q_0\}_2$ et $w = \{w_1w_0\}_2$ sont des éléments dans GF(2^2).

Après la simplification de cette équation et la substitution de x^2 par x+1, l'expression finale de cette multiplication sera donnée par :

$$\begin{cases} K_1 = q_1w_1 \oplus q_0w_1 \oplus_1 w_0. \\ K_0 = q_1w_1 \oplus q_0w_0. \end{cases}$$

La représentation matérielle de cette expression est donnée par la figure4.13:

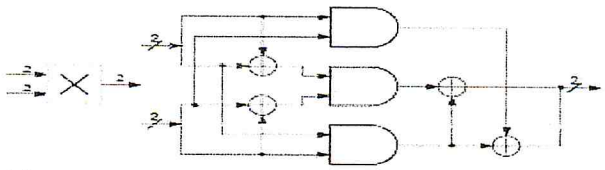


Figure 4.13. Architecture matérielle de la multiplication dans le corps GF (2).

b.6. Multiplication par la constante φ

Soit $k = q\varphi$, où $k = \{k_1k_0\}_2$, $q = \{q_1q_0\}_2$ et $\varphi = \{10\}_2$ sont des éléments dans le GF(2^2).

Après la simplification de cette équation et la substitution de x^2 par x + 1, l'expression finale est donnée par :

$$\begin{cases} K_1 = q_1 \oplus q_0. \\ K_0 = q_1. \end{cases}$$

Ce qui est traduit par la représentation matérielle donnée par la figure 4.14

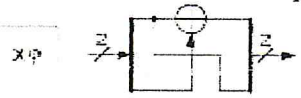


Figure 4.14. Architecture matérielle de la multiplication par φ.

b.7. L'inverse multiplicatif dans le GF (2^4)

A partir de l'article [NAI 08], la formule de calcul de l'inverse multiplicative est donnée par :

$$\begin{cases} q_3^{-1} = q_3 \oplus q_3q_2q_1 \oplus q_3q_0 \oplus q_2. \\ q_2^{-1} = q_3q_2q_1 \oplus q_3q_2q_0 \oplus q_3q_0 \oplus q_2 \oplus q_2q_1 \\ q_1^{-1} = q_3 \oplus q_3q_2q_1 \oplus q_3q_1q_0 \oplus q_2 \oplus q_2q_0 \oplus q_1. \\ q_0^{-1} = q_3q_2q_1 \oplus q_3q_2q_0 \oplus q_3q_1 \oplus q_3q_1q_0 \oplus q_3q_0 \oplus q_2 \oplus q_2q_1 \oplus q_2q_1q_0 \oplus q_1 \oplus q_0. \end{cases}$$

Les résultats de l'inverse multiplicatif sont donnés par le tableau suivant :

Q	0	1	2	3	4	5	6	7	8	9	a	b	c	D	e	f
q^{-1}	0	1	3	2	f	c	9	B	a	6	8	7	5	E	d	4

Tableau 4.6. Résultats pré-calculés de l'opération inverse multiplicative dans le GF (2^4).

4.4.2. Processus de déchiffrement

4.4.2.1. Exécution Série_Série

Cette architecture a les mêmes composants que l'architecture Série-Série pour le chiffrement. Dans le déchiffrement, le module AES_BLOC comporte des opérations inverses, telles qu'Inv_SUB_BYTE au lieu de SUB_BYTE, Inv_MixColumns au lieu de MixColumns et l'opération **AddRK**a le même principe que dans le chiffrement mais avec un autre ordre d'utilisation des clés pour l'opération AddRK.

Dans notre conception et comme dans le chiffrement, la manière d'accéder aux données de **State**^{i} se fait par la diagonale comme il est représenté par la figure 4.15, suivant le principe de l'opération **Inv_ShiftRows**.

$$\begin{pmatrix} \boldsymbol{S_{15}^i} & S_{11}^i & S_7^i & S_3^i \\ S_{14}^i & S_{10}^i & S_6^i & \boldsymbol{S_2^i} \\ S_{13}^i & S_9^i & \boldsymbol{S_5^i} & S_1^i \\ S_{12}^i & \boldsymbol{S_8^i} & S_4^i & S_0^i \end{pmatrix}$$

Figure 4.15 : Représentation matricielle de State de déchiffrement pour le i^{eme} rond.

On commence à lire le premier quart du round avec le Word (W1) qui est constitué des quatre octets suivants: $\boldsymbol{S_{15}^i}$,$\boldsymbol{S_2^i}$, $\boldsymbol{S_5^i}$ *et* $\boldsymbol{S_8^i}$en parallèle, puis on exécute les trois opérations (Inv_SUB_BYTE, Inv_MixColumns et AddRK) comme suit :

- **Inv_SUB_BYTE (W1)** = (Inv_SUB_BYTE ($\boldsymbol{S_{15}^i}$), Inv_SUB_BYTE ($\boldsymbol{S_2^i}$), Inv_SUB_BYTE ($\boldsymbol{S_5^i}$), Inv_SUB_BYTE ($\boldsymbol{S_8^i}$)).
- **Inv_MixColumns (W1)** = Inv_SUB_BYTE (**W1**) * Inv_M(x) =

$$\begin{pmatrix} \textbf{Inv\_SUB\_BYTE } (\boldsymbol{S_{15}^i}) \\ \textbf{Inv\_SUB\_BYTE}(\boldsymbol{S_2^i}) \\ \textbf{Inv\_SUB\_BYTE } (\boldsymbol{S_5^i}) \\ \textbf{Inv\_SUB\_BYTE}(\boldsymbol{S_8^i}) \end{pmatrix} * \begin{pmatrix} \textbf{'0e'} & \textbf{'0b'} & \textbf{'0d'} & \textbf{'09'} \\ \text{'09'} & \text{'0e'} & \text{'0b'} & \text{'0d'} \\ \text{'0d'} & \text{'09'} & \text{'0e'} & \text{'0b'} \\ \text{'0b'} & \text{'0d'} & \text{'09'} & \text{'0e'} \end{pmatrix}$$

- **AddRK (W1)** = Inv_MixColumns (W1) **Xor** AddRK^{11-i} comme suit:

$$\text{AddRK (W1)} = \begin{pmatrix} \boldsymbol{Inv\_MixColumns\ (S_{15}^i)} \\ \boldsymbol{Inv\_MixColumns\ (S_2^i)} \\ \boldsymbol{Inv\_MixColumns\ (S_5^i)} \\ \boldsymbol{Inv\_MixColumns\ (S_8^i)} \end{pmatrix} \oplus \begin{pmatrix} \boldsymbol{Rk_{15}^{11-i}} & Rk_{11}^{11-i} & Rk_7^{11-i} & Rk_3^{11-i} \\ \boldsymbol{Rk_{14}^{11-i}} & Rk_{10}^{11-i} & Rk_6^{11-i} & Rk_2^{11-i} \\ \boldsymbol{Rk_{13}^{11-i}} & Rk_9^{11-i} & Rk_5^{11-i} & Rk_1^{11-i} \\ \boldsymbol{Rk_{12}^{11-i}} & Rk_8^{11-i} & Rk_4^{11-i} & Rk_0^{11-i} \end{pmatrix}$$

Puis on réécrit les résultats dans le State^{i+1} enword W1'= $S_{15}^{i+1}, S_{14}^{i+1}, S_{13}^{i+1}$ et S_{12}^{i+1} comme le montre la figure 4.16.

$$\text{State}^{i+1} = \begin{pmatrix} S_{15}^{i+1} & S_{11}^{i+1} & S_{7}^{i+1} & S_{3}^{i+1} \\ S_{14}^{i+1} & S_{10}^{i+1} & S_{6}^{i+1} & S_{2}^{+1i} \\ S_{13}^{i+1} & S_{9}^{+1i} & S_{5}^{i+1} & S_{1}^{i+1} \\ S_{12}^{i+1} & S_{8}^{i+1} & S_{4}^{i+1} & S_{0}^{i+1} \end{pmatrix}$$

Figure 4.16. Résultat intermédiaire de déchiffrement de l'i^{eme} rond.

❖ Les trois autres words (W2, W3 et W4) sont traités de la même manière que W1avec W2 =($S_{11}^{i}, S_{14}^{i}, S_{1}^{i}$ et S_{4}^{i}), W3 = ($S_{7}^{i}, S_{10}^{i}, S_{13}^{i}$ et S_{0}^{i}) et W4 = ($S_{3}^{i+1}, S_{6}^{i+1}, S_{9}^{i+1}$ et S_{12}^{i+1}) suivant le principe de l'opération Inv_ShiftRows.

Cette architecture a les mêmes caractéristiques d'implémentation que le chiffrement, elle a besoin de 4 Inv_Sboxs pour l'implémentation de l'opération Inv_SUB_BYTE et une colonne de 32 bits pour l'implémentation de l'opération Inv_MixColumns, qui a besoin d'une matrice avec des coefficients plus complexes comme il est indiqué dans la matrice **Inv_M(x)**pour le calcul de l'opération Inv_MixColumns par rapport à la matrice utilisée dans le l'opération MixColumns pour le processus de chiffrement.

Les constantes multiplicatives de cette matrice peuvent être exprimées comme suit :

- $\{0b\}S_j^i = \{08\}S_j^i \oplus \{02\}S_j^i \oplus S_j^i$
- $\{0d\}S_j^i = \{08\}S_j^i \oplus \{04\}S_j^i \oplus S_j^i$.
- $\{09\}S_j^i = \{08\}S_j^i \oplus S_j^i$.
- $\{0e\}S_j^i = \{08\}S_j^i \oplus \{04\}S_j^i \oplus \{02\}S_j^i$. (avec : $0 \leq i \leq 15$ et $0 \leq j \leq 3$).

Dans notre conception, nous avons utilisé la fonction XTime (décrite en annexe A) pour le calcul de la multiplication par deux dans le corps GF(2^8). Avec l'utilisation de cette fonction et les simplifications précédentes des coefficients ('0b', '0d', '09', et '0e'), une colonne de cette opération est implémentée par l'utilisation de **92 portes XORs** et donc **368 portes XORs** sont nécessaires pour l'implémentation des quatre colonnes.

Remarque : Les deux autres architectures (Parallèle-Série et Parallèle-Pipeline) pour le déchiffrement ont le même principe et les mêmes caractéristiques d'implémentation que dans le chiffrement, avec les changements mentionnées précédemment dans l'architecture **exécution Série-Série** pour le module **AES_BOC.**

4.5. Fonctionnement du Controller

Toutes les opérations de cryptage et décryptage sont dirigées par le contrôleur de l'IP_AES, appelé "Controller". Le fonctionnement interne du contrôleur est géré par des processus. Pour déclencher le processus de chiffrement, on doit effectuer un test sur le signal *mode*. Le mode 0 indique l'opération de cryptage tandis que le mode 1 indique l'opération de décryptage. Pour générer les clés, le contrôleur met le signal *GenKey* à '1' pour activer la fonction Key_Expansion.

L'emplacement des clés se réalise suivant une variable appelée *Compteur* qui suivant sa valeur, le signal *WriteRam* sera activé pour placer les clés générées dans le bloc Key_Ram.

- Pour lire les clés générées dans le mode de cryptage, le signal *ReadRam* doit être activé. En fait, la fonction AES_Core reçoit les clés du bloc Key_Ram pour crypter le texte en clair (*PlainText*) à partir de la case mémoire d'indice 0(clé originale).
- Pour lire les clés dans le mode de décryptage, le signal *ReadRam* doit être activé. En fait, la fonction AES_Core reçoit les clés du bloc Key_Ram pour décrypter les données à partir de la case mémoire d'indice 10.

4.6. Implémentation du processus Key_Expansion

La fonction Key_Expansion permet de générer les 10 clés et de les garder dans le bloc Key_Ram. Cette fonction consiste à calculer tous les *RoundKey* utilisés dans les opérations *AddRoundKey*de toutes les transformations. Le premier round utilise la clé de cryptage *CipherKey*. Les 9 rounds suivants utilisent ses propres clés dérivant de la clé originale.

Dans toutes les architectures précédentes, nous avons proposé une implémentation Soft pour la génération des clés, gérée par une application C, afin de garantir une optimisation dans les ressources utilisées.

4.6.1. Fonctionnement de Key_Ram

Dans notre conception, nous avons proposé deux architectures de ce bloc. Une architecture (***Key_RAM_4_REGISTERS***) pour les deux modes d'exécution (Exécution Série-Série et Exécution Parallèle-Série) du bloc AES_Core et la deuxième architecture (***Key_RAM_10_REGISTERS***) pour le mode d'exécution Parallèle-.Pipeline.

