**REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE**

**MINISTERE DE L'ENSEIGNEMENT SUPERIEUR**

**ET DE LA RECHERCHE SCIENTIFIQUE**

**UNIVERSITE SAAD DAHLAB DE BLIDA 1**

**FACULTE DES SCIENCES**

# PROJET DE FIN D'ETUDE

**POUR L'OBTENTION DU DIPLÔME DE MASTER EN INFORMATIQUE**

**SPECIALITE : Sécurité des Systèmes d'Information**

# THÈME

# Implémentation embraquée d'un nouvel algorithme de multiplication scalaire pour les applications de cryptage basée sur les courbes elliptiques

**Mémoire réalisé par :**

Mr CHABOU Tarek

Mr BELKHELFA Billel

Soutenue le : 10/09/2020, devant le jury composé de :

- Président : Mr BENYAHIA Mohamed
- Examinateur : Mr C H E R I F Z a h a r
- Promotrice : Mme BOUSTIA Narihmane
- Encadreur : Mr BAKIRI Mohamed
- Co-Encadreur : Mr OUDJIDA Abdelkarim

**Année Universitaire :2019 - 2020**

# Remerciements

*Nous tenons en premier lieu à exprimer nos profondes gratitudes et nos vifs remerciements à Dieu qui nous a donné la force et la patience d'accomplir ce modeste travail.*

*Nous exprimons notre grande gratitude à notre promotrice Mme. BOUSTIA N, d'avoir accepté de nous encadrer durant notre travail et pour ses précieux conseils et ses orientations.*

*Nous remercions très chaleureusement Mr. BAKIRI M et OUDJIDA K pour son soutien, ses précieux conseils tout au long de notre stage au niveau du CDTA. Nous avons eu le privilège de travailler parmi votre équipe et d'apprécier vos qualités et vos valeurs, votre sérieux et votre compétence.*

*Nous remercions sincèrement les membres de jury qui nous avoir fait l'honneur de juger ce travail.*

*Nos remerciements les plus chaleureux vont vers toute nos familles, nos parents pour leurs encouragements et leur soutenue depuis toujours.*

***Merci !!!***

# Dédicaces

Je dédie ce travail qui n'aura jamais pu voir le jour sans les soutiens indéfectibles et sans limite de mes chers parents qui ne cessent de me donner avec amour le nécessaire pour que je puisse arriver à ce que je suis aujourd'hui. Je dédie également ce modeste travail :

~ A mes très chères frères et sœurs et mon neveu Anes et ma nièce Chahed et pour m'avoir soutenu et aidé tout au long de ce projet.

~ A toute ma famille : mes grands-parents, mes tantes et mes cousins, en particulier Mohamed Dahmani.

~ A la personne qui m'a soutenu toute l'année, mon binôme et frère de cœur « Billel », merci.

~ A tous les enseignants de l'université de Blida.

~ A mes amis : Mohamed Chakib Chikhi, Mohamed Hadjersi, Riadh Lahreche, Zakaria Hamaidi, Mohamed Gherbi, Mohamed Lamine Bouzaoui, Imade Hmida, Mhamed Chabou, Fouzi Sahnoune

~ Enfin, a tous les étudiants de la promotion 2019 / 2020 SSI et notre club scientifique ITC.

TAREK

## Résumé

Depuis toujours, l'être humain a cherché à conserver certaines informations ou données secrètes, donc il a été obligé de trouver des nouvelles méthodes pour sécuriser la communication et les informations envoyées, l'une des méthodes la plus efficace est la cryptographie, au fil des années a connu des améliorations. Ces dernières années les algorithmes de cryptographie ne répondent plus à satisfaire les exigences des usagers de côté disponibilité, confidentialité et l'intégrité dans les systèmes embarques. L'une des algorithmes de cryptographie les plus utilisant est basé sur les courbes elliptique qui offre la même robustesse que RSA avec une clé plus courte. Cependant, les courbes elliptiques souffrent de certaine limitation comme l'utilisation des opérations mathématiques plus complexes avec une grande latence. Dans notre travail on s'intéresse à l'implémentation logicielle et matérielle sur un système embarqué basé sur le circuit électronique FPGA d'une nouvelle optimisation mathématique de l'opération du multiplication nommé Radix-$2^r$. Finalement, une application démonstrative de nos travaux sur le chiffrement et déchiffrement avec ECDH_ Radix-$2^r$ pour garantir plus de sécurité dans les systèmes informatique.

**Mot clé :** Cryptographie sur les courbes elliptiques, Radix-$2^r$, FPGA, Diffie-Hellman.

## Summary

Since always, the human being has tried to protect some information or data, so he has been obliged to find new methods to secure the communication and the information sent, one of this more effective method is cryptography, over the years has seen improvements but in recent years. Cryptographic algorithms no longer satisfy users requirements in terms of availability, confidentiality, integrity and authentication. Therefore, several researchers have implemented a new cryptographic algorithm based on elliptic curves because this new solution can offer the same robustness as RSA as well with a shorter key and there are also mathematical methods to improve its performance. In our work we are interested in the software and hardware implementation on an embedded system based on the FPGA electronic circuit with a new mathematical optimization of the multiplication operation called Radix-2r. Finally, a demonstrative application of our work on encryption and decryption with ECDH_ Radix-2r is to guarantee more security in computer systems.

**Key words:** Elliptic curve cryptography, Radix-$2^r$, FPGA, Diffie-Hillman.

*ملخص*

منذ ذلك الحين، حاول الإنسان الحفاظ على سرية معلومات أو بيانات معينة، لذلك اضطر إلى إيجاد طرق جديدة لتأمين الاتصال والمعلومات المرسلة، وأحد هذه الطرق الأكثر فاعلية هو التشفير، وقد شهد على مر السنين تحسينات ولكن في السنوات الاخيرة. لم تعد خوارزميات التشفير تلبي متطلبات المستخدمين من حيث التوافر والسرية والسلامة والمصادقة. هذا هو السبب في أن العديد من الباحثين قاموا بتطبيق خوارزمية تشفير جديدة تعتمد على منحنيات بيضاوية لأن هذا الحل الجديد يمكن أن يقدم نفس القوة مثل RSA مع مفتاح أقصر بكثير وهناك أيضًا طرق رياضية لتحسين أدائها. نحن مهتمون في عملنا، نحن مهتمون بتنفيذ البرامج والأجهزة على نظام مضمن يعتمد على الدائرة الإلكترونية FPGA لتحسين رياضي جديد لعملية الضرب تسمى Radix-$2^r$. أخيرًا، تطبيق توضيحي لعملنا على التشفير وفك التشفير باستخدام _ECDH Radix-$2^r$ لضمان مزيد من الأمان في أنظمة الحاسوب.

**الكلمات المفتاحية :** تشفير المنحنى الإهليلجي , راديكس-2ر ,فبجيا ,ديفي-هيلمان.

# Sommaire

## Liste des figures

## Liste des tableaux



## Liste des algorithmes

## Liste des abréviations

| | | |
|---|---|---|
| **ECC** | : | Elliptic curve cryptography |
| **NAF** | : | Non-Adjacent Form |
| **FPGA** | : | Feld-programmable gate array |
| **CPU** | : | Central processing unit |
| **DSP** | : | Digital Signal Processor |
| **TRNG** | : | True random number generator |
| **PRNG** | : | Pseudo Random Number Generator |
| **NIST** | : | National Institute of Standards and Technology |
| **SoC** | : | System on chip |
| **AES** | : | Advanced Encryption Standard |
| **DES** | : | Data Encryption Standard |
| **ECDSA** | : | Elliptic Curve Digital Signature Algorithm |
| **ECIES** | : | Elliptic Curve Integrated Encryption Scheme |
| **ECDH** | : | Elliptic Curve Diffie-Hillman |
| **DLL** | : | Dynamic Link Library |
| **API** | : | Application programming interface |
| **ECDLP** | : | Elliptic curve discrete logarithm problem |
| **VHDL** | : | VHSIC Hardware Description Language |
| **LCA** | : | Logic Cell Array |
| **CLB** | : | Configurable Logic Block |
| **IOB** | : | Input output bloc |
| **DCM** | : | Digital Clock Manager |
| **ADC** | : | Analog to Digital Converter |
| **DAC** | : | Digital to analog conversion |
| **LUT** | : | Look-Up Table |
| **HLS** | : | High Level Structure |
| **MAC** | : | Media Access Control |
| **AXI** | : | Advanced eXtensible Interface |
| **SDK** | : | Software Development Kit |

# Introduction générale

## a) Contexte :

Depuis 1976, la cryptographie à clé publique a révolutionné les systèmes de communication utilisés par plusieurs entreprises (tel que Microsoft, Facebook, etc.), gouvernements et banques. La cryptographie à clé publique est appelée aussi la cryptographie asymétrique. Pour garder la confidentialité dans la communication, nous avons besoin de sécuriser nos messages leur transmission, C'est pour cette raison la cryptographie est indispensable. Aujourd'hui ECC est dominée dans la cryptographie Asymétrique, signature digital et généralisation des nombre pseudo-aléatoire, toutefois ECC exige un niveau de sécurité équivalent aux clés bien plus petites que RSA et si nous parlons de la cryptographie ECC, nous parlons de calculs arithmétiques complexes qui nécessitent plusieurs structures mathématiques. Pour ces nouveaux protocoles de chiffrement à clé publique, plusieurs implémentations efficaces de ces arithmétiques complexes ont été proposées au fil des années donc c'est l'une des raisons pour lesquelles les cryptosystèmes basés sur les courbes elliptiques connaissent une grande importance pour l'utilisation depuis leur proposition par Miller et Koblitz en 1985.

## b) Problématique :

Les premières utilisations des courbes elliptiques en cryptographie remontent au début des années 90. Relativement aux techniques de cryptage asymétrique tel que le RSA, les courbes elliptiques requiert une clé beaucoup plus courte, ce qui se traduit par un gain en ressources hardware, vitesse et en consommation de puissance, enjeu primordial dans les applications embarquées actuelles. Cependant, les courbes elliptiques exigent une complexité de calcul excessive due principalement à l'opération de multiplication scalaire.

## c) Objectifs :

Les cryptosystèmes à courbes elliptiques ont un grand intérêt dans différent domaine des applications. En conséquence, son implémentation est devenue intéressante.

Notre objectif est d'implémenter matérielle et logicielle sur circuit FPGA un algorithme de multiplication scalaire développé par CDTA ayant l'avantage d'être rapide, à faible consommation de mémoire et fortement sécurisé sur leurs aspects théoriques et matérielles pour une application de cryptographie effective et faire les comparaisons nécessaires (benchmarking) avec les algorithmes classiques, tels que l'algorithme binaire, Montgomery NAF, w-NAF, etc. Une application d'échange de clé basé sur le protocole Diffie-Hillman et chiffrement serons développé comme un support de validation et test de nos résultats.

## d) Organisation du mémoire :

L’ensemble de ce travail est réparti comme suit :

Dans le premier chapitre, nous commencerons par présenter la sécurité et la cryptographie dans les systèmes embarqués en général, leur objectifs et une démarche générale.

Une deuxième partie présente la cryptographie sur les courbes elliptiques et comparaison avec RSA.

Une troisième partie présente une nouvelle génération de développement sur circuit FPGA.

Le second chapitre est consacré d’une part à un rappel sur ECC et une étude de l'arithmétique de Radix-2r avec l'intégration dans ECC et d’autre part le mécanisme proposé de chiffrement et déchiffrement.

L’étape suivante de ce projet présente les travaux pratiques de l'implémentation embarqué de ECC Radix-$2^r$ logiciel et matériel dans le troisième chapitre.

Et enfin une conclusion générale sur les différentes étapes de conception et de l'implémentation embarque de ECC Radix-$2^r$ logiciel et matériel ainsi que les différents problèmes rencontrés et les solutions données.

# Chapitre 1 : La sécurité et la cryptographie dans les systèmes embarqués

## 1.1 Introduction

La sécurité informatique ou cybersécurité est une discipline consiste à protéger les informations dans un système informatique (ordinateurs, serveurs, appareils mobiles...) contre les attaques malveillantes. Le mot est très large et englobe chaque élément de la sécurité de l'ordinateur à l'activité et la formation des utilisateurs. Dans ce chapitre, nous définissons la sécurité informatique pour les systèmes embarqué, puis introduire les différentes types et algorithmes de cryptographie utilisés dans ces systèmes en citant par la suite la sécurité et la cryptographie dans les systèmes embarqués à base des circuits programmable FPGA comme application.