4.6.1.1. Architecture Key_RAM_4_REGISTERS

Dans notre conception du bloc AES_Core en mode Exécution Parallèle-Série, nous avons utilisé quatre blocs de module (AES_BLOC) pour le module AES_ROUND afin de

minimiser le temps de latence et accélérer la rapidité d'exécution. Lorsque ces quatre unités agissent simultanément, chacune va être dirigée sur sa propre Key_Ram. Pour cela, ce dernier est sectionné en quatre mémoires: Key_RAM_4_REGISTERS_i (pour i = 1 à 4). Comme le montre la figure 4.17

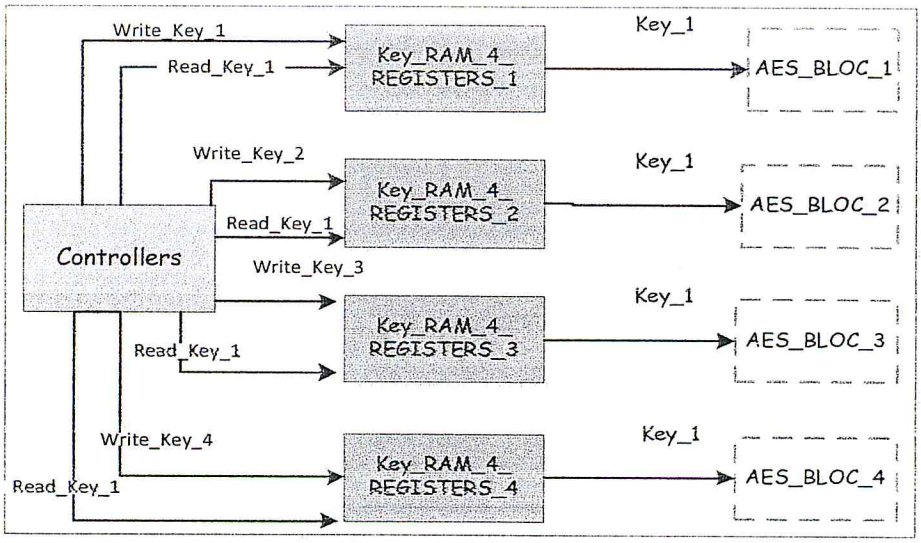


Figure 4.17. Architecture d'AES_ROUND avec Key_RAM.

Dans cette architecture, chaque bloc Key_RAM_4_REGISTERS_i(pour i= 1 à 4) est composée de 11 cases mémoires codées sur 32 bits chacune (10 cases pour les clés générées ainsi que la clé originale) comme le montre la figure 4.18et gérée par les deux signaux Read_Key_i pour la lecture des clés et Write_Key_i pour l'écriture des clés générées à partir la procédure Key_Expansion (pour i = 1 à 4).

Le sens de lecture des clés dans le mode de cryptage passe par la case mémoire d'indice 0 jusqu'à la case d'indice 10 et par la case d'indice 10 jusqu'à la case d'indice 0 pour le mode de décryptage.

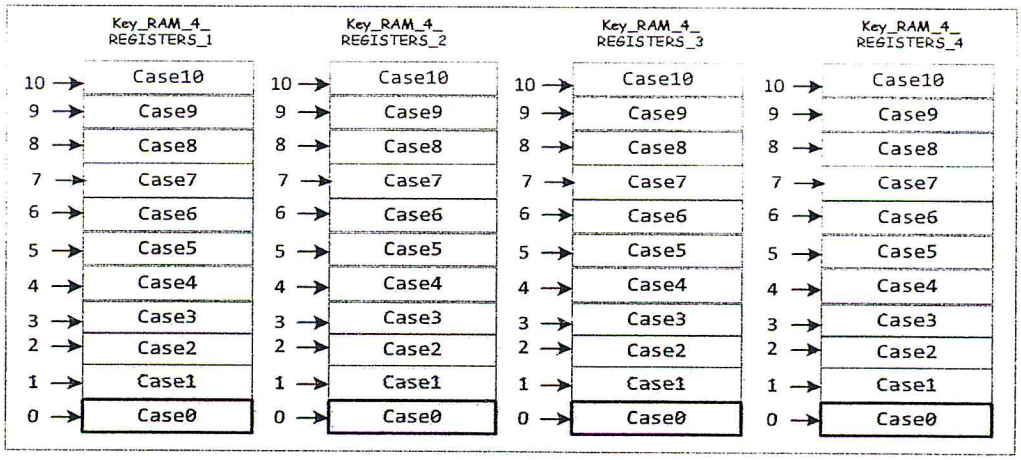


Figure 4.18. Architecture de Key_RAM_4_REGISTERS.

4.6.1.2. Architecture Key_RAM_10_REGISTERS

Dans notre conception du bloc AES_Core en mode Exécution Parallèle-Pipeline, nous avons utilisé dix blocs AES_ROUND pour minimiser le temps de la latence prohibitif et accélérer la rapidité de l'exécution. Lorsque ces dix unités agissent simultanément, chacune va être dirigée sur sa propre Key_Ram. Pour cela, ce dernier bloc est sectionné en dix mémoires : Key_RAM_4_REGISTERS_i (pour i = 1 à 10).Comme il est illustré parla figure 4.19, chacune de ces mémoires est composée de 11 cases mémoires de 128 bits chacune (10 clés ainsi que la clé originale). Le sens de lecture des clés dans le mode de cryptage passe par la case mémoire d'indice 0 jusqu'à la case d'indice 10 et par la case d'indice 10 jusqu'à la case d'indice 0 pour le mode de décryptage.

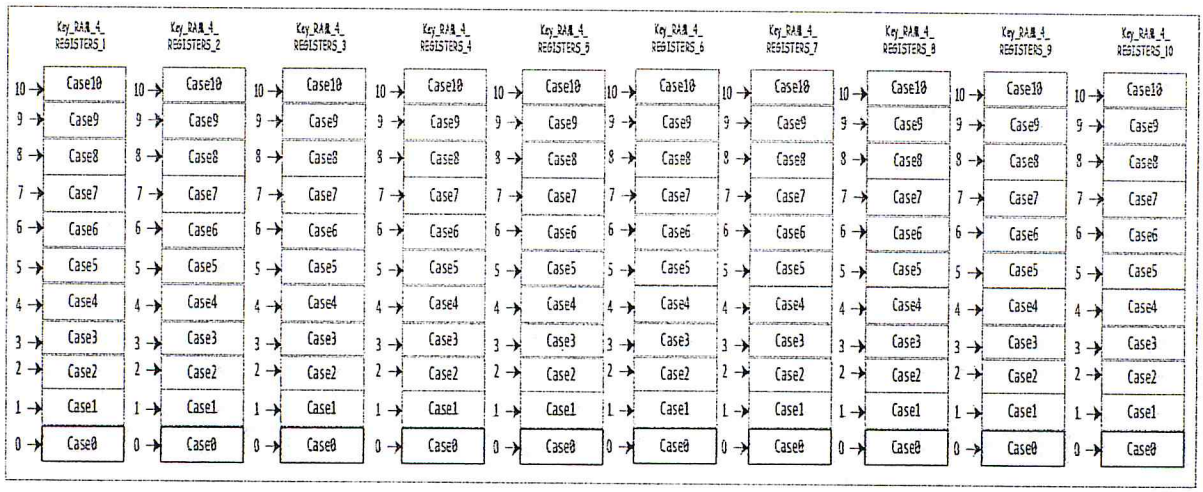


Figure 4.19. Architecture de Key_RAM_10_REGISTERS.

4.7. Implémentation du RSA

Le RSA est utilisé pour résoudre le problème d'échange de clé pour l'algorithme AES, en cryptant la clé utilisé par l'AES pour être transmise via un canal non sécurisé en même temps que le message chiffré par l'AES.

Dans notre travail, nous avons proposé une plateforme de chiffrement /déchiffrement de l'algorithme RSA gérée par une application C.

4.7.1. Limites du RSA

Dans la pratique, le RSA est un code sûr, si l'on respecte les règles suivantes :

1. **p** et **q** doivent être très grands.
2. Il faut crypter le message par blocs de plusieurs caractères.

Le RSA possède également quelques inconvénients d'ordre mathématique :

1. On ne peut pas choisir un **n** inférieur à la valeur maximale à coder. Si **n** est trop petit, plusieurs caractères pourront être cryptés par le même nombre, et ne pourront donc plus être différenciés lors du décryptage.
2. Les calculs sont souvent très lourds, du fait de la taille des entiers à manipuler. Le cryptage d'un message long, avec des clés de grande taille, peut prendre plusieurs heures sur un ordinateur puissant.

4.7.2. Choix algorithmiques

Nous allons maintenant décrire les étapes utilisés dans le RSA :

4.7.2.1. Génération des clés

L'algorithme suivant permet de vérifier si un nombre est premier ou pas.

```
booléen EstPremier(n : entier long ; premier : booléen)

    booléen premier
    premier <- VRAI
   POUR i <- 2 JUSQU'A i < n FAIRE
   SI (reste de n modulo i = 0)
    ALORS
    premier <- FAUX
Finsi
 i <- i+1
 RENVOYER premier
```

- Il nous suffit maintenant de choisir un nombre au hasard, et tester si il est premier ou non. S'il ne l'est pas, on choisit au hasard un nouveau nombre, et ainsi de suite jusqu'à en obtenir un premier. On répète cette opération deux fois, pour **p** et pour **q**.
- A partir de p et q, on calcule n, puis z.

4.7.2.1.1. Génération de e

La clé publique **e** doit impérativement être première avec **z**. Nous devons donc construire un algorithme capable de déterminer si deux nombres (a,b) sont premiers entre eux, c'est à dire que PGCD(a,b) = 1.La méthode la plus simple et la plus rapide pour déterminer un PGCD reste sans conteste l'algorithme d'Euclide, que nous pouvons programmer de la manière suivante :

```
entier long Pgcd (entier long a, entier long b ; entier long
pgcd)

      entier long c ;
      c <- 0 ;
      c <- reste de a modulo b ;
     SI (c = 0)  ALORS
      RENVOYER b ; // a est multiple de b, le pgcd est donc b
   SINON
   TANT QUE c ≠ 0 FAIRE
       a <- b ;
       b <- c ;
      c <- reste de a modulo b ;
 RENVOYER b ;
```

Nous pouvons alors choisir un nombre e < z au hasard et déterminer si celui-ci convient (⇔ Pgcd(e,z) = 1). Si ce n'est pas le cas, on choisit un autre e jusqu'à en trouver un correspondant :

```
entier long generer_e (entier long z)

entier long e ;
   REPETER
     e <- (3 + rand()) % z ;
JUSQU'A(Pgcd(e, z) ≠1) ;

   RENVOYER e ;
```

4.7.2.1.2. Génération de d

Le calcul de **d**, la clé privée, est l'opération la plus lourde. Rappelons tout d'abord ce qu'est l'inverse d'un nombre modulo z :

***d** inverse de **e** modulo z⇔d = e^{-1} [z]<⇔**d** * **e** = 1 [z].*

On utilise alors la boucle suivante pour déterminer **d.**

```
entier long generer_d (entier long e, entier long z)

entier long d ;
  d <- 0 ;
  TANT QUE (reste de (e*d) modulo z) ≠1 FAIRE
    d <- d+1 ;
   RENVOYER e ;
```

Nous avons donc généré nos clés publiques et privées et pouvons donc entamer le cryptage d'un message.

4.7.2.2. Cryptage

L'algorithme suivant permet le cryptage d'un message:

```
entier court codage(lettre : caractère, n : entier long , e
: entier long )
entiers courts : i, c, t ;
 t = (entier court)lettre ;
 c = 1;
 // effectue c = t*e mod n
  POUR i <- 0 JUSQU'A i < e FAIRE
   c <- c * t ;
   c <- reste de c modulo n ;
   i <- i+1 ;
RENVOYER c ;
```

On peut remarquer deux choses :

- Le caractère lettre (8 bits) est converti au format « entier court » 16 bits au moyen d'un transtypage.

- La fonction puissance est réalisée au moyen d'une boucle de multiplication. Toutefois, à chaque cycle, on ne multiple que le reste modulo n de la multiplication précédente. Cela permet d'économiser un temps de calcul considérable et d'empêcher des débordements de mémoire.

4.7.2.3. Décryptage

Le décryptage fonctionne comme le cryptage, à trois détails près :

- L'algorithme de décodage devient $\mathbf{t = c^d\ [n]}$,

- Le fichier source devient le fichier crypté et le fichier destination le fichier une fois décrypté,

- Lors du décodage, on lit 2 octets à la fois dans le fichier source et en décryptant, on obtient un caractère codé sur 1 octet.

```
Caractère décodage (lettre:entiercourt,n:entierlong,d:entierlong)
caractère : c ;
entiers courts : i, c ;
caractère : t
  // effectue c = t*e mod n
   POUR i <- 0 JUSQU'A i < d FAIRE
     c <- c * lettre ;
     c <- reste de c modulo n ;
     i <- i+1 ;
     t = (caractère) c ;
RENVOYER t ;
```

4.8. Conclusion

Dans cette partie de notre travail, nous avons présenté la conception de l'architecture matérielle pour l'implémentation du chiffrement/déchiffrement de l'AES, des différents modes d'exécution à savoir le mode exécution Série-Série, exécution Parallèle-Série et de l'exécution Parallèle-Pipeline. Vu ces caractéristiques d'implémentation, l'architecture en mode Pipeline est le bon choix en terme de temps d'exécution pour une implémentation matérielle de l'AES dans des applications qui nécessitent un fort débit.

Ainsi, nous avons présenté l'implémentation du processus de génération des clés d'AES Key expansion et les différents algorithmes utilisés pour l'implémentation du cryptage RSA qui sont implémentés par une application C.

Dans le chapitre suivant, nous présenterons l'évaluation de l'implémentation de l'IP d'AES en VHDL sur une plateforme FPGA de Xilinx, ainsi que les tests de fonctionnement de ces composants à travers des simulations avec des testbenshs.

Chapitre 05

Simulation Et Résultats

5.1. Introduction

Après avoir détaillé la conception de notre IP-AES dans le chapitre précédent et choisi l'architecture pipeline présentant les meilleures performances en termes de débit, nous présenterons, dans ce chapitre, les résultats de simulation de cette architecture bloc par bloc et leurs implémentations sur circuit FPGA.

Pour implémenter notre architecture, nous avons utilisé l'outil ISE Simulator de Xilinx par l'implémentation des Tests-Benchs. Nous avons d'abord testé les composants d'un seul round, puis nous avons combiné les composants pour tester le fonctionnement d'un round d'AES.

5.2. Méthodologie de conception

La méthodologie de conception est basée sur le langage de description matériel VHDL, l'outil ISE Simulator (VHDL/ Verilog) a été utilisé pour la simulation et la synthèse au niveau RTL a été faite par XST (VHDL / Verilog).

Cette méthodologie est utilisée pour tester le bon fonctionnement de notre architecture et pour extraire les résultats de l'implémentation matérielle sur circuit FPGA de la famille Virtex-5, lexc5vlx330t-2ff1738.

Elle doit répondre à la fois aux objectifs de la description architecturale utilisée comme spécification et aux contraintes de réalisation (les performances).

5.2.1. Description de l'ISE «Integrated Software Environnement »

C'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 contrainte de timings sur les horloges, de déterminer l'emplacement des broches et de créer des fichiers d'essai de simulation (Test Bench).

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

- ✓ Un éditeur de textes, de schémas et diagrammes d'états.
- ✓ D'un compilateur VHDL et Verilog.
- ✓ D'un simulateur.
- ✓ D'outils pour la gestion des contraintes temporelles

- ✓ D'outils pour la synthèse.
- ✓ D'outils pour la vérification.
- ✓ D'outils pour l'implémentation sur FPGA et CPLD.

Les étapes pour l'implémentation d'une spécification HDL sur un FPGA sont illustrées sur la figure 5.1.

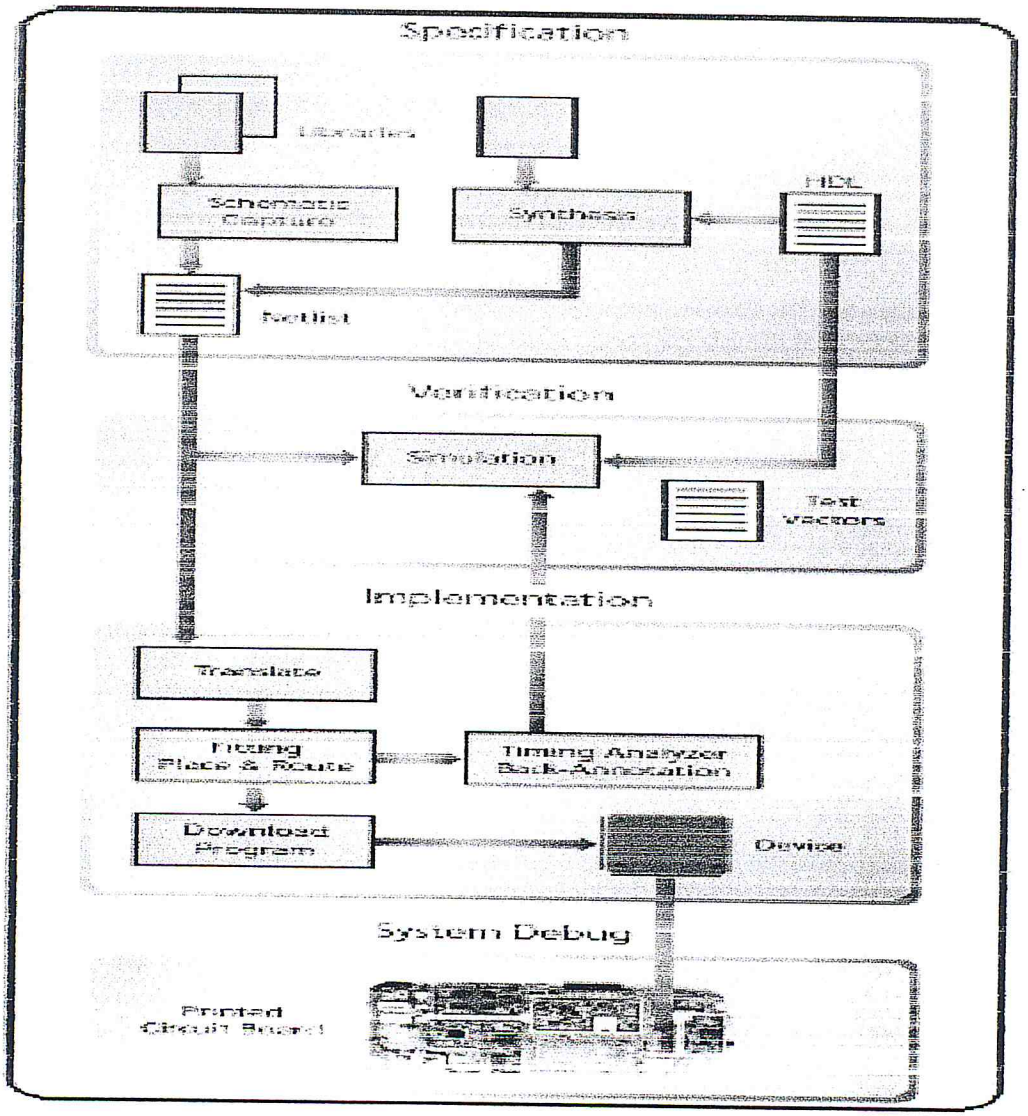


Figure 5.1. Les étapes d'implémentation d'un circuit sur un circuit logique programmable Xilinx [WEP 06].

5.2.1.1. Spécification

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

5.2.1.2. Vérification :

La vérification du design est une étape parallèle où le concepteur observe le comportement du code et 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.

5.2.1.3. Implémentation

Une fois la vérification est terminée, le circuit est implémenté sur le composant en spécifiant les références exactes de celui-ci à savoir : la carte utilisée, 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).

Cette étape se décompose à son tour en sous-étapes qui sont :

a) **Place and Route** : Les sous-programmes de placement et routage sont exécutés après compilation du code.

a.1 .Place : c'est le processus de sélection où les portes logiques seront placées.

a.2. Route : le routage est l'interconnexion physique entre les différents blocs logiques.

5.2.2. Language de programmation VHDL

Le VHDL (**V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage) est un langage de description matériel HDL portable et synthétisable.

5.2.2.1. Structure d'un programme VHDL

a) Entity: La partie déclarative de l'entité d'un circuit est décrite à travers les entrées et le sorties.

b) Architecture : l'architecture décrit le comportement que doit effectuer le circuit. Une architecture se doit toujours d'être attachée à une entité. C'est dans cette section que le programme est rédigé. Un programme comporte essentiellement les éléments suivants :
(Les signaux internes, opérateurs logiques (synchrone ou les process).

5.3. Implémentation sur circuit FPGA

Dans cette étape, il est question de concevoir la partie matérielle de l'IP AES de notre système.

Notre implémentation de cet *IP* été basée sur les composants qui permettent d'utiliser et de fournir un ensemble de modules. Sachant que, nous avons implémenté trois architectures pour le chiffrement AES. Ces architectures sont : Exécution série-série, Exécution parallèle-série et Exécution parallèle-pipeline, qui se diffèrent entre eux dans la taille du chemin de données et la manière d'exécution série ou parallèle.

5.3.1. Résultats de simulation

Les différents blocs fonctionnels de l'architecture de l'IP-AES ont été testés et vérifiés par simulation avant de les implémenter dans un circuit FPGA. La simulation consiste à envoyer, via un fichier Test-Bench décrit en VHDL, des stimuli aux entrées du système et à observer le comportement de ses sorties. L'outil de simulation utilisé est ISE Simulator permettant de visualiser la variation des signaux de sortie et par la suite effectuer des modifications au niveau de la description en cas de résultats insatisfaisants.

5.3.1.1. Résultats de simulation du module SUB_BYTE

5.3.1.1.1. Implémentation en BRam

Les résultats de simulation du module SUB_BYTE en mode cryptage sont illustrés par la figure 5.2.

Figure 5.2. Résultats de simulation de module SUB_BYTE en BRam.

Dans cette figure, nous avons choisi comme entrée du module SUB_BYTE deux blocs de données (*BLOC_in*) de 32 bits chacun, avec les valeurs: *X"00010203" et X"0C0D0E0F"*. Après la compilation et la simulation du code et son Test-Bench, les valeurs obtenues de ces deux blocs sont apparues sur le signal de sortie (*BLOC_out)* avec les valeurs : X"637c777b" et X"fed7ab76" pour le premier et deuxième bloc respectivement, ces résultats sont identiques aux valeurs du signal de comparaison (*correct_output).*

5.3.1.1.2. Implémentation dans le GF (2^8)

La figure 5.3 représente les résultats de simulation du module SUB_BYTE en mode de cryptage par l'implémentation dans le corps GF(2^8) avec 7 étages pipelines.

L'entrée (BLOC_in) du module SUB_BYTE est une donnée de 8 bits chaque 10 ns. Après compilation et simulation du code et son Test-Bench, les résultats sont apparus sur le signal de sortie (BLOC_out) après l'insertion de la septième donnée avec des valeurs correspondantes aux valeurs du signal de comparaison (correct_output) comme il est indiqué sur la figure 5.3.

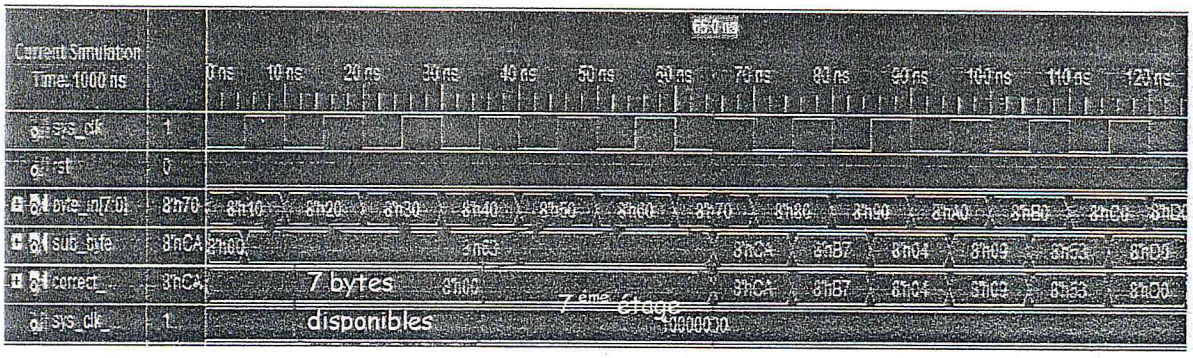


Figure 5.3. Résultats de simulation du module SUB_BYTE dans GF (2^8).

5.3.1.2. Résultats de simulation du module MixColumns

La figure 5.4 représente les résultats de simulation du module MixColumns en mode cryptage.

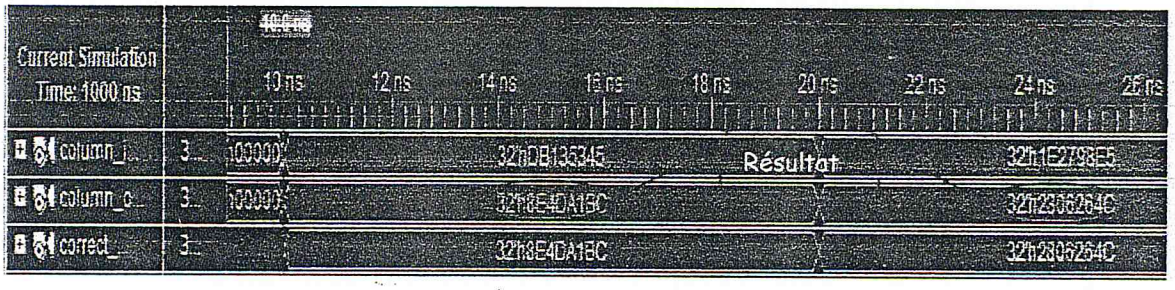


Figure 5.4. Résultats de simulation du module MixColumns.

Dans cette figure, nous avons choisi comme entrée du module MixColumns deux blocs de données(*Column_in*) de 32 bits chacun avec les valeurs : *X"db135345"* et *X"1e2798e5"*, après 10 ns les résultats de simulation sont obtenus. Après compilation et simulation du code et son Test-Bench, les valeurs obtenues de ces deux blocs sont apparues sur le signal de sortie (*Column_out)* avec les valeurs : X"8e4da1bc" et X"2806264c", ces valeurs sont identiques aux valeurs du signal de comparaison (*correct_output)*.

5.3.1.3. Résultats de simulation de module LatcINPUT

La figure 5.5 représente les résultats de simulation du module LatcINPUT en mode cryptage. Nous avons choisi comme entrée de ce module une donnée *(INPUT)* de 128 bits avec la valeur : X"db135345AB12F5ED58DA276AFF11FF5D". Après compilation et simulation du code et son Test-Bench, le résultat est apparu sur les quatre signaux de sorties *(W1, W2, W3 et W4)* avec les valeurs : X"DB12275D", X"ABDAFF45",X"581153ED" et X"FF13F56A" respectivement.