## 1.2 Sécurité d'information pour les systèmes embarqués

L'utilisation croissante des systèmes électroniques dans la vie de tous les gens a amené trop d'informations sensibles dans ces équipements. Maintenant nous pouvons avoir plusieurs données personnelles et confidentielles stockées dans des appareils électroniques tels que : informations bancaires (carte bancaire avec puce, sauvegardes dans l'ordinateur ou même dans le téléphone portable pour achats en lignes), mots de passe (ordinateur personnel), courriers et historique de navigation (ordinateur), musiques ou vidéos protégées par copyright, …. Ect.

### 1.2.1 Généralité sur les systèmes embarqués

Un système embarqué est un ensemble des composants matériel exécutant du logiciel (processeurs, mémoires et des circuits dédiés à des applications spécifiques).

Au début, les applications des systèmes embarqués sont utilisées dans le milieu militaire à cause de leur prix et l'avantage stratégique d'une telle technologie. Cependant, leur grande potentialité leur a rapidement ouvert les portes vers des applications civiles (i.e. pilotes automatiques pour avions de ligne, etc. …). Avec le progrès technologique leur prix sera baissé. Les facteurs clés d'un système embarqué c'est la taille, poids et consommation. Ces dernières années, tous les avantages des systèmes embarqués (vitesse, taille, consommation, etc. …) sont augmentés et aussi la complexité du développement. Un bon exemple c'est les téléphones mobiles, depuis les « valises » des années 80 jusqu'aux petit modèles actuels.

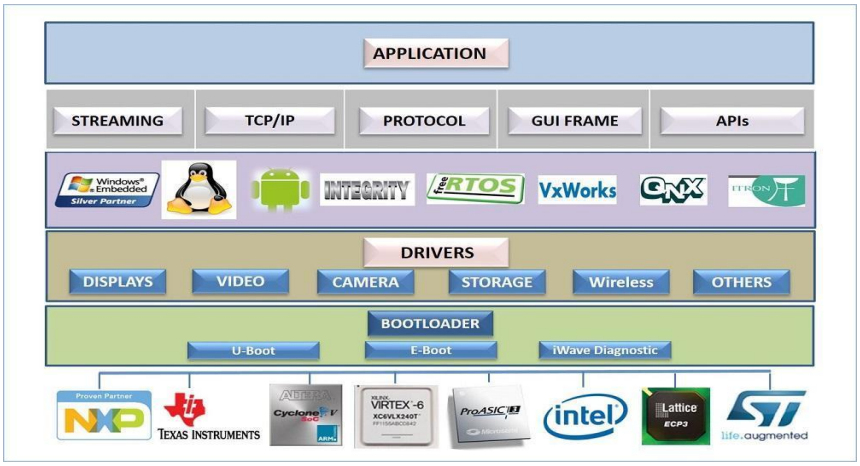



**Figure 1.1: Le hardware d'un système embarqué**

Ce phénomène est encore plus marqué par le développement des systèmes embarqués générant presque chaque jour de nouveaux outils avec de nouvelles possibilités et de nouveaux risques d'intrusion. En effet, plus la connectivité augmente et plus nous offrons de points d'entrée pour voler nos secrets : nous pouvons voir comment le standard de communication sans fils wifi le plus utilisé aujourd'hui, est intrinsèquement vulnérable aux attaques. Le principal système pour garantir la confidentialité est dans le chiffrement des données : les principes mathématiques mis en jeu sont très complexes et nous ne chercherons ici qu'à en donner un aperçu le plus complet possible sans descendre dans des détails théoriques trop complexes.

**Tableau 1.1 : Comparaison entre différent technologie utilisé dans des systèmes embarqués**

| **Traitement FPGA** | **Traitement CPU** | **Traitement GPU** | **Traitement DSP** |
|---|---|---|---|
| Traitement parallèle massif | Traitement séquentiel | Traitement parallèle limité | Traitement parallèle limité |
| La mise en œuvre est matérielle | La mise en œuvre est un logiciel | La mise en œuvre est un logiciel | Traitement parallèle limité |
| Héritage inexistant ou limité en raison du processus " pipeline" | Beaucoup d'opérations d'E / S, temps de chargement, etc | Beaucoup d'opérations d'E / S, temps de chargement, etc | Beaucoup d'opérations d'E / S, temps de chargement, etc |
| Espace limité | Espace de fonctionnement presque illimité (programme) | Dépend de la puce GPU (limité) | Dépend de la puce DSP (limité) |
| Pas bon pour certaines opérations (virgule flottante ect) | Extrêmement polyvalent | Extrêmement fort dans certaines opérations | Extrêmement fort dans certaines opérations |
| Compliqué à programmer (VHDL) | Avec des langages comme le c ++ très faciles à programme | Avec des bibliothèques d'imagerie très faciles à programmer | Pas facile de programmer un environnement de développement propriétaire |
| Test / débogage difficile | Avec une gamme d'outils pour le débogage | Quelques outils de débogage disponibles | Quelques outils de débogage disponibles |
| Besoin d'interface au logiciel | Pas besoin d'interfaçage spécial | Pas besoin d'interfaçage spécial | Doit s'interfacer avec le logiciel |

### 1.2.2 La sécurité des systèmes embarqués

Le système d'information est généralement défini par l'ensemble des données et des ressources matérielles et logicielles de l'entreprise permettant de les stocker ou de les faire circuler. Les principaux objectifs de la sécurité utilisé dans les systèmes embarqués sont : garantit que les données reçues sont exactement celles qui ont été émises par l'émetteur (L'intégrité), assurer qu'une information est accessible uniquement par les entités qui ont le droit d'accéder à celle-ci (La confidentialité), assurer que seules les personnes autorisées aient accès aux ressources (L'authentification), maintenir le bon fonctionnement du système d'information (La disponibilité), garantir qu'une transaction ne peut être niée(La non répudiation). Suite à cela, deux niveaux de protections peuvent être intégrées :

#### 1.2.2.1 Protection niveau matériel

Le niveau matériel a été le premier à être pris en compte pour la sûreté de fonctionnement et a donc atteint un degré d'évolution très élevé : comme nous le verrons dans les paragraphes suivants, la plupart des techniques appliquées à n'importe quel niveau ont été à l'origine développées pour le matériel et ensuite adaptées à d'autres niveaux. Quand nous parlons de protection matérielle, il faut bien clarifier le concept de niveau d'intervention. Ce sont des techniques qui concernent le technologue plus que le concepteur de circuits, développement de cellules technologiques intrinsèquement résistantes à travers une redondance dans la structure électrique au niveau transistors et/ou une modification des caractéristiques géométriques des transistors (dimensionnement) , protection au niveau portes : fonctions logiques et éléments mémoire de base (bascules) , modification au niveau des primitives de calcul , modification au niveau micro-architectural , modification de l'architecture globale.

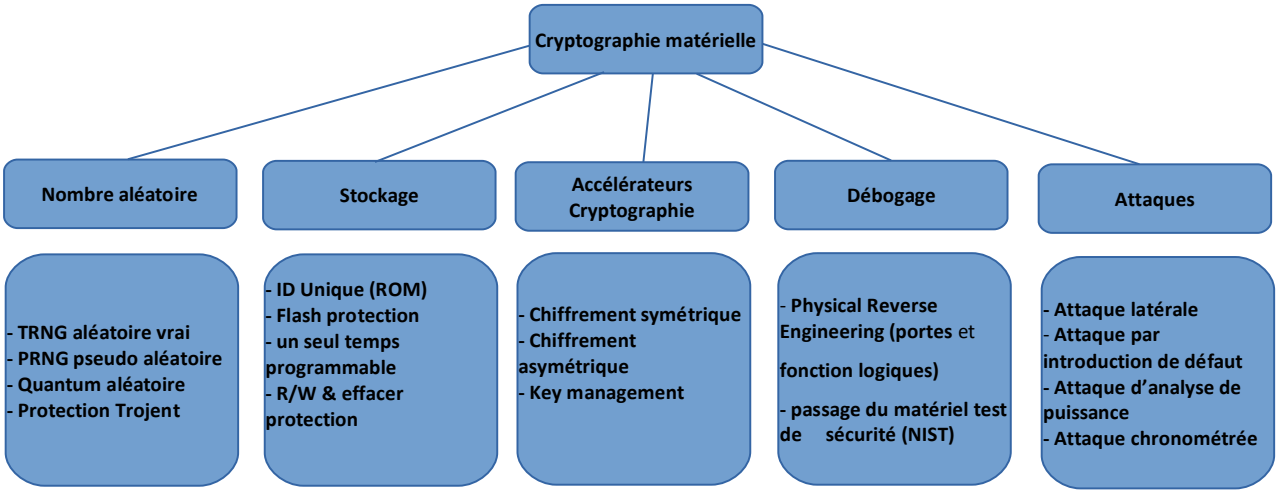



**Figure 1.2 : Différent niveaux de protection matériel dans SoC des systèmes embarqués**

La figure 1.2 précédente représente les niveaux de protection dans SoC, nous avons premièrement générateur des nombres aléatoires pour les clés, deuxièment protection de stockage, après l'accélérateur de cryptographie et débogage et à la fin les attaques qui il faut l'éviter.

#### 1.2.2.2 Protection niveau logiciel

Le logiciel est un des éléments les plus critiques dans un système, nous pouvons distinguer deux couches de protection aux niveau logicielle : En premier lieux, il y a la protection au niveau système d'exploitation qui sont très difficiles à protéger car souvent nous ne pouvons pas connaître à l'avance leur comportement, surtout si nous ne connaissons pas a priori et en détail les tâches qui vont être exécutées. Cependant leur importance dans le fonctionnement global des systèmes complets est grande et ils ne peuvent pas être oubliés. Puis, nous avons la protection niveau application : La raison principale est la grande complexité que les programmes peuvent atteindre, rendant très difficile leur vérification et leur validation. Tous les programmeurs et utilisateurs des ordinateurs savent que tous les programmes sont hantés par des erreurs de programmation ou d'algorithme pouvant causer des problèmes plus ou moins graves, être systématiques ou plutôt imprévisibles, mais ils ont le même résultat, depuis longtemps les programmes à exécuter doivent être traités comme des éléments non fiables et lorsqu'il n'est pas possible d'implanter suffisamment de protections au niveau du matériel il devient nécessaire d'implanter ces protections dans le logiciel d'application.

### 1.2.3 Cryptographie dans les systèmes embarquées

La cryptographie se divise en deux parties nettement différenciées :

#### 1.2.3.1 Cryptographie symétrique

Dans ce mode, une seule clé est utilisée pour chiffrer et pour déchiffrer le message. C'est un chiffrement à clé unique. Le transmetteur qui a le privilège de générer la clé de chiffrement doit communiquer cette clé au récepteur pour qu'il puisse le déchiffrer, nous utilisons pour cela un canal secret. La clé unique qui est utilisée par les deux parties (émetteur/récepteur) doit être changée à chaque information traitée, il faut donc plusieurs clés pour le même message. Cette règle de changement de clés permet de renforcer la sécurité en cas d'attaques pour la récupération de clé. Il existe plusieurs algorithmes de ce type de chiffrement dont : AES, DES, Triple DES ; ...etc.

Nous distinguons dans la cryptographie symétrique deux modes de chiffrement selon la méthode de traitement des données utilisées : Le chiffrement par flot (Stream cipher), Le chiffrement par bloc (Block cipher).

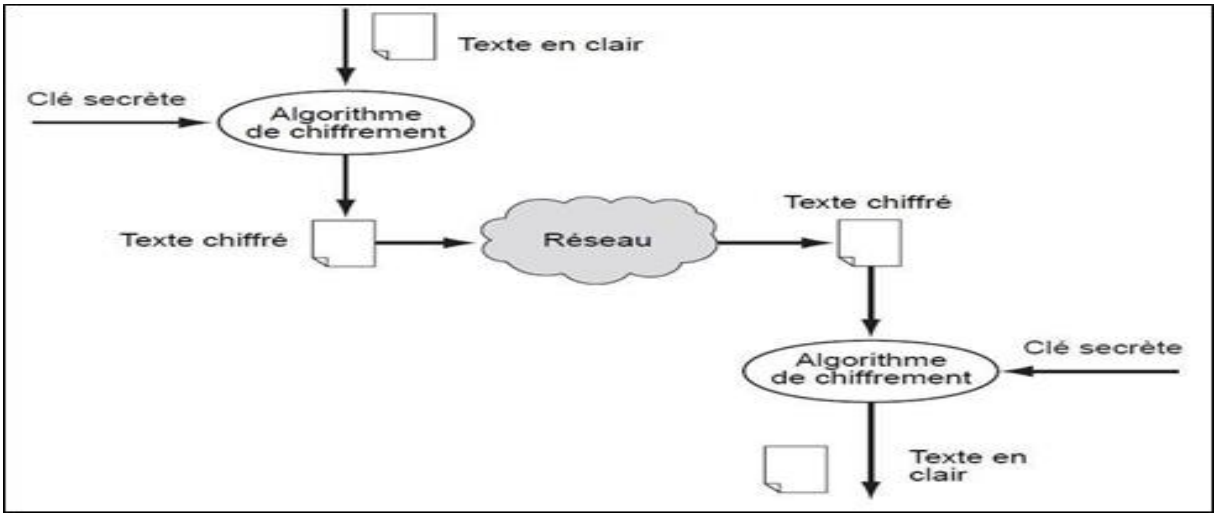



**Figure 1.3: Algorithme de chiffrement symétrique [1]**

### 1.2.3.2 Cryptographie asymétrique

Dans ce mode, la clé de chiffrement est différente de la clé de déchiffrement et elle ne peut pas être déduite ou calculée à partir de la clé de chiffrement et inversement. La clé de chiffrement appelée clé publique est destinée à être échangée ou diffusée, tandis que la clé de déchiffrement appelée clé privée est gardée secrète car elle n'est utilisée que par le récepteur et c'est le récepteur qui génère la clé publique et la transmet à l'émetteur. Dans ce mode seul le récepteur dispose de la clé de déchiffrement et donc elle ne peut être interceptée. Généralement nous trouvons les algorithmes asymétriques suivants : RSA, Diffie-Hellman et les courbe elliptiques … etc.