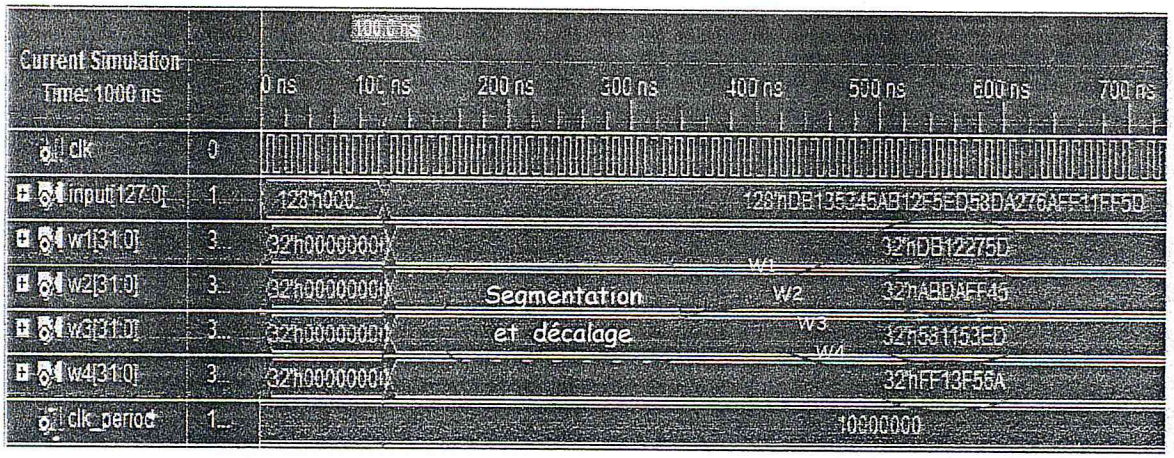


Figure 5.5. Résultats de simulation du module LatchINPUT.

Dans cette figure, on remarque qu'à chaque front montant de l'horloge Clk, l'interface d'entrée reçoit un bloc de données de 128 bits à travers l'entrée INPUT. Après décalage et partitionnement en quatre words de données d'une largeur de 32 bits chacun, cette interface les envoie à leur destination à travers ses quatre sorties.

5.3.1.4. Résultats de simulation du module BRam (cas de 32 bits)

La figure 5.6 représente les résultats de simulation du module BRam.

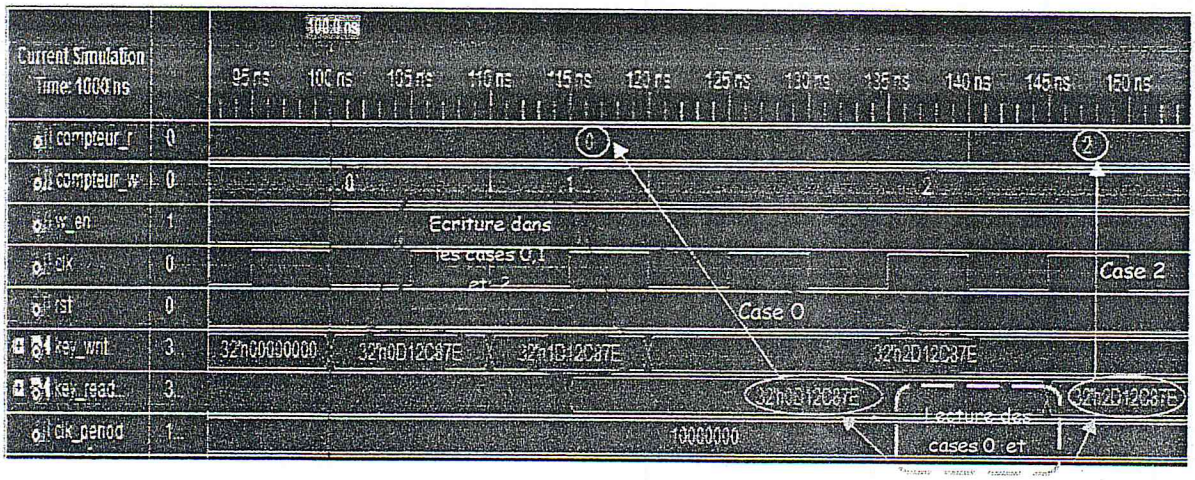


Figure 5.6. Résultats de simulation du module BRam.

Dans la figure 5.6, nous avons choisi comme entrée du module BRam trois blocs de données *(Key_Write)* de 32 bits chacun avec les valeurs : X"0D12C87E", X"1D12C87E" et X"2D12C87E" dans le but de les insérer dans les trois cases mémoires (0, 1 et 2) respectivement qui sont définies par le signal *Compteur_w*. Par la suite, nous avons testé la lecture des deux cases mémoires 0 et 2, à l'aide du signal *Compteur_r*. Après compilation et simulation du code et son Test-Bench, les données entrantes par le signal *Key_Write*sont bien insérées dans les bonnes cases mémoires et les valeurs obtenues par le signal de sortie *(Key_Read)* sont :X"0D12C87E" et X"2D12C87E", identiques aux données insérées dans ces cases mémoires par le signal d'entrée (*Key_Write*).

5.3.1.5. Résultats de simulation du module Inv_MixColumns

La figure 5.7 représente les résultats de simulation du module Inv_MixColumns.

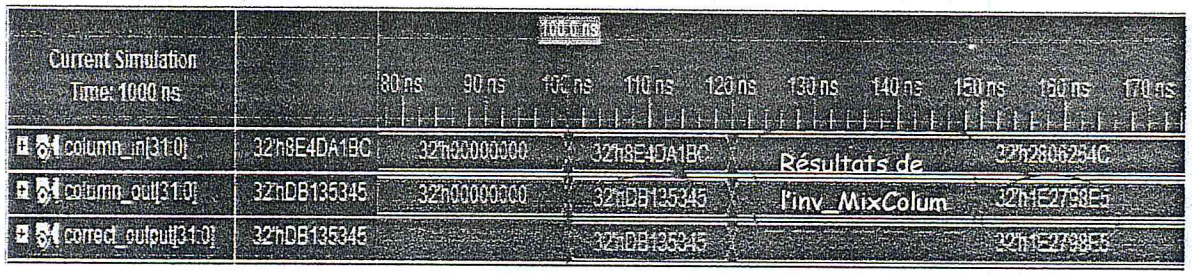


Figure 5.7. Résultats de simulation du module Inv_MixColumns.

Dans cette figure, nous avons choisi comme entrée du module Inv_MixColumns deux blocs de données (*Column_in*) de 32 bits chacun avec les valeurs : *X"8e4da1bc"*et*X"2806264c"*etaprès 10ns, les résultats de simulation sont obtenus. Après compilation et simulation du code et son Test-Bench, les valeurs obtenues de ces deux blocs sont apparues sur la sortie (*Column_out)* avec les valeurs : X"db135345"et X"1e2798e5", ce résultat est identique aux valeurs du signal de comparaison *correct_output*.

5.3.1.6. Résultats de simulation du cryptage en mode «Parallèle-Pipeline »

La figure 5.8 représente le résultat de simulation de cryptage d'un bloc de 128 bits en mode d'exécution parallèle-pipeline. Nous avons choisi comme entrée : une donnée (data_in) de 128 bits et une clé de chiffrement (Partial_Keyi) « pour i = 0 à 10 » de 128 bits chaque 10 ns. Après compilation et simulation du code et son Test-Bench, les valeurs obtenues sont apparues sur le signal de sortie (Word_out) avec une taille de 128 bits après l'insertion de la onzième donnée comme c'est indiqué sur la figure 5.8.

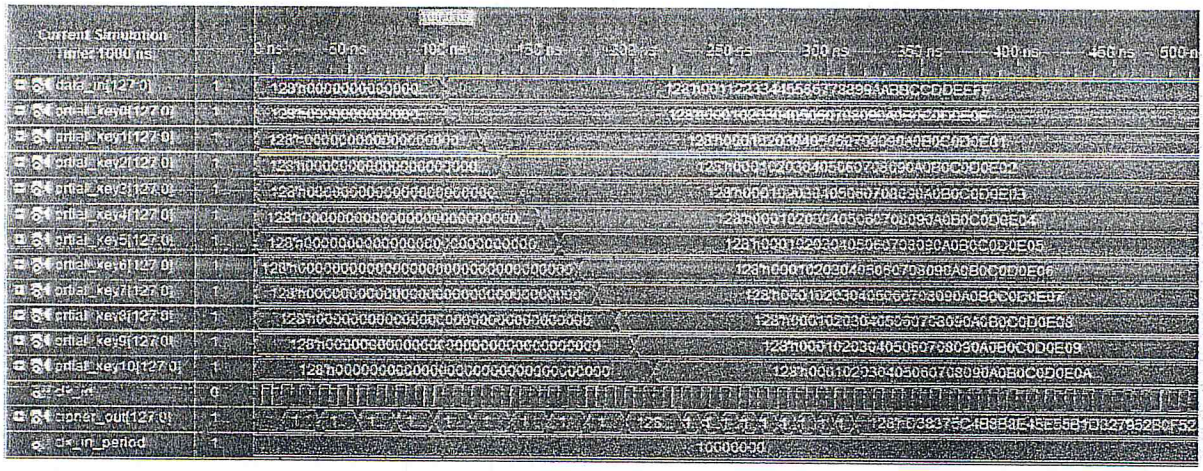


Figure 5.8. Résultats de simulation du cryptage en mode Exécution parallèle-pipeline.

5.3.2. Résultats de synthèse et d'implémentation

Dans ce travail, la synthèse a été effectuée en utilisant l'outil XST (VHDL /Verilog) (Xilinx Simulator Tools VHDL/Verilog). La cible matérielle est le FPGA, Virtex-5Target devicexc5vlx330-2ff1760 pour toutes nos implémentations.

5.3.2.1. Résultats d'implémentation de l'architecture Série-Série

Les résultats de l'implémentation des différents modules de cette architecture sont regroupés dans le tableau 5.1 suivant :

Ressources / Modules	Slice Registers	Slice LUTs	Logic	LUT Flip Flop pairs	FREQ (MHz)
AddRK	/	32 (0%)	32	32	645,161
SUB_BYTE	/	128 (0%)	128	128	678,426
Inv_SUB_BYTE	/	128 (0%)	128	128	678,886
MixColumns	/	40 (0%)	40	40	456,621
Inv_MixColumns	/	103(0%)	103	103	368,324
Aigulleur	/	32 (0%)	32	32	588,235
D_aiguilleur	128	4(0%)	4	4	645,161
LatchINPUT	128	/	/	/	571,428
BRam	384	237(0%)	237	551	468,603
LATCHOUTPUT		/	/	/	574 ,712
AES_BLOC	/	174 (0%)	174	174	261,369
AES_ROUND	/	174 (0%)	174	174	74,074

Tableau 5.1. Résultats d'implémentation des modules d'IP d'AES en mode (Exécution Série-Série).

Selon les résultats d'implémentation de cet IP sur FPGA de la famille Virtex-5,on peut remarquer que l'espace occupé de la totalité des blocs est très réduit pour cette architecture (Exécution série-série),avec une fréquence de : 74,074 MHz. La fréquence d'opération est très faible.

L'implémentation de cet IP montre que celui-ci possède un débit de 39,272 Mbits/s contre un taux d'occupation de1%. Le débit de cet IP est calculé par la formule suivante:

$$V\,(Mbit/s) = (Nb\ de\ cycle / Fréquence)^{-1} \times 32.$$

Nous avons pris des schémas RTL et Floorplanner d'un round de cet IP illustrés dans les figures5.9 et 5.10 respectivement.

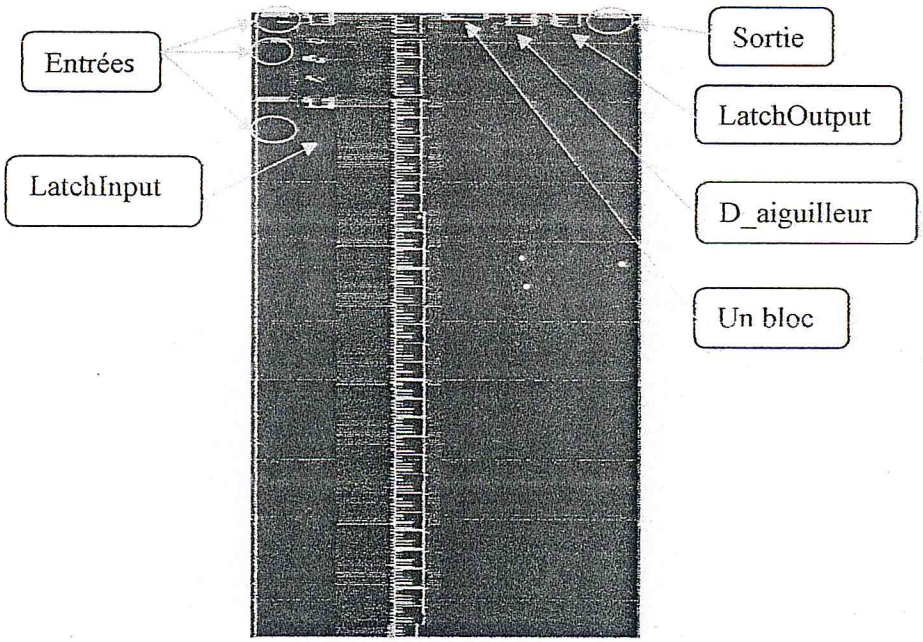


Figure 5.9. Schéma RTL en blocs d'un round d'IP AES (Exécution série-série).

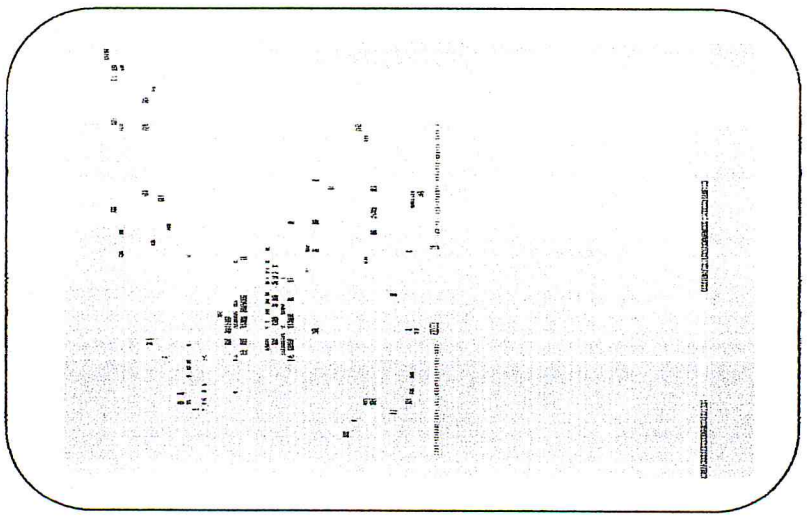


Figure 5.10. Schéma Floorplanner d'un round d'IP AES (Exécution série-série).

5.3.2.2. Résultat d'implémentation des architectures parallèle-Série et Parallèle-pipeline

Les différents modules des deux architectures (parallèle-série et parallèle-pipeline) sont de même taille de 128 bits, sauf que l'architecture parallèle-pipeline présente un dédoublement des ressources de 10 fois par rapport à celle parallèle-série

Les résultats de synthèse et d'implémentation de ces modules sont regroupés dans le tableau 5.2.

Modules	Slice Registers	Slice LUTs	Logic	LUT Flip Flop pairs	FREQ (MHZ)
AddRK	/	129 (0%)	129	129	645,161
SUB_BYTE	/	512 (0%)	512	512	678,426
Inv_SUB_BYTE	/	512 (0%)	512	512	678,886
MixColumns	256	158 (0%)	158	286	313,479
Inv_MixColumns	/	103(0%)	103	103	269,179
LatchINPUT	128 (0%)	/	/	/	571,428
LATCHOUTPUT		/	/	/	574 ,712
AES_BLOC		174 (0%)	174	174	261,369
BRam	1536	633 (0%)	633	1932	431,034
AES_ROUND	384	169 (0%)	169	425	205,338

Tableau 5.2. Résultats d'implémentation des modules de l'IP d'AES (parallèle-série et parallèle-pipeline).

✓ Selon les résultats d'implémentation sur FPGA de la famille Virtex-5,on peut remarquer que l'espace occupé de la totalité des blocs de l'IP AES est réduit pour la version d'exécution parallèle-série 1% et élevé pour la version parallèle-pipeline 9% et le débit de fonctionnement de cette dernière version est très rapide par rapport à la première.

En conclusion, les résultats observés dans cet IP pour les deux modes d'exécution parallèle-série et parallèle-pipeline sont liés à plusieurs facteurs et il est difficile de se prononcer sur ces résultats s'ils sont purement semblables aux résultats des autres conceptions, compilations, ou facteurs de technologie. En plus, le test avec différents outils de technologie et de compilation de dispositif aiderait également à évaluer l'efficacité d'un système.

Nous avons pris des schémas RTL illustrés par les figures 5.11 pour l'IP AES en mode parallèle-pipeline et deux schémas Floorplanner (a) et (b) illustrés par les figures5.12 pour l'IP AES en mode parallèle-pipeline et parallèle-série respectivement.

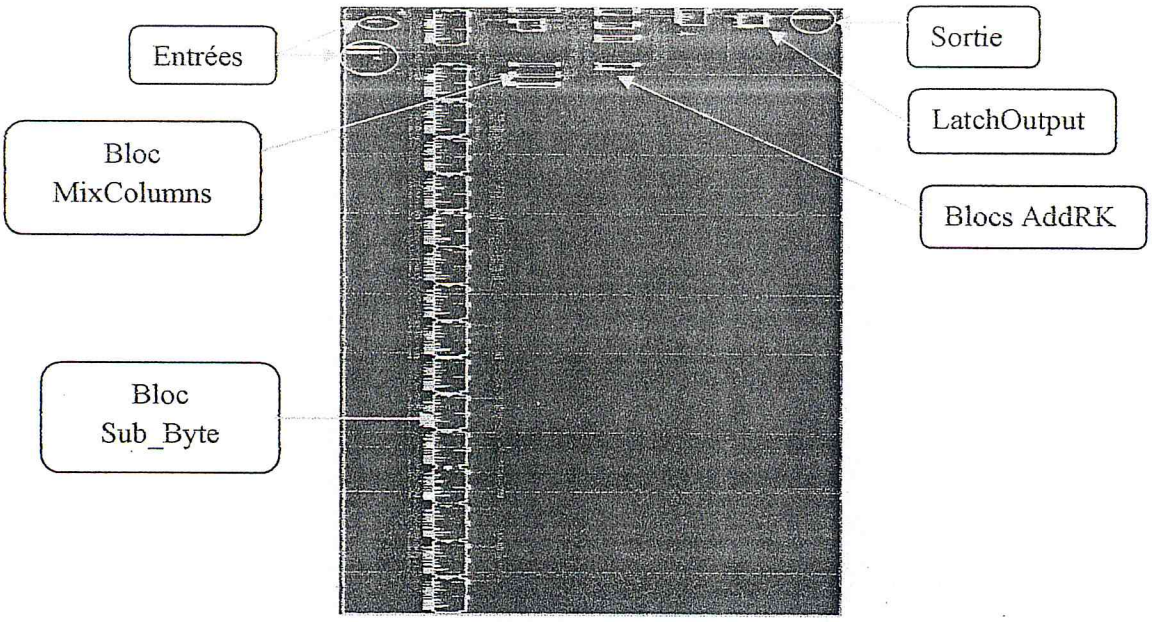
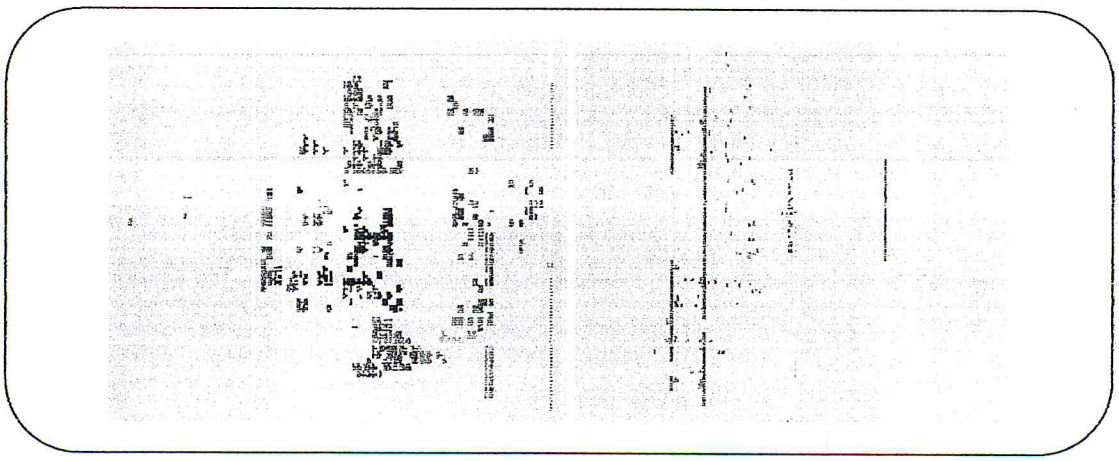


Figure 5.11. Schéma RTL De l'IP AES en mode parallèle-pipeline.



(a) (b)

Figure 5.12. Schéma RTL De l'IP AES en mode parallèle-pipeline et parallèle-série.

5.3.2.3. Résultats d'implémentation du module SUB_BYT dans le GF (2^8)

Les résultats d'implémentation de ce module sont regroupés dans le tableau 5.3

Paramètres	Slice Registers	Slice LUTs	Logic	LUT Flip Flop pairs	FREQ (MHz)	Timing (ns)
Valeurs	84	67 (0%)	67	96	635,727	1.573

Tableau 5.3. Résultats d'implémentation de module SUB_BYTE dans le corps GF (2^8).

En comparant avec l'implémentation en BRams et selon les résultats d'implémentation sur le circuit FPGA de Virtex-5, le nombre de slices occupés par la S-Box implémentée dans le GF (2^8) est réduit de 128 à 67 par l'assemblage de l'inverse multiplicatif avec la transformation affine. D'où dans l'implémentation sur FPGA, la fonction de mappage inverse (δ^{-1}) et la transformation affine sont combinées pour réduire le nombre de slices occupés par la S-box.

Ainsi, **7 étages pipeline** sont utilisés pour réduire le retard de la logique dans le but d'atteindre une fréquence élevéede635.526 MHz.

5.4. Conclusion

Au fil de ce dernier maillon de notre travail, nous avons présenté les résultats des tests d'implémentation des différents modules de l'IP AES. Cet IP a été évalué dans trois types d'architectures: Série-Série, Parallèle-Série et Parallèle-Pipeline pour qu'il soit intégré dans un système hybride RSA-AES embarqué sur circuit FPGA de Xilinx.

D'après les résultats de simulation avant et après la synthèse, l'IP AES en mode pipeline a été testé efficacement, avec une vitesse élevée sur la technologie FPGA. Pour valider réellement le fonctionnement de cet IP, on doit l'intégrer dans un système embarqué sur puce pour s'assurer de son bon niveau de sécurité.

Le test de l'implémentation du module SUB_BYTE dans le corps GF (2^8) avec sept étages pipelines permet de rajouter de bonnes performances à cet IP, car ce mode d'implémentation permet une réduction dans l'occupation des ressources avec une vitesse élevée garantie par l'insertion des registres pipelines.

Conclusion Générale

Conclusion Générale

Au terme de notre étude qui est la cryptographie embarquée, nous avons introduit des notions générales sur la cryptographie et plus précisément les protocoles symétriques et asymétriques. Une conclusion est faite que les protocoles cryptographiques à clé publique présentent l'avantage d'échanger des messages de manière sûre sans échange préalable de secret et les protocoles symétriques sont rapides et simples à implémenter. Les avantages des chiffrements symétrique et asymétrique ont été combinés pour décrire le principe de fonctionnement d'un crypto-système hybride basé sur le RSA et l'AES.

D'un point de vue matériel, la réalisation d'un système sur puce, aujourd'hui, est plus accessible grâce aux FPGAs de plus en plus performant alliés aux blocs IPs réutilisables qui sont très répandus et qui permettent de maîtriser la complexité croissante des SoCs.

Plusieurs travaux ont intégrés des cœurs de sécurité dans des plates-formes reconfigurables ou des cartes à puce. Ils ont utilisés des crypto systèmes tels que l'AES, le SHA, ou le 3-DES.

Dans notre crypto-système hybride RSA-AES à intégrer dans une plate-forme de chiffrement reconfigurable, le partitionnement de ce système sur les deux ressources logicielles et matérielles pour une bonne conception s'est effectué en tenant compte des performances à atteindre telles que la surface occupée et le temps d'exécution.

En effet le RSA est un algorithme de chiffrement asymétrique utilisé dans notre système hybride RSA-AES une seule fois pour chiffrer la clé secrète de l'AES. Donc l'implémentation logicielle du RSA sera le bon choix pour une optimisation dans le matériel. En outre l'AES est un algorithme symétrique rapide car il utilise des clés de petites tailles et s'adapte bien à une implémentation matérielle car il nécessite des opérations simples telles que l'addition et des décalages. En plus l'AES nécessite une procédure de diversification des clés « *Key Expansion* » qui permet de créer des sous clés utilisées dans le chiffrement.

Certaines architectures matérielles de l'AES sont appropriées à des applications de haute vitesse, tandis que d'autres sont destinées à des applications à faible puissance, alors nous avons étudié des techniques d'optimisation architecturales et algorithmiques pour une implémentation matérielle efficace de l'AES.

L'AES a été implémenté sur le matériel par trois architectures en utilisant les circuits FPGA. Avec ces architectures nous avons procédé à une nouvelle manière de conception où l'accès au State de données par la diagonale nous a permis de résoudre le problème de goulot

d'étranglement dans l'opération de ShiftRows. Aussi, l'implémentation de l'opération MixColumns avec la fonction Xtime a permis une réduction dans les portes logiques.

Les performances obtenues par l'implémentation de ces architectures ont été satisfaisantes en termes de surface, débit et fréquence de fonctionnement où l'architecture pipeline offre de meilleurs résultats comparés à l'architecture série.

Ce projet est donc, un petit pas pour la cryptographie embarquée, mais un grand pas dans notre expérience!

Perspectives

La recherche effectuée dans notre travail a également ouvert d'autres axes de recherche qui pourraient être explorées. Le premier est de pouvoir apporter d'autres améliorations pour une conception efficace de l'AES. Cependant l'utilisation d'un corps composé GF $(((2^2)^2)^2$ basé sur l'inverse multiplicatif pour le calcul des valeurs de la S-Box a été examinée, il a été suggéré que les implémentations basées sur GF $((2^4)^2)$ pourraient absorber moins de puissance. Une version pipeline de l'implémentation de la S-Box a pu mener aux résultats efficaces. En outre, cette expérience pourrait ajouter de la valeur à d'autres structures d'AES, telles qu'employer des largeurs de chemin de données inférieures ou dérouler entièrement l'architecture afin d'évaluer l'impact de ces contraintes de conception sur l'occupation des ressources ou d'employer le mode sous-pipeline pour une architecture plus rapide.