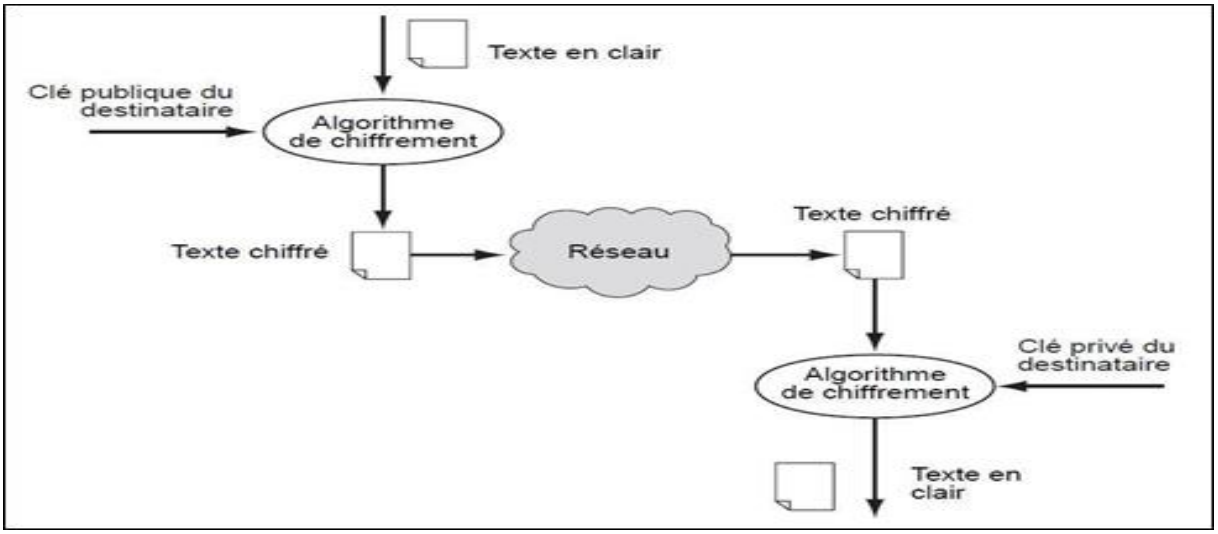



**Figure 1.4: Algorithme de chiffrement asymétrique [1]**

### 1.2.4 Les attaques physiques dans un environnement embarqué

Afin d'évaluer la résistance d'un algorithme cryptographique, il est nécessaire de préciser un modèle d'attaque pour délimiter le périmètre d'action de l'attaquant. En théorie, il est supposé ne pouvoir obtenir qu'un certain nombre de couples (entrée, sortie) de l'algorithme. En pratique, les algorithmes cryptographiques sont souvent implémentés sur des modules physiques tels que les cartes à puce dont l'attaquant peut observer (et même perturber) les interactions avec le monde extérieur. Ce modèle d'attaque est beaucoup plus fort que le modèle théorique car ces interactions sont fonction des opérations effectuées dans le module.

Nous pouvons trouver des attaques actives qui tenter de modifier le comportement de l'élément (par exemple en induisant des erreurs) ou passives c'est d'observer le comportement de l'élément sans chercher à la modifier. Il y a aussi les attaques invasives qui ouvrir le système (puce, smart card, etc.) pour accéder aux composants (cellules mémoires, bus, etc.) ou Non-invasives c'est de recueillir l'information depuis l'extérieur sans modifier l'élément.

De nombreux types d'attaques matérielle appartiennent à cette branche de la cryptanalyse :

- Attaques par canaux auxiliaires (Side Channel Attacks) : qui étudier l'aspect physique d'un cryptosystème sous différents angles au lieu d'étudier son aspect logique.
- Attaque par induction de faute (Fault Induction Attack) : qui consiste à modifier l'environnement du cryptosystème afin de provoquer d'éventuelles erreurs (par exemple faire varier la température, le voltage, la fréquence, induire des champs magnétiques intenses **[2]**).
- Attaque par induction de fautes optiques (Optical Fault Induction Attack) : qui modifier le contenu des cellules mémoire grâce à un faisceau lumineux **[3]**.
- Attaque par analyse de la puissance (Power Analysis Attack) : qui analyse la consommation électrique des composants durant une phase de calcul.
- Attaque par le temps (Timing Attack) qu'évaluer les temps de calcul nécessaires (cf. RSA).
- Attaque par analyse électromagnétique (Electromagnetic Analysis Attack) : qu'étude du rayonnement électromagnétique d'un système afin de connaître les données traitées et les calculs réalisés.
- Attaque par analyse acoustique (Acoustic Analysis Attack) : qu'écoute et analyse les sons produits par le processeur (ou cryptoprocesseur) durant ses calculs **[4]**, ou le bruit des touches durant la frappe d'un mot de passe **[5]**.

Afin de renforcer ces systèmes embarqué contres ces différent types d'attaque, des nouveaux travaux d'intégration et optimisation matérielle des coprocesseurs cryptographiques basées sur des théories complexe au niveaux ressources telque les courbes elliptiques sont proposer.

## 1.3 Les courbes elliptique EC

La cryptographie sur les courbes elliptiques est proposée indépendamment par Koblitz **[6]** et Miller **[7]** dans les années 80. Elle comprend un ensemble de techniques qui nous permettent de sécuriser des données en consommant moins de ressources. Elle attire récemment de plus en plus d'attention des chercheurs du monde entier, notamment pour le domaine de systèmes embarqués dans lequel les dispositifs électroniques ne possèdent qu'une puissance de calcul très limitée. L'avantage le plus important d'ECC par rapport aux autres algorithmes de cryptographie asymétrique, par exemple RSA **[10]**, est que l'on peut avoir un bon niveau de sécurité en utilisant une clé beaucoup plus courte. Les tests de performance de Gura et al. **[11]** ont montré que pour avoir le même niveau de sécurité qu'une clé RSA de 1024 bits, avec ECC, il suffit d'utiliser une clé de 160 bits. D'ailleurs, l'utilisation d'une clé plus courte implique aussi une consommation de mémoire et d'énergie moins importante et un calcul plus rapide.

### 1.3.2 Représentation de la courbe elliptique

Pour comprendre les notions mathématiques nécessaires voir l'annexe mathématique, dans cette section, nous présentons la définition des courbes elliptiques avec l'ensemble d'opérations que nous pouvons effectuer sur elles.

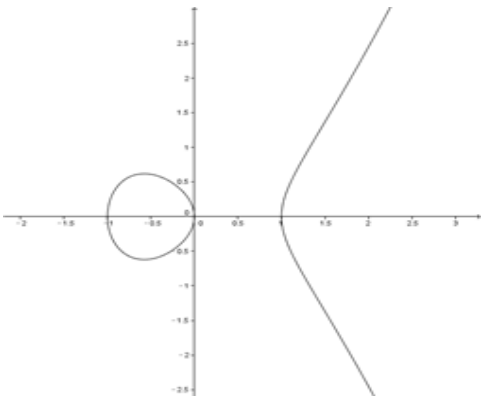
#### 1.3.2.1 Définition de EC

La courbe elliptique E est une courbe algébrique qui peut être représentée par l'équation Weierstrass (formule 1.1).

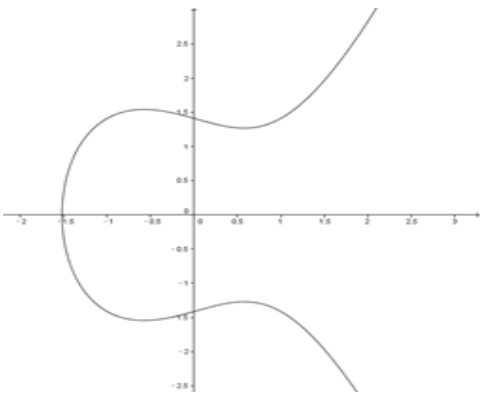
$$E: y^2 + a1xy + a3y = x^3 + a2x^2 + a4x + a6 \qquad (1.1)$$

Nous supposons que la courbe est définie dans un corps *K* et les paramètres *a1, a2, a3, a4, a6* $\in K$.

La forme de la courbe peut varier en fonction des paramètres choisis, dans la figure 1.5 nous avons deux exemples de courbe elliptique.



$Y^2 = X^3 - X$



$Y^2 = X^3 - X + 2$

**Figure 1.5: Exemples de courbe elliptique [16]**

Dans le domaine cryptographique, nous utilisons les courbes elliptiques qui sont définies dans un corps fini dont l'ordre $q = p^n$. Ce corps peut être soit premier, soit binaire et le choix de corps n'a pas une influence importante sur la performance du cryptosystèmes. Dans la littérature, il existe différents algorithmes et techniques qui nous permettent d'optimiser les performances de calcul sur les courbes qui sont définies sur les 2 types de corps **[13]**. Cependant pour une raison de simplicité d'implémentation et de présentation, durant les travaux de recherche de cette thèse, nous n'avons utilisé que des courbes qui sont définies sur un corps premier.

L'équation Weierstrass d'une courbe elliptique peut être simplifiée, si la courbe est définie sur un corps premier $F_p$ dont la caractéristique est différente de 2 et de 3. Nous pouvons transformer la formule 1.1 à l'équation de Weierstrass simplifiée (formule 1.2).

$$E/F_{2^m} : y^2 + xy = x^3 + ax^2 + b \qquad (1.2)$$

C'est aussi la forme de courbe que nous utilisons dans la suite de cette thèse.

#### 1.3.2.2 Opérations arithmétiques de EC

Pour obtenir le point symétrique de *P*, il suffit de changer le signe de sa coordonnée *y*, si $P = (x, y)$, alors $-P = (x, x+ y)$ *et* $P + (-P) = \infty$.