Ainsi, pour valider réellement le fonctionnement de l'IP AES, on doit l'intégrer dans un système embarqué sur puce pour s'assurer du bon niveau de sécurité. Dont le but final est de l'intégrer dans une plate-forme de chiffrement reconfigurable où l'IP AES sera placé avec d'autres IPs des autres algorithmes tels que les ECC, SHA... etc.

En conclusion, les résultats observés dans l'IP AES en mode pipeline sont liés à plusieurs facteurs et il est difficile de se prononcer sur les résultats s'ils sont exactement semblables à ceux d'autres conceptions, compilations, ou facteurs de technologie. En plus, le test avec différents outils de technologie et de compilation du dispositif aiderait également à évaluer l'efficacité d'un système.

ANNEXES

Annexe A

1. Galois Field GF (2^8)

1.2.2. L'Arithmétique de l'AES

L'**AES** est un algorithme de chiffrement orienté « **byte** », un **byte** est la concaténation de 8 bits **{b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0}**. Les valeurs de ces bytes peuvent être interprétées comme des éléments dans le corps Galois Field GF (2^8), et peuvent être représentés algébriquement sous forme de polynômes de degrés ≤ 7 par :

($b = b_7 x^7+b_6 x^6+b_5 x^5+b_4 x^4+b_3 x^3+b_2 x^2 +b_1x+b_0$), à coefficients dans (0 ou 1) [DUT 11], [DAR 99] et [ROL 02].

1.2.3. Le corps de Galois (2^8)

Les éléments du corps GF (Galois Field) (2^8) peuvent être représentés de plusieurs façons qui sont tous isomorphes.

Un octet **b** (byte), est représenté par les bits: b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0. Il peut être aussi considéré comme un polynôme ayant des coefficients dans {0,1} représenté par:

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

L'information peut être représentée sous trois formes

1. **Binaire :** avec {0, 1}
2. **Polynômiale :** avec des coefficients dans {0,1}.
3. **Hexadécimal :** avec des éléments dans {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.

Exemple : L'octet de la valeur hexadécimale **'57 '**(binaire **01010111)** correspond au polynôme: $x^6 + x^4 + x^2 + x + 1$ [ARM 11], [DUT 11] et [DAR 99].

1.2.3. Lois de Composition

1. L'Addition : 'XOR'

L'addition de deux éléments dans le GF (2^8) correspond à un simple XOR bit à bit au niveau du byte, avec un élément neutre ('00 ').

$$A(x) = \sum_{i=0}^{7} a_i\ x^i \oplus \sum_{i=0}^{7} b_i\ x^i = \sum_{i=0}^{7} (a_i \oplus b_i)\ x^i$$

Exemple

- ✓ 01010111 + 10000011 = 11010100 → Représentation en Binaire.
- ✓ {57} + {83} = {d4}→ Représentation en Hexadécimal.
- ✓ $(x^6+x^4+x^2+x+1) + (x^7+x+1) = x^7+x^6+x^4+x^2$→ Représentation polynômiale.

Remarque : La soustraction et l'addition sont les mêmes [ARM 11], [DUT 11] et [DAR 99].

2. Multiplication

Dans la représentation polynomiale, la multiplication dans le corps GF(2^8) correspond à la multiplication des polynômes modulo un polynôme irréductible de degré 8.Un polynôme est irréductible s'il n'a pas des diviseurs autres que le un « 1 » et lui-même.

Pour l'AES, ce polynôme est appelé m(**x**) et donné par :

$M(x) = x^8+x^4+x^3+x+1$, ou X"11B". →Avec la représentation en hexadécimale.

Exemple:'57 '×'83' = 'C1' Car :

$(x^6+x^4+x^2+x+1) \times (x^7+x+1) = x^{13}+x^{11}+x^9+x^8+x^7+x^7+x^5+x^3 + x^2 +x+x^6+x^4+x^2+x+1$

$= (X^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3 + 1)$

$= (X^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3 + 1)$ modulo $(x^8+x^4 + x^3+x+1) = X^7+x^6+1$.

- ✓ Le résultat est un polynôme binaire de degré inférieur à 8.

Remarque : La multiplication définie ci-dessus est associative et admet un élément neutre ('01') [ARM 11], [DUT 11] et [DAR 99].

2.2.4. Multiplication par « X », (par (00000010) ou {02})

La multiplication d'un byte b par « **x** » dans le corps GF (2^8) est notée « **xtime ()** »et peut être appréhendée comme un décalage d'un seul bit vers la gauche, si son bit de poids fort est égal à 0, sinon elle est égale à la valeur elle-même décalée d'un seul bit vers la gauche, suivi d'un XOR avec la valeur {1B}. Cette multiplication peut s'utiliser pour simplifier des calculs [ARM 11], [DUT 11] et [DAR 99].

Exemple : {57} ×{13} = {fe}. Car :

1. {57} × {02} = xtime ({57}) = {ae}
2. {57} ×{04} = xtime ({ae}) = {47}
3. {57} × {08} = xtime ({47}) = {8e}
4. {57} × {10} = xtime ({8e}) = {07}

Alors,{57} ×{13} = {57}×({01} + {02} + {10}) = {57} + {ae} + {07}= {fe}.

ANNEXE B

Detail de la S-Box et son inverse

1- La S-Box

```
const F256 SBox[256] = {
0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE,
0xD7, 0xAB, 0x76, 0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4,
0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0, 0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7,
0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15, 0x04, 0xC7, 0x23, 0xC3,
0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75, 0x09,
0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3,
0x2F, 0x84, 0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE,
0x39, 0x4A, 0x4C, 0x58, 0xCF, 0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85,
0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8, 0x51, 0xA3, 0x40, 0x8F, 0x92,
0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2, 0xCD, 0x0C,
0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19,
0x73, 0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14,
0xDE, 0x5E, 0x0B, 0xDB, 0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2,
0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79, 0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5,
0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08, 0xBA, 0x78, 0x25,
0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86,
0xC1, 0x1D, 0x9E, 0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E,
0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF, 0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42,
0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
};
```

2- inverse S-Box

```
const F256 Inv_SBox[256] = {
0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81,
0xF3, 0xD7, 0xFB, 0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E,
0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB, 0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23,
0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E, 0x08, 0x2E, 0xA1, 0x66,
0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25, 0x72,
0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65,
0xB6, 0x92, 0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46,
0x57, 0xA7, 0x8D, 0x9D, 0x84, 0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A,
0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06, 0xD0, 0x2C, 0x1E, 0x8F, 0xCA,
0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B, 0x3A, 0x91,
0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6,
0x73, 0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8,
0x1C, 0x75, 0xDF, 0x6E, 0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F,
0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B, 0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2,
0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4, 0x1F, 0xDD, 0xA8,
0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93,
0xC9, 0x9C, 0xEF, 0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB,
0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61, 0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6,
0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D
};
```

ANNEXE C

1. Architecture du processeur Microblaze

Le processeur MicroBlaze de Xilinx est un processeur RISC de 32 bits [MSR 02]. Son code VHDL est fermé. Ses caractéristiques se résument comme suit :

- L'architecture du MicroBlaze est une architecture « Harvard » avec ses bus d'instructions et bus de données séparés.
- Ses instructions sont de 32 bits.
- Il possède 32 registres de 32 bits à usage général.
- Ses bus d'adresse sont également de 32 bits.
- Il possède un « pipeline » de 5 étages.

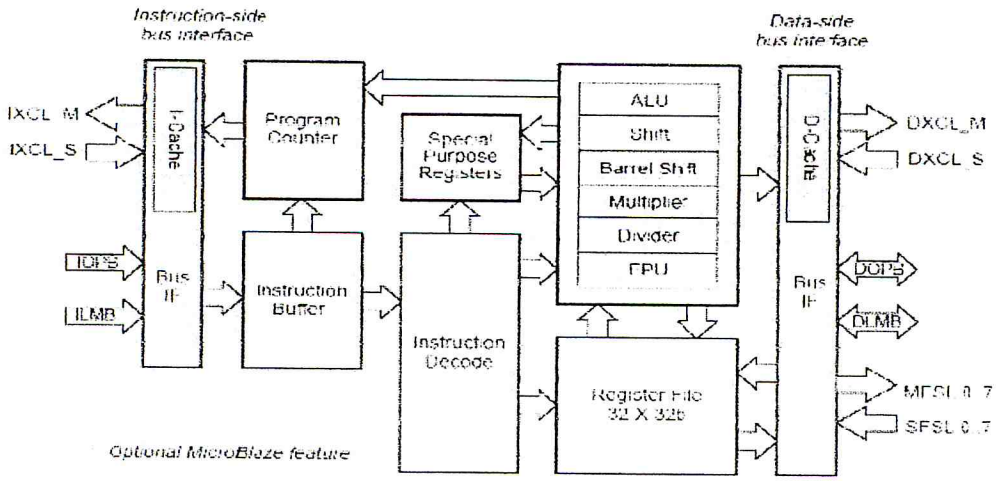


Figure C.1. Architecture de Microblaze [MSR 02].

Les parties en gris sont optionnelles et montrent à quel point ce processeur est configurable. Les mémoires Cache permettent au processeur de gagner en temps d'exécution en récupérant des données directement à l'intérieur sans aller les chercher dans la mémoire. Le «Barrel Shift», le «Multiplier», le «Divider» et le «FPU: Floating Point Unit » permettent d'accélérer des traitements de données.

Le tableau suivant illustre la fréquence du fonctionnement varie selon le circuit utilisé

FPGA	Fréquence
Virtex-II	125Mhz
Virtex-II Pro	150Mhz
Virtex-4	180Mhz
Virtex-5	235Mhz

Tableau C.1. Fréquence du processeur Microblaze sur quelques circuits.

Généralement, l'interconnexion entre l'ensemble des composants qui constituent un SoPC est assurée par un système de bus bien déterminé. A cet effet, dans l'environnement ou il est implémenté, Microblaze possède quatre types de bus de communication.

1.1. On-Chip Peripheral Bus (OPB)

Le bus OPB, conçu par IBM pour ses microcontrôleurs PowerPC, permet de lier plusieurs maîtres à plusieurs esclaves. Il autorise un maximum de 16 maîtres et un nombre d'esclaves illimité selon les ressources disponibles. Xilinx conseille néanmoins un maximum de 16 esclaves. Ce bus permet d'ajouter des périphériques au processeur MicroBlaze dont les besoins en communication seront faibles.

1.2.Local Memory Bus (LMB)

Le bus LMB est un bus synchrone utilisé principalement pour accéder aux blocs RAM inclus dans le circuit FPGA. Il utilise un minimum de signaux de contrôle et protocole simple pour s'assurer d'accéder à la mémoire rapidement.

1.3.Fast Simplex Link (FSL)

Le processeur MicroBlaze comporte 8 liens d'entrées/sorties FSL. Le bus FSL est un moyen rapide de communication entre le processeur et une autre entité. Chaque lien FSL est unidirectionnel (simplex) et met en œuvre une FIFO (pour stocker les données) et des signaux de contrôle (FULL, EMPTY, WRITE, READ,...). Il met aussi à la disposition du développeur plusieurs fonctions intéressantes dont les plus utilisées sont : *"microblaze_bwrite_datafsl"* et *"microblaze_bread_datafsl"*. Ces deux fonctions permettent d'échanger des données entre différents microblazes, par exemple, en utilisant la FIFO déjà intégrée dans le bus FSL. Ces deux fonctions sont bloquantes; *bwrite* se bloque lorsque la FIFO du bus FSL est saturée et *bread* se bloque lorsque la FIFO est vide. Les communications sur les liens FSL se font très simplement grâce à des instructions prédéfinies. Elles peuvent atteindre les 300 Mo/s à 150 Mhz.

1.4.Xilinx Cache Link (XCL)

Le lien XCL est un lien FSL particulier, dédié à la connexion d'un contrôleur mémoire externe avec la mémoire cache interne. Ceci permet au contrôleur de cache, de ne pas être ralenti par la latence du bus OPB.

1.5. La mémoire BRAM

Les blocs RAM sont des composants configurables de taille limitée par la capacité du circuit FPGA. Celle-ci peut être de 8 Kbits, 16 Kbits, 32 Kbits, ou 64 Kbits. Le fonctionnement des BRAMs est similaire aux RAMs existant dans un ordinateur. Ils servent à stocker les codes des programmes à exécuter de façon organisée. Les données et les instructions, dans le processeur Microblaze, sont stockées séparément dans deux blocks mémoires. La BRAM sert aussi à stocker le noyau (Kernel) du système d'exploitation. Elle peut être configurée à partir des Blocs select RAM de 18 Kbits qui se trouvent dans le circuit FPGA. Comme elle peut être implémentée sur CLB (Configurable Logic Block).

1.6. Périphériques du processeur Microblaze

De nombreux périphériques sont fournis avec MicroBlaze, afin de constituer un système complet et personnalisable. Il y a, entre autres :

- Contrôleur mémoires (SRAM, Flash)
- UART (Universal Asynchronous Receiver/Transmitter)
- Timer/compteur
- Interface SPI
- Contrôleur d'interruptions
- GPIO (entrées-sorties génériques)
- Convertisseurs A/N et N/A
- DMA (Direct Memory Access)

De plus, des périphériques payants sont proposés en version d'évaluation.