Le calcul de l'addition et le doublement de point est montré dans les formules 1.3 et 1.4. Supposons que $P = (x1, y1) \in E$, $Q = (x2, y2) \in E$ et $P, \pm Q$, alors $P + Q = (x3, y3)$ où

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \quad \text{et} \quad y_3 = \lambda(x_1 + x_2) + x_3 + y_1$$

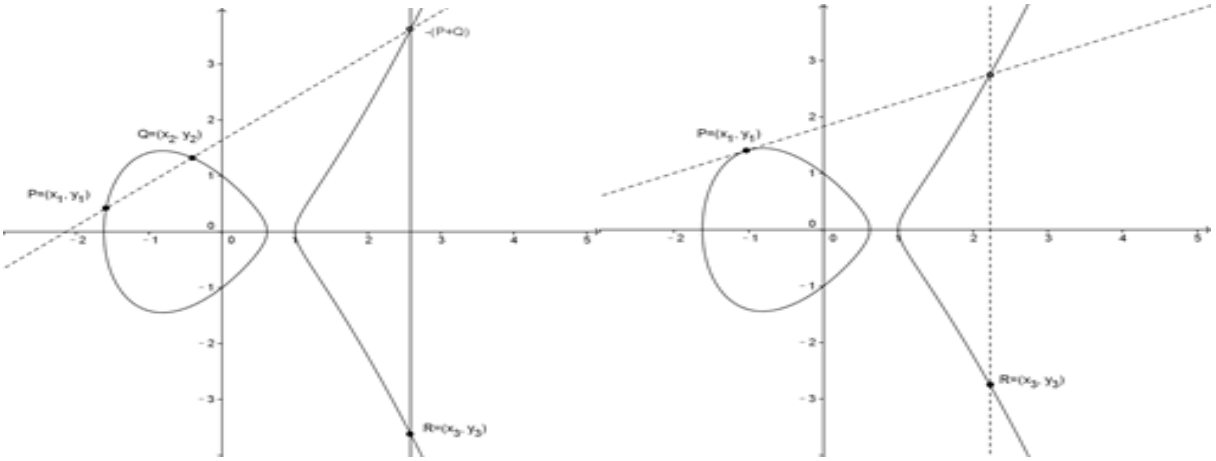
$$\lambda = (y_1 + y_2)/(x_1 + x_2) \qquad (1.3)$$

Si $P = (x1, y1) \in E$ et $P \neq -P$, alors $2P = (x3, y3)$ où

$$x_3 = \lambda^2 + \lambda + z = x_1^2 + b/x_1^2 \text{ et} \quad y_3 = x_1^2 + \lambda x_3 + x_3$$

$$\lambda = x_1 + y_1/x_1 \qquad (1.4)$$

Une représentation géométrique est donnée dans la figure 1.6. Pour additionner les points *P* et *Q*, nous traçons une droite qui passe par ces deux points, le résultat de l'addition est le point symétrique par rapport à l'axe abscisse du 3eme point d'intersection avec la courbe. Pour doubler le point *P*, il suffit de trouver la tangente à la courbe au point *P* et le résultat du doublement est le point symétrique par rapport à l'axe abscisse du deux points d'intersection avec la courbe.




Addition : $P + Q = R$ Doublement : $P + P = R$

**Figure 1.6: Addition de points sur les courbes elliptiques [16]**

### 1.3.3 Exemple d'application de ECC

Les courbes elliptiques sont applicables pour le cryptage, les signatures numériques, les générateurs pseudo-aléatoires et d'autres tâches.

**- Algorithme de signature numérique a courbe elliptique :** L'algorithme ECDSA prend en charge la signature de données avec des méthodes de courbe elliptique. Dans ce cas, nous effectuerons les opérations principales de signature et de vérification **[17]**.

**- Schéma de chiffrement intégré de courbe elliptique :** Le protocole ECIES est en effet une variante d'El Gamal standardisée. Avec ECIES, nous utilisons la clé publique de Cryptographie à courbe elliptique afin de dériver une clé symétrique. Dans ce cas, nous créons une clé symétrique à partir de la cryptographie à courbe elliptique, puis nous l'utilisons pour chiffrer avec AES **[9]**.

**- Courbe elliptique avec Diffie Hellman (ECDH) :** est utilisé pour créer une clé partagée. Bob générera une clé publique (*QB)* et une clé privée (*dB)*, tout comme Alice *(QA* et *dA).* Ils échangent ensuite leurs clés publiques et la clé partagée sera alors $dA \times dB \times G$ et où $G$ est le point générateur sur la courbe elliptique.

**- Google Tink :** est une bibliothèque cryptographique multi langue et multiplateforme. Avec OpenSSL, nous avons des liaisons complexes et qui étaient souvent axées sur des systèmes spécifiques, comme pour les DLL dans les systèmes Windows. Tink est open source et se concentre sur la création d'API simples et qui devraient rendre l'infrastructure plus portable. Nous utilisons une clé privée pour signer et une clé publique pour prouver la signature.

### 1.3.4 Comparaison de performance entre ECC et RSA

RSA fait partie des premiers cryptosystème asymétriques qui sont largement appliqués et aujourd'hui, il est toujours considéré comme le cryptosystème asymétrique le plus utilisé, notamment pour échanger des données confidentielles sur internet. Il est proposé en 1977 par Ron Rivest, Adi Shamir et Leonard Adleman **[10]**. L'algorithme est basé sur l'hypothèse qu'il est extrêmement difficile d'effectuer la factorisation d'un nombre entier en facteurs premiers. Certes, la factorisation n'est pas la seule méthode pour casser RSA, mais pour l'instant, il n'existe pas d'autre attaque qui soit suffisamment efficace.

RSA est proposé avec trois algorithmes qui sont conçus respectivement pour la génération de clés, la signature numérique et la vérification de signature.

Afin d'évaluer et de comparer la performance de RSA et ECC, deux implémentations utilisant respectivement ces deux cryptosystèmes sont testés dans **[14]**. ECC peut avoir le même niveau de sécurité que RSA avec une clé beaucoup plus courte. Les longueurs de clé utilisées sont suggérées dans **[15]** (voir le tableau 1.2) et les longueurs qui se situent dans la même colonne sont censées pouvoir fournir le même niveau de robustesse.

Pour être honnête nous avons comparé seulement la taille des clés, il y'a d'autre facture de comparaison.

**Tableau 1.2: Longueur de clé en bit**

| **Algorithme** | **Longueur de clé (bit)** | | | | |
|---|---|---|---|---|---|
| **ECC** | 163 | 233 | 283 | 409 | 571 |
| **RSA** | 1024 | 2240 | 3072 | 7680 | 15360 |

### 1.3.5 Cryptanalyse dans ECC

La meilleure attaque polyvalente connue sur l'ECDLP c'est la combinaison de l'algorithme de Pohling-Hellman et de l'algorithme rho de Pollard qui a une complexité d'exécution de $O(\sqrt{p})$ où $p$ est le plus grand diviseur premier de l'ordre $n$.

Pour résister à cette attaque, $n$ doit être choisi de manière à être divisible par un nombre premier p suffisamment grand pour que le racine de p soit une quantité de calcul infaisable :

$$p \geq 2 \text{ puissante } 160$$

Courbes elliptiques recommandées par le NIST sur des champs binaires : GF (2163), GF (2233), GF (2283), GF (2409) et GF (2571). Pour $k \times P$, les longueurs binaires de k sont : 163, 233, 283, 409, 571. Clé de 163 bits dans ECC équivalent Clé de 1024 bits dans RSA. **[16]**.

## 1.4. Nouvelle génération de développement sur circuit FPGA

### 1.4.1 Architecture de FPGA

Les périphériques FPGA sont des systèmes matériels reconfigurables. Ils permettent un prototypage rapide, c'est-à-dire d'explorer un certain nombre de solutions matérielles et de sélectionner la meilleure dans un délai plus court **[14]**. La méthodologie de conception sur FPGA repose sur l'utilisation d'un langage à haute description (VHDL ou System C) et un outil de synthèse. Pour cette raison, FPGA est devenu des plateformes populaires pour la mise en œuvre de générateurs aléatoires ou de schémas cryptographiques complets, en raison de la possibilité de générer une génération rapide et de haute qualité aléatoire. L'architecture générale d'un FPGA présentée dans la figure 1.6 est basée sur LCA, qui se compose de trois parties, à savoir : le bloc logique configurable (CLB) **[8]**, le bloc d'entrée-sortie (IOB) et les commutateurs d'interconnexion. FPGA pourrait également inclure des composants plus complexes comme un traitement du signal numérique (DSP), une mémoire vive (RAM), un gestionnaire d'horloge numérique (DCM) ou un convertisseur analogique-numérique (ADC / DAC). La nomination des blocs internes dépend des fournisseurs de FPGA (Xilinx, Altera, Actel ...) même s'ils ont une fonctionnalité similaire. La structure CLB est principalement basée sur des tables de consultation (LUTs **[18]**), avec en plus une bascule et quelques multiplexeurs. *K*-input LUT est un tableau de mémoire de $2K \times 1$ bit basé sur une table de vérité d'entrées de *K* bits. Ces derniers peuvent exécuter toutes les fonctions logiques comme XOR / ADD / SHIFT. . .etc.

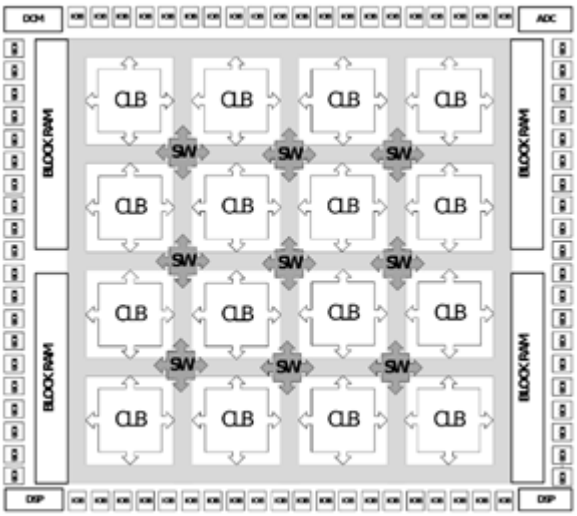



**Figure 1.6: Carte FPGA XILINX Kintex-7**

### 1.4.2 Nouvelle génération de conception logicielle sur circuit FPGA

L'une des innovations dans le domaine de conception matérielle sur FPGA, c'est l'intégration de programmation purement logicielle dans ce domaine pour accélérer.

L'outil Vitis HLS de Xilinx a été développé pour simplifier l'utilisation des fonctions C / C ++ pour l'implémentation en tant que noyaux matériels dans le flux de développement d'accélération d'application Vitis et d'utiliser le code C / C ++ pour développer RTL IP pour les conceptions FPGA.

Dans le flux d'accélération d'application Vitis, l'outil Vitis HLS automatise une grande partie des modifications de code nécessaires pour implémenter et optimiser le code C / C ++ en logique programmable et obtenir une faible latence et un débit élevé. L'inférence des pragmas requis pour produire la bonne interface pour vos arguments de fonction et pour pipeline des boucles et des fonctions dans votre code est la base de Vitis HLS dans le flux d'accélération d'application.

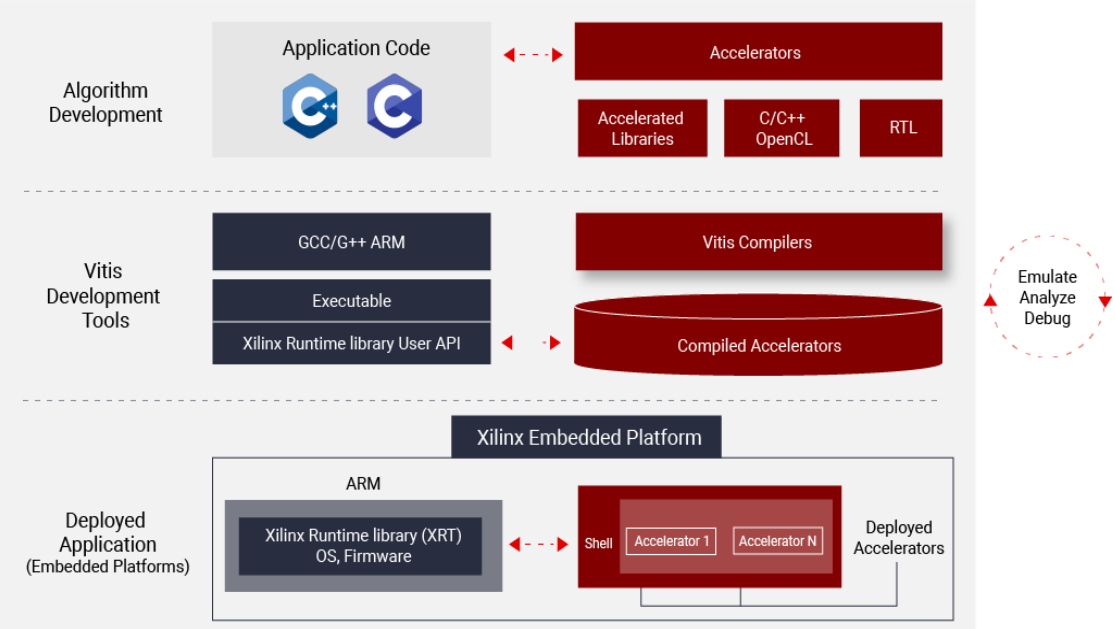



**Figure 1.7: Conception logicielle sur circuit FPGA**

### 1.4.3 Les avantages d'utilisation de HLS

Nous citons quelques avantages de son utilisation :

- Développer et valider des algorithmes au niveau C pour concevoir à un niveau qui est abstrait des détails d'implémentation matérielle.
- Utilisation de la simulation C pour valider la conception et itérer plus rapidement qu'avec la conception RTL traditionnelle.
- Contrôler le processus de C-synthèse à l'aide de pragmas d'optimisation pour créer des implémentations hautes performances.
- Création de plusieurs solutions de conception à partir du code source C et des pragmas pour explorer l'espace de conception et trouver une solution optimale.
- Recompilez rapidement la source C pour cibler différentes plateformes et périphériques matériels.