- UART 16550
- Interface Ethernet
- Interface PCI

2. Modèles d'implémentations des SoPCs

Chaque famille d'architecture est basée sur un modèle de traitement. Un modèle (SW) est basé sur une exécution séquentielle d'un algorithme par un processeur. Un modèle (HW), pour les circuits dédiés, offre l'option d'exécuter des tâches non parallélisables par

un processeur. Un troisième modèle est une combinaison entre les deux modèles (SW) et (HW) qui est le modèle dit Co-Conception.

2.1. Implémentation logicielle (SW)

Dans ce cas, le modèle de traitement est séquentiel. Les architectures basées sur ce modèle utilisent un ou plusieurs processeurs embarqués qui exécutent un ou plusieurs programmes définissant les opérations à réaliser et les données à récupérer de la mémoire.

En général, cette implémentation n'utilise qu'un faible nombre de ressources de calcul et des registres qui sont réutilisées dans le temps. Ces architectures sont généralement très flexibles, mais ne permettent pas d'atteindre des performances élevées.

La figure 3.7 illustre l'exécution des opérations de l'approche software.

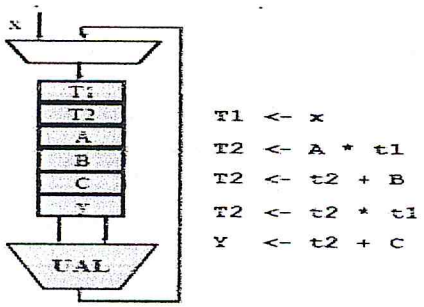


Figure C.2. Exécution séquentielle.

2.2. Implémentation matérielle (HW)

Les circuits FPGA sont basés sur l'approche hardware. Dans ce cas, afin d'exploiter le parallélisme de l'application, chaque opérateur traite des opérandes directement acheminés sur ses entrées. Dans certaines plateformes d'implémentation matérielle, la diversité des IPs fournis dans une bibliothèque permet d'implémenter des fonctions complexes, avec des performances élevées. Néanmoins, la flexibilité reste faible.

La figure suivante montre l'exécution des opérations dans l'approche hardware.

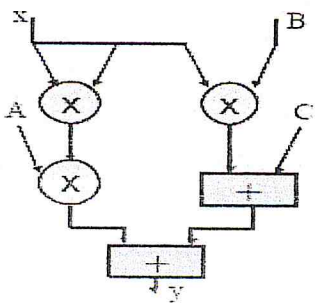


Figure C.3. Exécution parallèle.

2.3. Implémentation Co-conception (SW/HW)

Les architectures reconfigurables sont également basées sur cette approche. Afin d'apporter une souplesse supérieure aux circuits dédiés, l'utilisation de cette approche nous permet d'exploiter les avantages respectifs des matériels et logiciels.

Les caractéristiques de ce modèle de traitement confèrent une flexibilité certaine au matériel qui peut s'adapter à n'importe quelle application.

Pour mieux illustrer ce concept de développement, un exemple d'une architecture simple composée d'un microprocesseur et d'un coprocesseur est montré sur la figure 3.9.

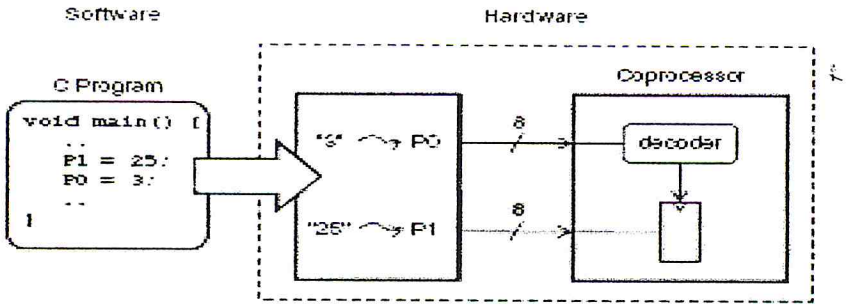


Figure C.4. Exemple d'implémentation en co-conception

La communication entre les deux composants est établie grâce aux deux ports P0 et P1. Ces derniers permettent d'accéder au coprocesseur de l'extérieur en utilisant un programme décrit en langage machine.

2.4. Conception du système et cycle de développement

L'implémentation des applications sur des plateformes SoPC offre plusieurs avantages. On peut citer entre autres : l'ajout de son propre IP (IP personnalisé) au système embarqué. Cet IP sera implémenté sur du matériel et peut en effet être considéré comme un coprocesseur par rapport au processeur principal. D'une manière générale, le cycle de conception d'un SoPC sur FPGA à base du processeur Microblaze comporte trois étapes :

- Configuration de Microblaze et des périphériques.
- Ajout du périphérique personnalisé.
- Programmation de la partie logicielle et chargement du système sur le circuit FPGA.

Le flot de conception des systèmes embarqués sur les FPGA de Xilinx est résumé sur la figure suivante :

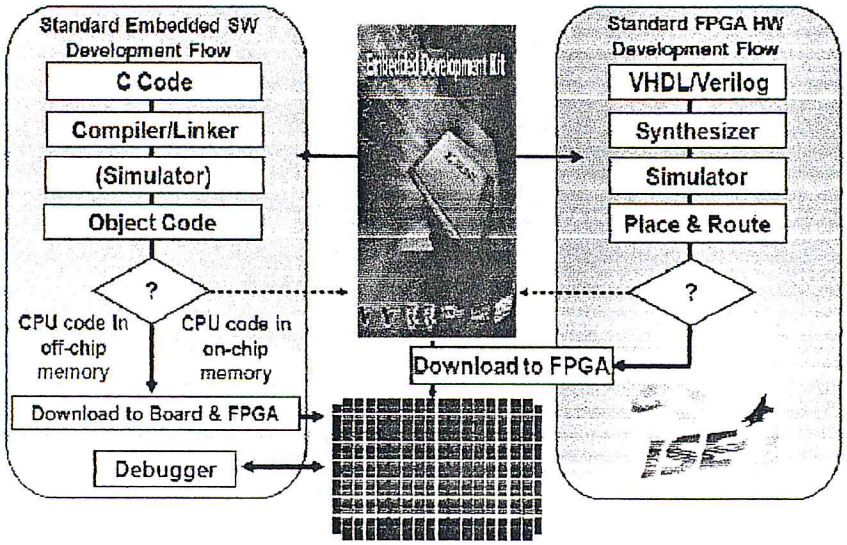


Figure C.5. cycle de conception de Xilinx.

2.4.1. Configuration de Microblaze et des périphériques

Dans cette première étape, on procède en premier lieu dans XPS par :

- Choisir Microblaze en tant que processeur principal.
- Définition de la fréquence système. Sur la carte de la bonne référence, il existe deux quartzs qui permettent de fournir deux fréquences, à savoir 100 Mhz et 24 Mhz.
- La polarisation du signal d'initialisation système (actif à l'état haut ou bas).
- Sélection des périphériques de base, tels que, l'UART, un module pour déboguer (en software, ou en hardware, ou pas), …etc.
- Sélection de la taille de la mémoire interne BRAM (64 kbits, 32 kbits, 16 kbits ou 8 kbits).
- Sélection ou pas de l'unité arithmétique à virgule flottante et de la mémoire cache.

Une fois avoir achevé cette configuration, il sera question ensuite de sélectionner les composants de la carte à inclure dans l'application, tels que le port RS232, les leds, la mémoire externe …etc. A ce niveau de configuration, d'autres périphériques peuvent être ajoutés, en outre, un timer ou/et un contrôleur de mémoire. Bien que ces derniers, peuvent aussi être ajoutés lors de la conception.

A la fin de cette première étape, XPS génère trois fichiers principaux, en l'occurrence, system.MHS (Micropocessor Hardware Spécification), system.MSS (Micropocessor Software Spécification) et system.ucf. Le premier fichier définit l'architecture matérielle de la plate forme, les connexions et les adresses mémoire de chaque périphérique. Le second fichier comporte les noms des pilotes (drivers) associés à chaque périphérique et leur

version. Le dernier fichier porte les informations concernant l'emplacement des signaux d'entrées/sorties sur les pins du circuit FPGA.

2.4.2. L'ajout d'un IP personnalisé

Dans cette seconde étape du cycle de conception, il est question de mettre en œuvre sur la plateforme SoPC son propre IP, une fois que celui-ci a été conçu et vérifié dans ISE. Généralement, le transfert des données entre les deux parties (processeur-IP) nécessite un système de communication qui assez complexe. Ce dernier est nommé *interface hardware/software* et qui a pour objectif, de réaliser la communication entre l'IP et le software exécuté par le processeur. En effet, l'implémentation d'un IP autour d'un processeur peut se faire en utilisant l'une des deux méthodes suivantes :

- Une implémentation en coprocesseur (coprocesseur interface).
- Une implémentation via la mémoire et le bus système (memory mapped interface).

Xilinx offre sur Microblaze une interface FSL (Fast Link Simplex) qui sont des ports de communication point-à-point **[SCH 10]**. De ce fait, la liaison avec l'IP sera réalisée sans passer par le bus système, comme le montre la figure suivante.

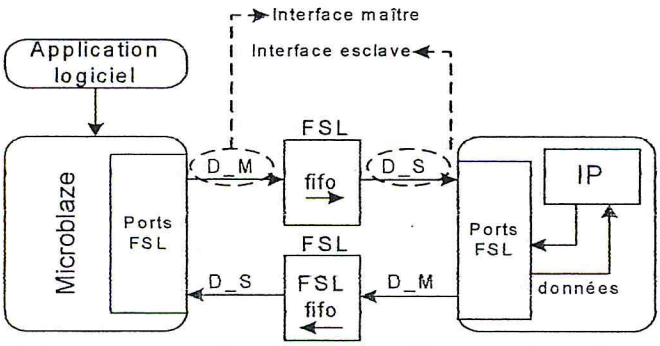


Figure C.6.Connexion d'un IP par l'utilisation d'une liaison FSL.

Dans la seconde approche d'implémentation, un espace mémoire est alloué à l'IP, pour permettre la communication entre ce dernier et le processeur. L'identification de l'IP par le processeur est effectuée par la déclaration de ses adressages dans la partie logicielle.

En général, ce type de communication est le plus répandu, bien qu'il soit considéré plus lent et plus complexe. Cette complexité relève principalement des intervenants mis en exécution. A savoir la mémoire et le bus système, comme ceci est montré sur la figure 3.10.

En plus du bus système, cette configuration nécessite l'utilisation d'une interface entre ce dernier et l'IP. Cette interface nommée IPIF (IP Interface) a pour objectif de gérer le transfert des données entre l'IP, le bus, le décoder et le protocole de communication du bus **[TUT 02] et [OPB 05]**.

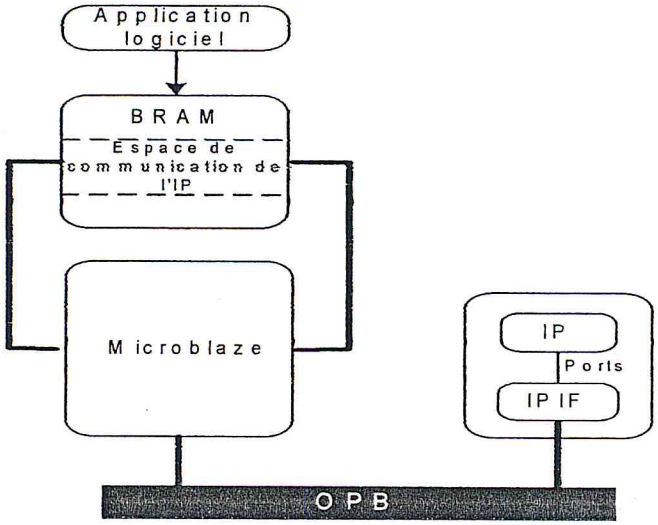


Figure C.7. Ajout d'un IP personnalisé au bus système

2.4.3 Développement de la partie logicielle et chargement du système sur FPGA.

Cette étape est réalisée dans l'outil SDK. Elle consiste à décrire la partie logicielle de l'application. Les points essentiels à développer dans cette troisième étape sont :

- Réalisation d'une API (Application Programming Interface) qui est une interface de programmation de l'application.
- Compilation de l'API.
- Chargement du ficher de configuration (bitstream) sur le circuit FPGA et visualisation des résultats sur un HyperTerminal.
- Si nécessaire, vérification du fonctionnement du circuit.

2.4.3.1 Interface de programmation de l'application.

L'API joue un rôle considérable de la partie SW. Celle-ci constitue une passerelle entre l'OS et les différents périphériques du processeur. Elle est composée principalement par les pilotes des différents IPs intégrés dans le système.

Sa programmation peut être effectuée en langage C ou C++, où il sera question de spécifier les fonctions qui permettent d'accéder aux périphériques. A titre d'exemple, la communication entre l'UART et l'IP personnalisé est effectuée en déclarant des variables intermédiaires entre les fonctions de l'UART et celles de l'IP.

2.4.3.2 Compilation de l'API

Cette étape consiste à convertir le programme C/C++ de l'application en un langage très proche du langage du processeur. Cette conversion n'est rien d'autre qu'une compilation de l'API.

Généralement, la configuration du circuit FPGA par son bitstream se déroule avant l'initialisation de la mémoire par l'exécutable de l'API.

3. Les systèmes d'exploitation compatibles avec Microblaze

Dans le but d'assurer la gestion des périphériques, plusieurs types de système d'exploitation OS (Operating System) sont compatibles avec le processeur Microblaze. Ces systèmes peuvent gérer les accès mémoire, la communication via le port RS232, etc.