### 1.4.4 Les critère de performance de HLS sur FPGA

#### 1.4.4.1 planifications et de liaison

La figure suivante montre un exemple des phases de planification et de liaison pour cet exemple de code :

```
int foo (char x, char a, char b, char c) {
Char y ;
y = x*a+b+c;
Return y ;}
```

Dans la phase de planification de cet exemple, la synthèse de haut niveau planifie les opérations suivantes pour qu'elles se produisent pendant chaque cycle d'horloge et comme montre la Figure1.8:

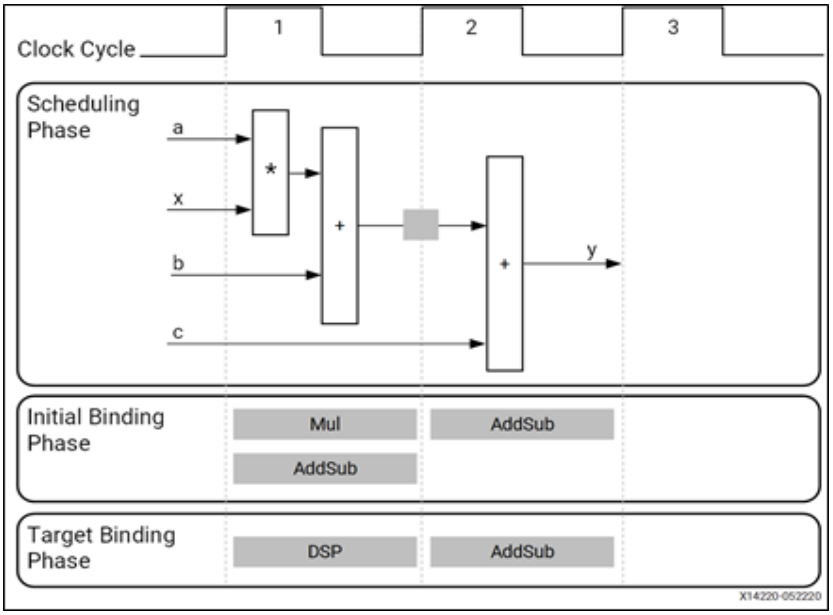



**Figure 1.8: Exemple de planification et de liaison**

- Premier cycle d'horloge : multiplication et premier ajout
- Deuxième cycle d'horloge : deuxième ajout, si le résultat de la première addition est disponible dans le deuxième cycle d'horloge et génération de sortie.

Comme nous distinguons dans le code C originale de l'algorithme, nous pouvons distinguer deux propriétés qui peuvent provoquer la différence dans le matérielle FPGA.

***a) Récursivité :***

Dans les FPGA, les feedback Loop sont indésirable et peuvent pas des fois synthétisée (malgré qu'ils puissent être simulés), comme For, While Loop, ou un chemin retour qui dépend de l'ancien. Ça peut résoudre par ce bloc de feedback comme une machine d'état FSM.

***b ) DSP :***

Les blocs DSP48 support plusieur types de fonction comme : multiplicateur, accumulateur multiplicateur (MAC), multiplicateur suivi d'un additionneur, additionneur à trois entrées, décaleur en barillet, multiplexeurs à bus large, comparateur d'amplitude ou compteur large. Tout ça sans utilise les ressources de FPGA. Généralement pour la multiplication nous pouvons trouver des multiplier de 18x18-bits ou 25x18-bit. Donc pour faire la multiplication *1812433253*X* il nous coûtera plus de 3 multiplicateurs DSP.

***c ) Les Mémoires :***

FPGA contient plusieurs types de mémoires qui peuvent être utilisé dans le design. Une mémoire avec un seul entré (Single RAM), une mémoire avec deux entré (Dual Port RAM), ou des ROM. Ce qui nous intéresse et le nombre de cycle qui peut prendre une donnée être écrite dedans et lire depuis. Si le design est synchronisé par une seule fréquence le minimum cycle d'horloge pour lire et écrire après seras en deux tops d'horloge. C'est nous appliquons ça avec la fréquence d'entré nous disons que le design est ½ moins. C.à.d., nous ne pouvons pas lire et écrire en même temps pour la même adresse si la mémoire est cloquée par la même horloge. Suite à cela, l'élément qui vas définie l'implémentation de cette partie dans le FPGA sont les mémoires. Comment lire et écrire depuis et combien de cycle d'horloge nécessaire pour compléter une génération d'une sortie finale.

### 1.4.4.2 Latence et fréquence de fonctionnement

Chaque architecture implémentée dans le circuit FPGA doit être synchronisé par l'horloge, à chaque top d'horloge (positive Edge) ont Read ou Write dans un bloc. Donc il faut bien comprendre cette notion très bien, car ont FPGA (Hardware) les résultats peut diffèrent selon le programme et comment il est synchronisé. Le cycle est le nombre de top d'horloge nécessaires pour avoir une sortie.

Nous avons deux types de lecteur de la fréquence ou période dans un design :

- Fréquence d'entré qui est réactionnaire avec l'oscillateur extérieur de FPGA est limitée en période. Nous pouvons aussi le divisé ou multiplié pour avoir le période d'entré souhaité pour le fonctionnement de design.
- Fréquence de design : cette fréquence est reliée au temps parcouru par le DATA d'un registre a un autre. Dans le FPGA est nommé « Clock to Setup » c.à.d que le donné propagé de registre doit arriver avant que le période se termine et une autre période commence (Setup time).

La figure suivante montre l'exécution cycle par cycle complète du code de l'exemple précédent, y compris les états de chaque cycle d'horloge, des opérations de lecture, des opérations de calcul et des opérations d'écriture.

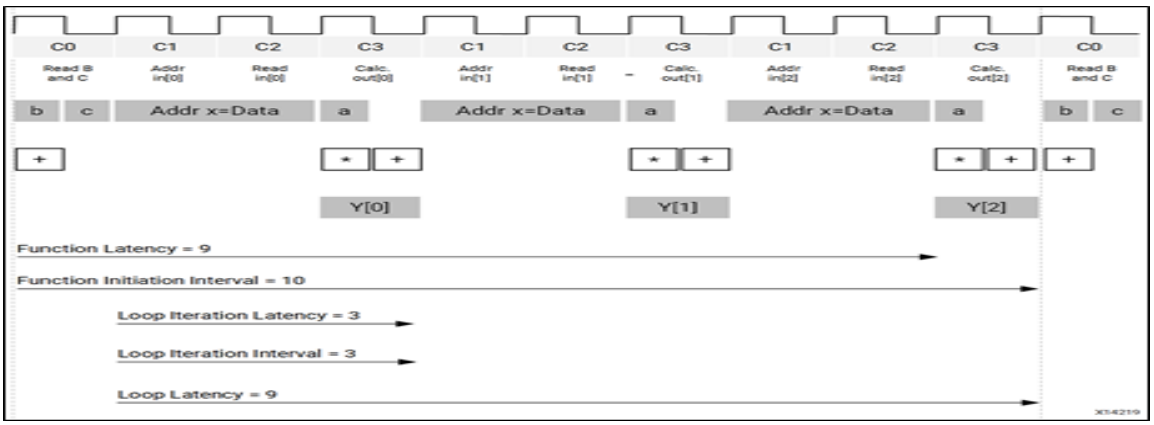



**Figure 1.9: Exemple de latence et d'intervalle d'initiation**

### 1.4.5 Résultats d'implémentation ECC sur FPGA

Dans cette partie nous allons voir quelques résultats d'implémentation de ECC sur FPGA.

**Tableau 1.3 : Table de résultats d'implémentation ECC sur FPGA [30]**

| | **FPGA** | **Fréquence (Mhz)** | **LUT** | **FF** | **Courbe** |
|---|---|---|---|---|---|
| **Sinha Roy** | Virtex 5 | 0,105 | 10195 | - | 163 |
| **Ansari** | Virtex 2 | 0,0243 | 8300 | 1100 | 163 |
| **Liu** | Virtex 4 | 0,111 | - | - | 163 |
| **Luts** | Virtex 2 | 0,002 | 7362 | 1930 | 163 |
| **Loi** | Virtex 4 | 0,001 | 3815 | 1219 | 163 |
| **Sutter** | Virtex 5 | 0,05 | 22340 | - | 163 |
| **Chelton** | Virtex 4 | 0,05 | - | - | 163 |

## 1.5 Conclusion

En conclusion, la sécurité informatique est l'ensemble des moyens mis en œuvre pour réduire la vulnérabilité d'un système contre les menaces accidentelles ou intentionnelles, parmi ces moyens la cryptographie qui est devenu aujourd'hui un élément important dans les entreprises.

Après quelque test de fréquences des anciennes solution algorithmiques, nous avons trouvé qu'ils ne sont pas assez pas rapides pour optimiser le déroulement de ECC au niveau matériel, c'est pour cela nous somme intéressé par l'accélération de performance de ECC à base FPGA afin d'éviter les attaques matérielles.

# Chapitre 2 : Diffie-Hellman basé sur la cryptographie des courbes elliptiques avec Radix-$2^r$

## 2.1 Introduction

Dans ce chapitre nous avons fait un rappel sur les courbes elliptiques, ensuite nous avons expliqué l'opération la plus importante sur les courbes, la multiplication scalaire et après une présentation de l'ensemble d'algorithmes et de techniques proposés dans la littérature qui visent à améliorer la performance de calculs sur les courbes et la nouvelle solution Radix-$2^r$. Enfin nous présentons l'architecture et l'implémentation matérielle sur FPGA des courbes elliptiques.

## 2.2 Rappel sur les Courbes Elliptiques

La courbe elliptique *E* est une courbe algébrique qui peut être représentée par l'équation Weierstrass (formule 2.1).

$$E : y2 + a_1\, x.y + a_3 y = x^3 + a_2\, x^2 + a_4\, .x + a_6 \qquad (2.1)$$

Nous supposons que la courbe est définie dans un corps *K* et les paramètres *a1, a2, a3, a4, a6* $\in$ *K*. Après la présentation des notions mathématiques nécessaires, nous allons passer, dans cette section, à la définition des courbes elliptiques avec l'ensemble d'opérations que nous pouvons effectuer sur elles. Dans ce qui suit, nous allons détaillés les principaux piliers qui constitues les courbes elliptiques, la figure ci-dessous représente les niveaux d'abstraction de ECC, nous commençons par la multiplication scalaire jusqu'à les opérations arithmétiques.

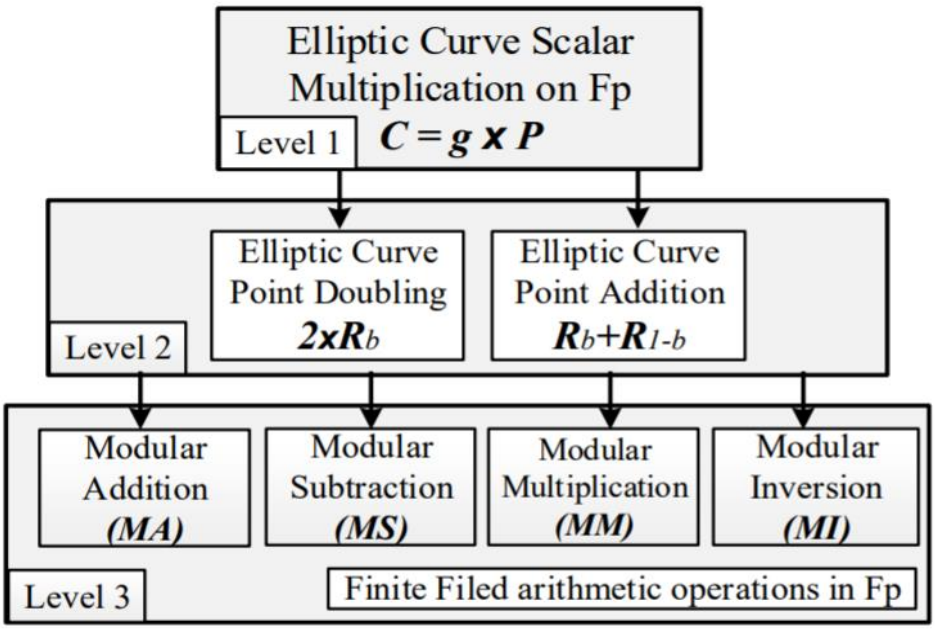



**Figure 2.1: Niveaux d'abstraction d'implémentation de ECC [31]**

### 2.2.1 Exemple d'application ECC avec protocole Diffie-Hellman

Comme nous avons vu dans la partie précédente, le chiffrement symétrique permet d'échanger des données chiffrées sur un canal non sécurisé. Cependant, ce système nécessite d'échanger la clé de chiffrement entre l'expéditeur et le destinataire d'un texte chiffré. Cela pose un problème lors d'un échange à distance par exemple : comment échanger la clé de manière confidentielle si l'on ne dispose pas de canal de communication sécurisé ?

En 1976, les cryptologues Diffie et Hellman ont proposé une solution révolutionnaire à ce problème : le protocole d'échange de clés Diffie-Hellman. Ce système a posé les bases de la cryptographie asymétrique et a permis de généraliser l'usage de la cryptographie dans les communications informatiques. L'échange de clés Diffie-Hellman consiste à échanger une clé secrète entre Alice et Bob sur un réseau non sécurisé de manière confidentielle. Il se déroule comme ceci :

1. P est un point dans la courbe elliptique avec *(x, y)* de grande taille choisir aléatoirement.
2. Alice choisit un entier aléatoire de grande taille a qui est sa clé privée.
3. Elle calcule $A = P*a$. a est la clé privée d'Alice et *A* est la clé publique d'Alice.
4. Alice transmet à Bob la valeur de *A*, sa clé publique.
5. De même, Bob choisit un entier aléatoire de grande taille *b*. Il calcule $B = P*b$. *b* est la clé privée de Bob et *B* est la clé publique de Bob.
6. Bob transmet à Alice *B*, sa clé publique.
7. À ce moment, Alice calcule $K=B*a = P*(b*a)$ et Bob calcule $K=A*b = P*(a*b)$. Ces deux variables ont la même valeur. Cette variable *K* est la clé secrète partagée par Alice et Bob.

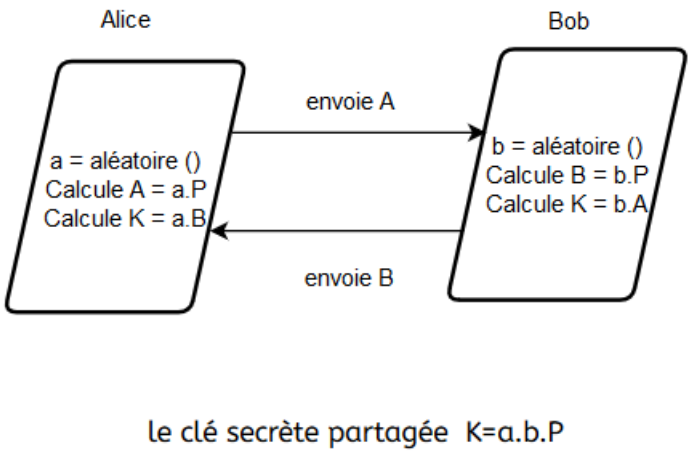



**Figure 2.2: Échange de clés Diffie-Hellman (ECDH)**

Imaginons maintenant qu'une tierce personne, qu'on appellera Adam, ait écouté toutes les communications entre Alice et Bob. Adam connaît *P, A = P*a* et *B = P*b*. Cependant, Adam ne peut pas calculer a car la fonction *log(A) = log(P*a) = a* est impossible à calculer. De même, elle ne peut pas calculer *b.* Adam ne peut donc pas calculer *B*a* ou *A*b.* Cette variable est donc confidentielle entre Alice et Bob, qui peuvent l'utiliser comme clé secrète partagée pour chiffrer leurs échanges de manière confidentielle avec le chiffrement symétrique. L'opération le plus complexe dans le protocole Diffie-Hellman et ECC c'est la multiplication scalaire.

### 2.2.2 Différentes formes de système de coordonnées

Un système de coordonnées est une organisation qui utilise des nombres, ou des coordonnées, pour déterminer la position d'un point ou d'un autre sur l'espace euclidien. L'utilisation d'un système de coordonnées permet de traduire des problèmes de géométrie en problèmes de nombres. Une courbe elliptique peut être représentée par plusieurs systèmes de coordonnées **[11]**. Les additions de points (ADD) et les doublages de points (DBL) peuvent être mené en utilisant le système de coordonnées suivant **[5][6].**

- Système de coordonnées affines.
- Projection standard.
- Projection jacobienne.
- Lopez-Dahab.

**Tableau 2.1: Coût du système de coordonnées le plus utilisé dans GF (2)**

| Système de coordonnées | ADD | DBL |
|---|---|---|
| **Projectif standard** | 14M+2S | 7M+3S |
| **Lopez-Dahab** | 13M+4S | 3M+5S |
| **Coordonnée affine** | 10M+1S | 10M+2S |

### 2.2.3 Multiplication scalaire

Sur la base de l'addition de points, nous trouvons la multiplication de points, notée $Q = kP$ sur une courbe elliptique $E$ où $k \in Z^+$ et $(P, Q) \in E$. Cette opération est la plus coûteuse et importante dans ECC. Une multiplication de point est une suite d'additions de point consécutives :

$$Q = KP = P + P + P \ldots + P$$

Nous pouvons voir dans les formules 1.3 et 1.4 (voir chapitre 1) que le calcul de l'addition de point est compliqué, il est donc inefficace de répéter successivement l'addition de point. La méthode pour accélérer la multiplication scalaire est d'utiliser l'algorithme doublement et addition. Dans le tableau ce dessus, nous avons une liste de nombres d'opérations nécessaires pour calculer une addition et un doublement sur les courbes elliptiques, Nous avons utilisé coordonnées affines.

### 2.2.4 Problème du logarithme discret sur ECC (ECDLP)

Nous avons déjà expliqué les opérations qui nous utilisons dans les courbes elliptiques, notamment la multiplication scalaire qui est utilisée pendant les calculs cryptographiques. Dans cette partie, nous expliquons que la multiplication scalaire est la clé pour assurer la sécurité d'un cryptosystème d'ECC. Le niveau de sécurité d'ECC dépend de la difficulté pour traiter le problème de logarithme discret sur les courbes elliptiques.

Nous choisissons, sur une courbe elliptique définie dans un corps premier fini $E$ ($F_P$), un point $P$ de l'ordre n comme le générateur du groupe cyclique $<P>$, un autre point $Q \in <P>$. Le problème du logarithme discret sur les courbes elliptiques est de trouver $L \in [0, n-1]$ satisfaisant $Q = L*P$. La solution pour résoudre ce problème est de calculer exhaustivement $1P, 2P, 3P \ldots$ jusqu'à ce que nous trouvions $Q$, mais le calcul peut devenir très long si la valeur de $L$ est suffisamment grande. Il est donc difficile de retrouver la valeur de $L$ à partir de $Q$ et $P$ (voir la formule 2.2).

$$L = log_p Q \tag{2.2}$$

La formule (2.2) n'est pas preuve mathématiquement qu'elle est non résoluble **[20]**.

### 2.2.5 Optimisation de multiplication scalaire

L'utilisation de petites clés est l'avantage de ECC car elle augmente la vitesse des calculs. Elle est un choix approprié pour les matériels ayant une mémoire et une puissance de calcul limitées. Les performances de la multiplication scalaire ont une forte influence sur les performances de l'ensemble du cryptosystèmes.

Nous allons présenter ci-dessous Certains algorithmes d'optimisation et ses problèmes :

### 2.2.5.1 Méthode binaire

**Algorithme 2.1: méthode binaire de gauche à droite pour le scalaire de multiplication**

Entrée : $k = (k_{t-1}, \dots, k_1, k_0)_2, P \epsilon E(F_q)$;

Sortie : Q=k*P ;

1 $Q \leftarrow \infty$;

2 Pour i=l-1 à 0 faire

2.1 $Q \leftarrow 2 * Q$;

2.2 $Si\ k_i$=1 alors $Q \leftarrow Q + P$;

3 Retourner Q ;

Dans cette algorithme le nombre moyen d'opérations de doublement *DBL = l-1* et l'addition *ADD = l / 2 -1* ( *l* est la taille de clé ). Le problème de cet algorithme c'est qu'est vulnérable par l'attaque d'analyse de puissance simple. (Voir la figure 2.3)

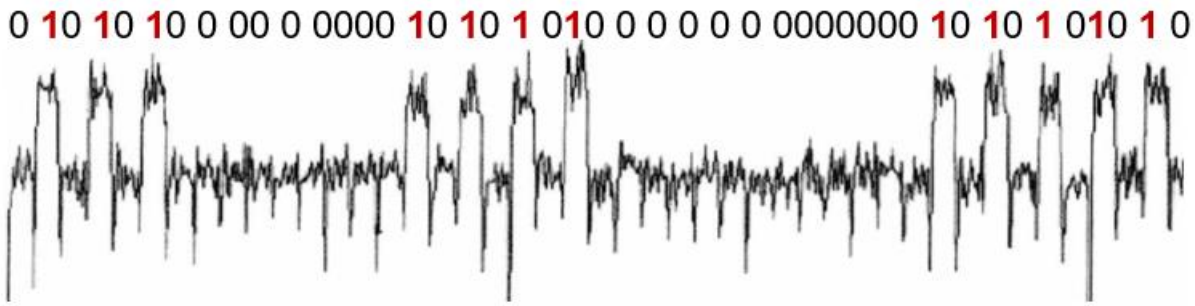



**Figure 2.3: attaque d'analyse de puissance simple**

### 2.2.5.2 Échelle de Montgomery

L'idée principale de l'algorithme consiste à exécuter une double exponentiation, tout en maintenant une différence constante entre les deux points concernés **[21]**. Plus précisément, à chaque étape, l'algorithme calcule de manière entremêlée deux points *P1* et *P2*, dont la différence est constamment égale à *P*. Dans cette algorithme le nombre moyen d'opérations de doublement *DBL = l-1* et l'addition *ADD = l -1* ( *l* est la taille de clé ). Le nombre moyen de combinaisons pour pirater le clé *k* est $2^{(l-1)}$.

**Algorithme 2.2: échelle de Montgomery de gauche à droite**

Entrée : $k = (1, k_{n-2}, \dots, k_1, k_0)_2, P = (x, y)$;

Sortie : Q=k*P ;

1 $R_0 \leftarrow P; R_1 \leftarrow 2P;$

2 Pour i=n-2 à 0 faire

2.1 $Si\ k_i = 1\ alors\ R_0 \leftarrow R_0 + R_1;\ R_1 \leftarrow 2R_1;$

2.2 Sinon $R_1 \leftarrow R_0 + R_1;\ R_0 \leftarrow 2R_0;$

3 Retourner Q=$R_0$ ;

### 2.2.5.3 Forme non adjacente (NAF)

Si *P= (x, y)*$\epsilon E(F_q)$ alors *-P= (x, x+ y)* si $F_q$ est champ binaire et *-P= (x, -y)* si $F_q$a la caractéristique $> 3$.

L'idée consistant à recoder des éléments d'un calcul pour en améliorer l'efficacité n'est évidemment pas nouvelle, déjà en 1951 Booth avait utilisé cette approche dans le cadre de la multiplication de deux entiers **[22]**. De nombreux recodage ont ainsi été proposés depuis. Citons par exemple **[23]** ou encore **[24]**. Parmi toutes ces représentations la forme non adjacente est particulièrement appropriée. Dans l'algorithme de NAF le nombre moyen d'opérations de doublement *DBL = l-1* et l'addition *ADD = [(l +1) /3] -8/9* ( *l* est la taille de clé ). Le nombre moyen de combinaisons pour pirater le clé k est $2^{\lceil (l+1)/3 \rceil} + 1/9$. Dans la partie suivante nous allons voir l'algorithmique proposée qui nous permettent d'accélérer le calcul de multiplication scalaire.