Dans ce qui suit, quelques systèmes d'exploitation pouvant être embarqués avec le processeur Microblaze seront présentés tels que : Xilkernel, uClinux et Asterix.

Chaque système a ses propres caractéristiques, en termes de vitesse d'exécution, de la taille mémoire occupée et des fonctionnalités offertes [RON 06].

3.1. uClinux

uClinux (prononcé "you-see-linux") est un type de Linux standard, destiné aux microprocesseurs qui n'ont pas une unité de gestion mémoire (MMU).

L'unité de gestion mémoire permet la traduction des adresses logiques en adresses physiques. Elle décide si la donnée est dans la RAM ou dans le disque dur et empêche le processeur à accéder directement à la mémoire.

Sans présence de cette unité, le programmeur est sensé faire la gestion de mémoire pour qu'il garde la cohérence entre les différents processus en utilisant vfork( ) qui est similaire au fok() de Linux.

uClinux est compatible avec plusieurs types de processeurs, Motorola Coldfire, Dragonball, Blackfin, ARM7TDMI et MicroBlaze.

uClinux contient deux package : le premier est le noyau (Kernel) du système et le second package contient toutes les bibliothèques et les libérés en C qui permettent la gestion des périphériques.

3.2.Asterix

Astérix est un petit noyau temps réel développé par Mälardalen Real-Time Research Center (MRTC), situé à l'Université de Mälardalen (Suède) où Il est enseigné et utilisé dans différents projets de recherche.

Astérix contient un environnement de débogage et fournit le soutien à une mesure à haute résolution de temps d'exécution des tâches et permet aussi l'exécution multitâches.

3.3. Xilkernel

Xilkernel est un noyau de petite taille, robuste et compatible avec les processeurs Microblaze et Power PC. L'OS en question est intégré dans EDK.

Xilkernel permet à l'utilisateur par exemple d'engager des gestionnaires d'interruption et peut être utilisé pour implémenter des applications à un niveau d'abstraction élevé, telle que les applications qui relèvent de la video, audio et réseaux.

3.4. Standalone

Cet OS est un système d'exploitation basic. II est livré par défaut sur EDK et comporte les librairies d'entrées/sorties pour les pilotages des IPs, tel que l'UART, BRAM, Timer... etc. Dans ce travail, nous nous sommes limités à l'utilisation de cet OS.

Références des Chapitres

Web graphies

[W01] . http://www.williamstallings.com/Crypto3e.html

[W02] . http://www.iaik.tugraz.at/content/research/krypto/AES/

[W03] . http://www.apprendre-en-ligne.net/crypto/rsa/index.html

[W04] . http://www.hsc.fr. (2001)

Bibliographies

[ALI 10] . H. ALIREZA , V. INGRID. « A 21.54 Gbits/s Fully Pipelined AES Processor on FPG » . Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04) IMECS , VOL.2 . (2010) .

[ALM 09]. Z. ALAOUI ISMAILI, A. MOUSSA . « Self-Partial and Dynamic Reconfiguration Implementation for AES using FPGA » . International Journal of Computer Science (IJCSI), Vol. 2, 2009

[ARM 11] . A, ARMOUSH . « Chapter 4: Advanced Encryption Standard ». (2011).

[BAR 13]. R. BAHRAM, M. RAJARAM. « FPGA Based A New Low Power and Self-Timed AES 128-bit Encryption Algorithm for Encryption Audio Signal » . (2013).

[BER 98]. A. UC Berkeley and USC ISI . «The network simulator NS-2 ». *Part of the VINT project*, http: //www.isi.edu/nsnam/ns. (1998).

[BSQ 08]. P. Bulens F.-X. Standaert J.-J. Quisquater P. Pellegrin G. Rouvroy. « Implementation of the AES-128 on Virtex-5 FPGAs ». (Juin, 2008).

[BUT 01]. P. BUTEL. « SoC : A New Approach to Enhance System Performances and to Combat the Long-Term Availability Issue». (Janvier 2001).

[CAN]. A. CANTEAUT. « La cryptologie moderne ». INRIA Projet CODES. http://www-rocq.inria.fr/~canteaut/crypto_moderne.pdf

[CHC 08]. J. CHANG, C-W. HUANG , K-H. CHANG, C-C.HSIEH. « Hight Throughput 32 bits AES Implementation in FPGA » . (2008).

[CHO 03]. K. AJ, P. CHODOWIEC. « very compact FPGA of the AES algorithme ». Vol; 2779 . (2003).

[CHO 05]. S. CHOOMCHUAY, S. PONGYUPINPANICH and S. PATHUMVANH. «A Compact 32-bit Architecture for an AES System» . ECTI TRANSACTIONS ON COMPUTER AND INFORMATION THEORY VOL.1, NO.1 . (Mai, 2005).

[CHS 08]. J.H. CHEN , M.D SHIEH. « Exploration of Low-Cost Configurable S-box Designs for AES Applications ». Embedded Software and Systems, ICESS '08. IEEE International Conference, pp. 422-428. (2008).

[DAR 99]. J, Daemen , V, Rijmen . « AES Proposal: Rijndael » . version N°2 . (1999). http://csrc.nist.gov/CryptoToolkit/aes/

[DBS 06]. J.P Deschamps, G.J.A Bioul, G.D. Sutte. r« *Synthesis Of Arithmetic Circuits Fpga, Asic, and Embedded Systems* », A John Wiley & Sons. (2006).

[DUT 11]. J.M, DUTERTRE . « Synthèse AES 128 » .(2011).

[ELR 04]. S. El ADIB and N. RAISSOUNI. « AES Encryption Algorithm Hardware Implementation Architecture: Resource and Execution Time Optimization » . International Journal of Information & Network Security (IJINS) Vol.1, No.2, pp. 110~118 ISSN: 2089-3299. (Juin, 2012).

[ENG 06]. A, ENGE . « La méthode RSA et la cryptographie fondée sur les courbes elliptiques Cryptologie » . (2006).

[EST]. « *Embedded System Tools Reference Manual, Embedded Development Kit* », EDK 9.1i

[FEI73]. H. FEISTEL. « Cryptography and Computer Privacy ». Scientific American; 228(5); p.15-23. (1973).

[FIP 01]. « Announcing the ADVANCED ENCRYPTION STANDARD (AES) » . Federal Information Processing Standards Publication 197. (26 Novembre, 2001).

http://csrc.nist.gov/publications/

[GAF 13]. M. GNANAMBIKA, S. ADILAKSHMI, Dr. FAZAL NOORBASHA. «AES-128 Bit Algorithm Using Fully Pipelined Architecture for Secret Communication » . (IJERA) ISSN: 2248-9622 ,Vol. 3, Issue 2, pp.166-169 . (Mars -Avril 2013).

[GOB 06]. T. GOOD, M. BENAISSA. « very small FPGA application-specific processor for AES ». IEEE. (2006).

[GVM 10]. J-M. GRANADO-CRIADO, M.A. VEGA-ROBRIGUEZ, J-M.SANCHEZ-PEREZ, J.A. GOMEZ. « very A new methodologie to implement the AES algorithm using partial and dynamic reconfiguration » . pp. 72- 80. (2010).

[HAM 10]. I. HAMMAD. « Efficient Hardware Implementations for the Advanced Encryption Standard algorithm » . (2010).

[HAS 10]. F. Haghighizadeh, H. Attarzadeh, M. Sharifkhani. «A Compact 8-bit AES Crypto-Processor». Second International Conference on Computer and Network Technology, IEEE. (2010).

[HDH 05] . A. Hodjat, D. David, Hwang, B- G. LAI, K. TIRI, I VERBAUWHEDE. « A 3.84 Gbits/s AES Crypto Coprocessor with Modes of Operation in a 0.18-ÿm CMOS Technology ». *GLSVLSI'05*, Chicago, Illinois, USA. Copyright ACM. (Avril, 2005).

[HOV 04]. A. HODJAT, I. VERBAUWHEDE. « A 21,54 Gbits /s fully pipelined AES processor on FPGA » . pp. 308- 309 . (2004).

[IBR 10] . K, IBRAHIMI . « Sécurité des Systèmes d'Information et des Réseaux : Cryptographie». Université d'Avignon et des Pays de Vaucluse CERI, M1-M2 Alternance. (2010).

[ISS 10]. M. H. ISSAM. « Efficient Hardware Implementations For The Advanced Encryption Standard (AES) Algorithm ». (2010).

[JAG 09] . A.JAGADEV. « Advanced Encryption Standard (AES) Implementation ». Bachelor of Technology THESIS, Department of Electronics and Communication Engineering National Institute of Technology, Rourkela. (Mai 2009).

[LAB 01]. G. LABOURTE: «Introduction a la Cryptographie ». (2001).

[LAU 10]. C, LAURADOUX . « Une introduction à la cryptographie Cours I: cryptographie symétrique ». (Janvier 2010).

[LIL 05]. H. Li and J. Li . « A high performance sub-piplined architecture for AES » . Computer Design (ICCD 05), pp. 491-496 . (2005).

[MAX]. C. MAXFIELD, «*FPGAs World Class Desogn*» ,Newnes.

[MSR 02]. « *MicroBlaze Software Reference Guide* » . (2002).

[MUR 09]. H. MUHAMMAD, M. RAIS and SYED. « A Novel FPGA Implementation of AES-128 using Reduced Residue of Prime Numbers based S-Box » . International Journal of Computer Science and Network Security (IJCSNS), VOL.9 No.9. (Septembre 2009).

[MUR 10]. H. MUHAMMAD, M. RAIS and SYED. « Efficient FPGA Realization of S-Box using Reduced Residue of Prime Numbers». International Journal of Computer Science and Network Security (IJCSNS), VOL.10 No.1. (Janvier, 2010).

[NAI 08]. S. NAZIRI et N. IDRIS. « The memory-less method of generating multiplicative inverse values for S-box in AES algorithm», Electronic Design, ICED. IEEE International Conference, pp. 1-5.(2008).

[NIST 01] . National Istitude of Standard and Technology (NIST). « Advanced Encryption Standard (AES) ». FIPS-197. (2001).

[OPB 05]. « *OPB IPIF* », DS414,v3.01c. (2005).

[KAK 13]. R A. KAUR, N. KUMER, P. BHARDWAJ. « FPGA Implementation of Efficient Hardware for the Advanced Encryption Standard ». (IJITEE) ISSN: 2278-3075, Volume-2, Issue-3, (2013).

[KER83]. A, Kerckhoffs . « La cryptographie militaire ». *Journal des sciences militaires.* Vol. IX, pp. 5-83 (Janvier 1883) et pp.161-191. (Février 1883).

[RAB 00]. J. RABAEY. « Silicon Platforms for the Next Generation Wireless Systems – What Role does Reconfigurable Hardware Play? » . *Proceedings FPL2000*, Austria. (2000).

[ROL 02]. B, ROLLA ND. « Principaux algorithmes de cryptage ». Department of Computer Science SEPRO Robotique. (Juillet 2002).

[RON 06]. A.RONNHLM. «*Evaluation of Real-Time Operating Systems for Xilinx MicroBlaze CPU* », Performed for ABB Corporate Research. (2006).

[RPZ 10]. T.RAHMAN, S. Pan, Q. ZHANG. «Design of a High Throughput 128-bit AES (Rijndael Block Cipher) ». IMECS. Vol 2. (2010).

[RSQ 04]. G. ROUVRROY, F-X. STANDART, J-J. QUISQUATER, J-D. LEGAT. « very compact and efficient encryption / decryption of the AES Rijndael very well suited for small embedded applications » . (2004).

[TUT 02]. «*Tutorial: Designing Custom OPB Slave Peripherals for MicroBlaze* ». (2002).

[SCH 10]. P. R. SCHAUMONT . « *A Practical Introduction to Hardware/Software Codesign*», Springer. (2010).

[SKS 05] . N. SKLAVOS, O. SELIMIS GAND KOUFOPAVLOU. « FPGA implementation cost and performance evaluation of IEEE 802.11 protocol encryption security schemes » . Institute of Physics Publishing Journal of Physics: Conference Series 10 . (2005).

[SNS 12]. N. SHYLASHREE, B. NAGARJUN, V. SHRIDHAR. «FPGA IMPLEMENTATIONS OF ADVANCED ENCRYPTION STANDARD: A SURVEY » . International Journal of Advances in Engineering & Technology Issues (IJAET).(Mai 2012).

[SSG 05]. N. Sklavos, O. Selimis Gand Koufopavlou « FPGA implementation cost and performance evaluation of IEEE 802.11 protocol encryption security schemes ». Institute of Physics Publishing Journal of Physics: Conference Series 10 . (2005).

[SWM 06]. R.Saleh, S.Wilton, S. Mirabbasi, A.Hu , M. Greenstreet, G. Lemieux, P.P.Pande,C.Grecu, A. Ivanov, « *System-on-Chip: Reuse and Integration*», Proceedings of the IEEE | Vol. 94,No. 6. (2006).

[TAB 10]. R. TABARY. « Cryptologie symétrique Cryptologie » . (2010).

[VPP 11]. G.VENKATESAN, J. PROF, R P. PERINBAM. « A New Reconfigurable Hardware Architecture for Cryptography Applications using AES by different Substitution box (S-Box) and Random Round Selection ». International Journal of Computer Science and Network Security (IJCSNS), VOL.11 No.12. (Decembre, 2011).

[ZEC 05]. P.M. Zeitzoff et J.E. Chung. « A perspective from the 2003 ITRS », *IEEE* circuits & devices magazine. (Janvier/Février 2005).