**Algorithme 2.3: calculer le NAF d'un entier positif**

Entrée :A Entier positive k;

Sortie : NAF(k) ;

1 $i \leftarrow 0;$

2 Tant que k>=1 faire

2.1 Si k est impair alors $k_i \leftarrow 2 - (k \bmod 4); k \leftarrow k - k_i;$

2.2 Sinon $k_i \leftarrow 0;$

2.3 $k \leftarrow \frac{k}{2}; i \leftarrow i + 1;$

Retourner :$(k_{i-1}, k_{i-2}, \dots, k_1, k_0);$

**Algorithme 2.4: méthode binaire NAF pour le scalaire de multiplication**

Entrée : $A\ Entier\ positive\ k, P\epsilon E(F_q)$;

Sortie : $Q=k*P$ ;

1 Utiliser l'algorithme 2.3 pour calculer NAF(k)=$\sum_{i=0}^{l-1} k_i * 2^i$;

2 $Q \leftarrow \infty$;

3 Pour i=l-1 à 0 faire

3.1 $Q \leftarrow 2 * Q$;

3.2 $Si\ k_i$=1 alors $Q \leftarrow Q + P$;

3.3 $Si\ k_i$=-1 alors $Q \leftarrow Q - P$;

Retourner Q ;

## 2.3 Arithmétique de multiplication scalaire Radix-$2^r$

### 2.3.1 Système de nombre à base $2^r$

La représentation Radix-$2^r$ a été développée par Sam en 1990 **[25] [26]**. En Radix-$2^r$, un nombre *x* en complément à deux de longueur *n* bits est écrit comme suit :

$$x = \sum_{j=0}^{\left(\frac{n}{r}\right)-1}\left(x_{rj-1} + 2^0 x_{rj} + 2^1 x_{rj+1} + 2^2 x_{rj+2} + \cdots + 2^{r-2} x_{rj+r-2} + 2^{r-1} x_{rj+r-1}\right) * 2^{rj}$$

$$x = \sum_{j=0}^{\left(\frac{n}{r}\right)-1} Q_j * 2^{rj} avec Q_j = \{-2^{r-1}, -2^{r-1}+1, \ldots, -1, 0, 1, \ldots, 2^{r-1}-1, 2^{r-1}\} \quad (2.3)$$

Ou *x-1 = 0* et *r E* $N^*$. Pour des raisons de simplicité, nous supposons que *r* est un diviseur de *N*. Dans l'équation (1.33), la représentation en complément à deux de *x* est divisée en *n*/*r* tranches représentées en complément à deux, chacune de longueur *r + 1* bits. Chaque paire de deux tranches contiguës ont un bit en commun.

Enfin, une constante *C* peut être exprimée comme suit :

$$c = \sum_{j=0}^{\frac{N-1}{r-1}} (-1)^{c_{rj+r-1}} * m_j * 2^{rj+k} \quad (2.4)$$

### 2.3.2 Représentation canonique de Radix-$2^r$

Radix-$2^r$ est une représentation redondante puisque d'autres simplifications sont possibles :

$$\ldots + 2^{r(j+1)} - 2^{rj+(r-1)} \pm \cdots = \cdots + 2^{rj+(r-1)} \pm \quad (2.5)$$

$$\ldots - 2^{r(j+1)} + 2^{rj+(r-1)} \pm \cdots = \cdots - 2^{rj+(r-1)} \pm \quad (2.6)$$

**Tableau 1.2. Table de recodage de Radix-$2^4$**

| $Q_j$ | | | | | $m_j$ | $k_j$ |
|---|---|---|---|---|---|---|
| $x_{4j+3}$ | $x_{4j+2}$ | $x_{4j+1}$ | $x_{4j}$ | $x_{4j-1}$ | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 3 | 0 |
| 0 | 0 | 1 | 1 | 0 | 3 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 2 |
| 0 | 1 | 0 | 0 | 0 | 1 | 2 |
| 0 | 1 | 0 | 0 | 1 | 5 | 0 |
| 0 | 1 | 0 | 1 | 0 | 5 | 0 |
| 0 | 1 | 0 | 1 | 1 | 3 | 1 |
| 0 | 1 | 1 | 0 | 0 | 3 | 1 |
| 0 | 1 | 1 | 0 | 1 | 7 | 0 |
| 0 | 1 | 1 | 1 | 0 | 7 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 3 |
| 1 | 0 | 0 | 0 | 0 | 1 | 3 |
| 1 | 0 | 0 | 0 | 1 | 7 | 0 |
| 1 | 0 | 0 | 1 | 0 | 7 | 0 |
| 1 | 0 | 0 | 1 | 1 | 3 | 1 |
| 1 | 0 | 1 | 0 | 0 | 3 | 1 |
| 1 | 0 | 1 | 0 | 1 | 5 | 0 |
| 1 | 0 | 1 | 1 | 0 | 5 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 2 |
| 1 | 1 | 0 | 0 | 0 | 1 | 2 |
| 1 | 1 | 0 | 0 | 1 | 3 | 0 |
| 1 | 1 | 0 | 1 | 0 | 3 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |

Notez que pour Radix-$2^4$, *k E {0,1, 2,3} et m $_j$ E {0,1,3,5,7}* **[28]**.

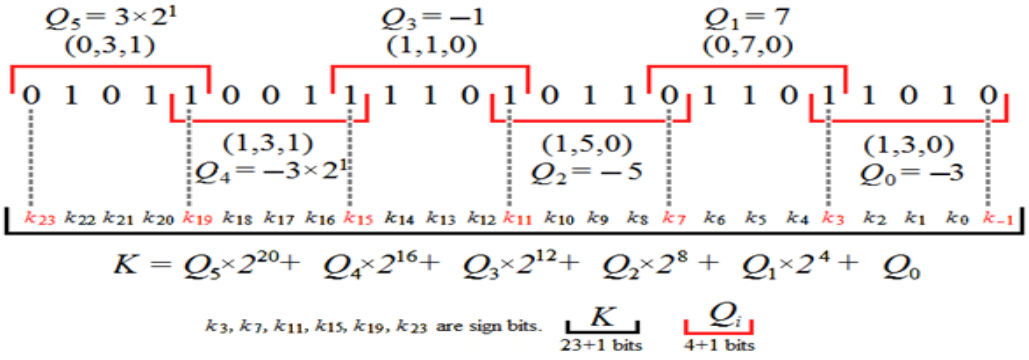



**Figure 2.7: Partitionnement de $(5892973)_{10}$ en Radix-$2^4$ [28].**

$k = 3 * 2^{21} - 3 * 2^{17} - 1 * 2^{12} - 5 * 2^8 + 7 * 2^4 - 3 * 2^0 = \left(0300\bar{3}0000\bar{1}000\bar{5}00070003\right)_{2^4}$

Nous avons diminué les Chiffres qui sont différents de zéro de 15 à 6.

### 2.3.3 Les caractéristiques de Radix-$2^r$

Nous présentons les caractéristiques essentielles de Radix-$2^r$ :

- Entièrement prévisible : Cette fonctionnalité permet non seulement aux concepteurs d'obtenir une image de pré-implémentation sur la surface, la vitesse et la puissance, mais elle permet également aux outils de synthèse de satisfaire rapidement les contraintes des utilisateurs et d'effectuer les compromis nécessaires sans les retours à la recherche de la solution appropriée.
- Sublinéaire : Pour les problèmes de grande complexité, Radix-$2^r$ est probablement le seul qui soit même réalisable.
- Solution haute vitesse et faible consommation : Adder-profondeur (Ath) est un mesure vitesse et aussi un bon indicateur de la consommation d'énergie. Les algorithmes MCM existants ne peuvent pas concurrencer Radix-$2^r$ car il permet une réduction logarithmique d'Ath.
- Une application de Radix-$2^r$ dans la communication sans fil où les filtres utilisés dans le circuit récepteur des systèmes mobiles doivent être d'ordre supérieur et doivent être réalisés pour consommer moins d'énergie et fonctionner à haute vitesse **[27]**.

### 2.3.4 Les mesures de Radix-$2^r$

Dans cette partie nous allons présenter les mesures de notre algorithme de multiplication scalaire **[28]** :

-Nombre maximum d'additions :

$$\lceil (l+1)/w \rceil + 2^{w-2} - 1w = 2.\frac{w\left(\sqrt{(l+1).log(2)}\right)}{log(2)}$$

-Nombre moyen d'additions :

$$(1-2^{-w}) * \lceil \frac{l+1}{w} \rceil + 2^{w-2} - 1$$

-Consommation de mémoire :

$$p_s(2^w) = \{3*p, 5*p, 7*p, \dots, \left(2^{(w-1)}-1\right)*p\}$$

$$|p_s(2^w)| = 2^{w-2} - 1$$

-Mémoire supplémentaire :

$$It = 2^w[(w-1) + \lceil log_2(w-1) \rceil]$$

-Nombre moyen de combinaisons pour pirater K :

$$a^{(1-2^{-w})*(l+n_z)/w}$$

$$a = \sum_{i=0}^{w-1} \left[\left(2^{2*(w-i)-1}\right)/(2^{w+1}-2)\right]$$

$$N_z = w - (1 mod w)$$

### 2.3.6 Exemple illustratif de Radix-$2^r$ pour ECC

Pour exprimer $x = (5892973)_{10}$ en Radix-$2^r$, une représentation en complément à deux de $x$ est nécessaire, ce qui est $(010110011110101101101101)_2$. Ainsi, dans la représentation en complément à deux, la taille en bit de la constante est égale à $n = 23$. Nous choisissons dans ce cas $r = 4$. Comme 23 n'est pas un multiple de 4, le bit de signe (0 dans ce cas) est dupliqué une fois de sorte que $n = 24$. Pour $C = 5892973$, les équations (1.33) et (1.34) deviennent respectivement :

$$x = \sum_{j=0}^{3} Q_j * 2^{4j} etx = \sum_{j=0}^{3} (-1)^{c_{4j+3}} * m_j * 2^{4j+k}$$

Pour déterminer les valeurs inconnues $c_{4j+3}$, $m_j$ et $k_j$, la table de correspondance de Radix-$2^4$ (tableau 1.2) est indexée par les termes $Q_j$. En se référant au tableau 1.2, les triplets $(c_{4j+3}, m_j, k_j)$ correspondant à $Q_0$, $Q_1$, $Q_2$, $Q_3$, $Q_4$ et $Q_5$ sont (0,3,1), (1,3,1), (1,1,0), (1,5,0), (0,7,0) et (1,3,0) respectivement. Par conséquent, nous pouvons écrire : $x = (3\times2^{21}) - (3\times2^{17}) - (1\times2^{12}) - (5\times2^{8}) + (7\times2^{4}) - (3\times2^{0}) = (5892973)_{10}$.

### 2.3.5 Algorithme proposée de Radix-$2^r$ pour ECC

**Algorithme 2.5: Méthode de Radix-$2^r$ proposée [28]**

Entrée : scalaire $k$ *avec taille de bit* $l, P \epsilon E(F)$;

Sortie : R=k*P ;

1 Calculer w

2 Déterminer $w_{min}$ parmi $(\lfloor w \rfloor, \lceil w \rceil)$

3 Calculer et sauvegarder l'ensemble Ps $(2^{w_{min}})$

4 Concaténer 0 a $k_0$

5 Concaténer Nz a $k_{l-1}$

6 R=0 ;

7 Pour i= (l+ Nz) /$w_{min}$ -1 à 0 faire

7.1 $Sik_{w_{min}*i+w_{min}-1} = 1 alors Q_i \leftarrow \overline{Q_i}$;

7.2 Utiliser $Q_i$ pour charger $(m_i, n_i)$ à partir de la table d'indexation

7.3 Pour j=$w_{min}$ -1 à 0 faire

7.3.1 R=R+R ; // DBL

7.3.2 Si $m_i \neq 0$ alors

7.3.2.1 Si $j = n_i$ alors

7.3.2.1.1 $Sik_{w_{min}*i+w_{min}-1} = 0 alors R \leftarrow R + (m_i * p)$; // ADD

7.3.2.1.2 Sinon $R \leftarrow R - (m_i * p)$;

Retourner R ;

## 2.4 Architecture proposée ECDH

Dans cette section, nous présentons maintenant les différentes architectures proposées pour le courbe elliptique sur circuit FPGA. La première architecture illustrée par la Figure 2.4 est basé sur le courbe elliptique avec Diffie-Hillman utilisant la multiplication de point de Montgomery, tandis que la deuxième architecture proposée est basée sur notre nouvel algorithme de multiplication Radix-$2^r$ représenté par la Figure 2.5.

Les deux propositions se composent sur certain module commun telque :

- PRNG : Une fonction pour générer un nombre aléatoire utilisé pour générer les clés privées initiale la graine est une constante a l'instant et peut être remplacer par autre mécanisme).
- Fonction Double (GF2Point) : Une fonction qui double un point *P*, nous avons utilisé la formule (1.4) de chapitre 1.
- Fonction ADD(GF2Point) : Une fonction qui calcule l'addition de deux points *P1* et *P2*, nous avons utilisé la formule (1.3) voir chapitre 1.

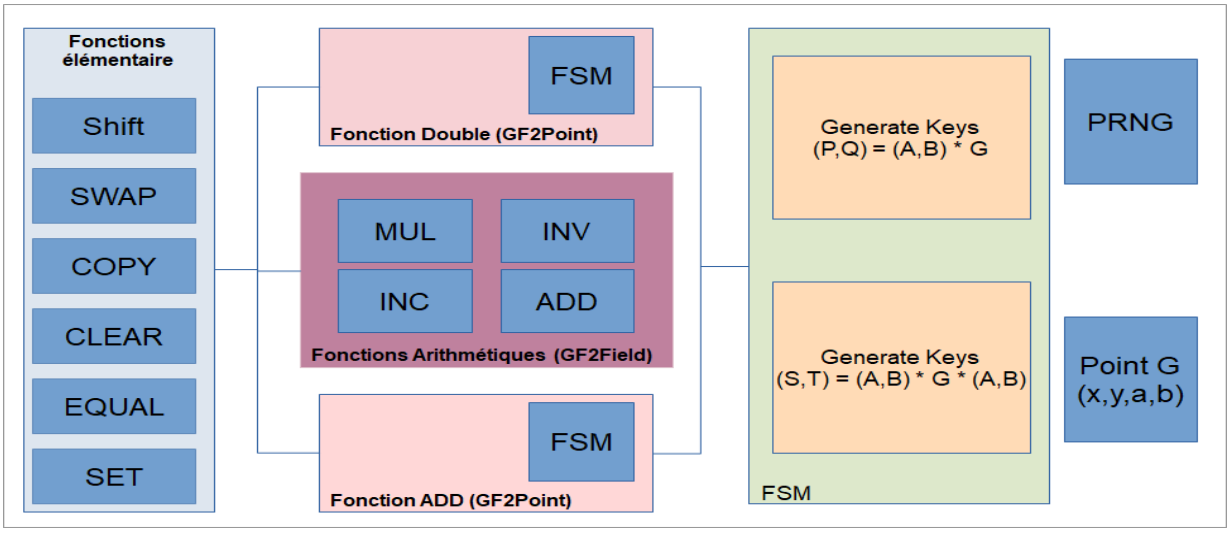



**Figure 2.4: Architecture ECDH basé sur la multiplication de Montgomery**

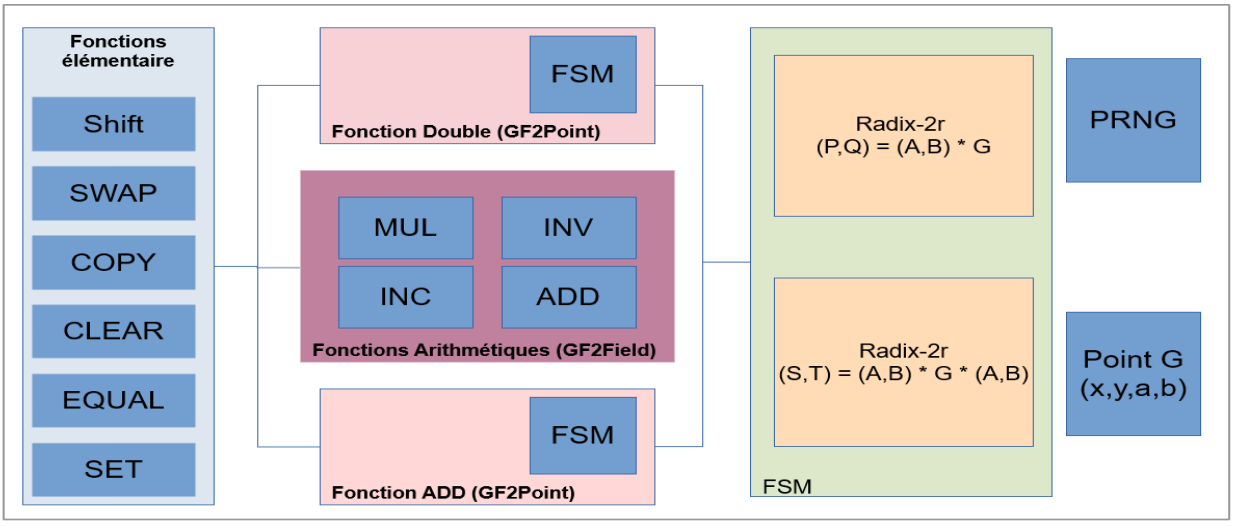



**Figure 2.5: Architecture ECDH basé sur la multiplication Radix-$2^r$**

Nous notons ici que le deux fonction double et addition de point sont basé sur des opérations arithmétiques élémentaires (comme nous avons déjà montré dans le niveau d'abstraction de ECC illustré dans la Figure 2.1). Ces opérations sont la multiplication, (MUL) l'inversion (INV), l'addition (ADD) et incrémentation (INC). De même, toutes ces modules décrits précédemment, utilise dans certain traitement autre opération de manipulation des bits comme le shift, swap, copie, mettre à zéros et a un et vérifie l'égalité.

### 2.4.1 Implémentation de multiplication scalaire

Cette figure ci-dessous représente l'architecture de multiplication scalaire qui nous avons implémenté dans le logicielle Vivado-HLS, les fonctions de la multiplication scalaire sont prises comme des modules tel que :

- L'addition.
- Le doublement.
- Les fonctions élémentaires.

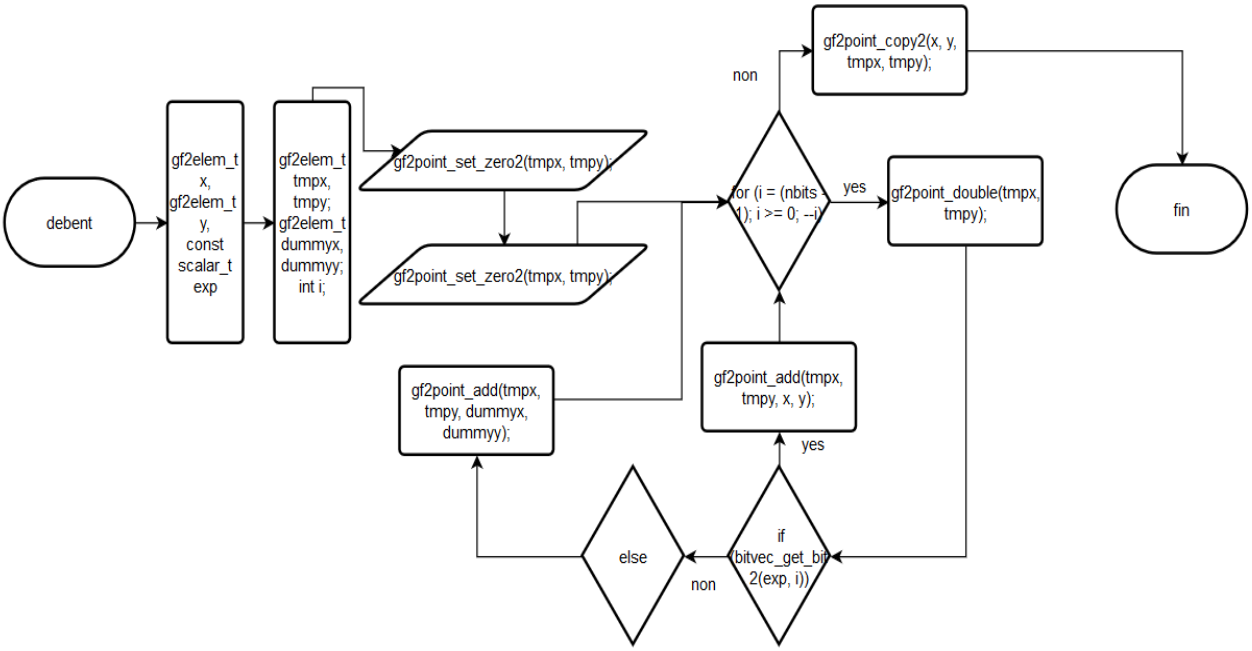



**Figure 2.5: Architecture de multiplication scalaire**

### 2.4.1.1 Opération de doublement

Cette opération est l'un des modules de la multiplication scalaire et y compris des modules comme l'inverse(inv), la multiplication(mul) et l'addition(add).

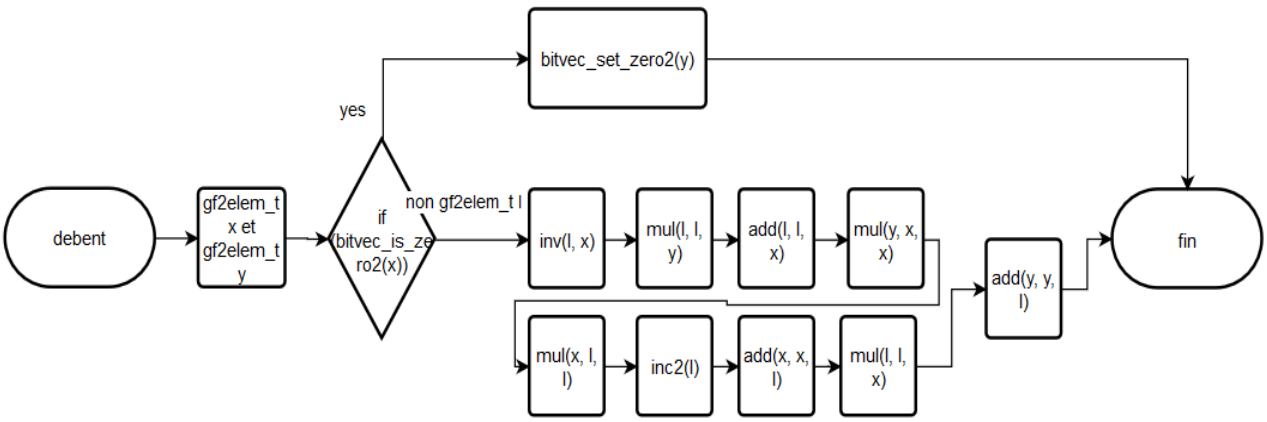



**Figure 2.6: Architecture de doublement**

### 2.4.1.2 Opération d'addition

Cette opération est l'un des modules de la multiplication scalaire et y compris des modules comme l'inverse(inv), la multiplication(mul), l'incrément(inc) et l'addition(add)…ect.

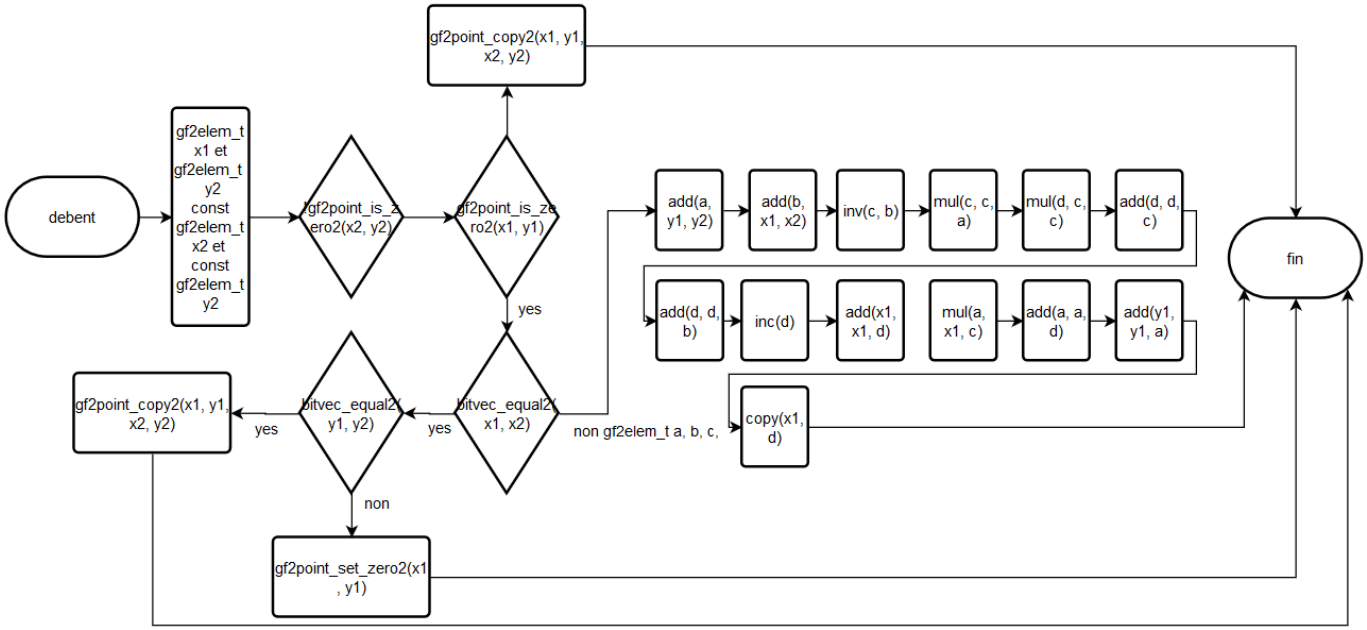



**Figure 2.7: Architecture de l'addition**

## 2.5 Schéma proposé pour chiffrer et déchiffrer avec ECDH_ Radix-$2^r$

La cryptographie à courbe elliptique est utilisée comme système de chiffrement à clé publique pour le chiffrement et le déchiffrement de telle sorte que si l'on doit chiffrer un message, alors ils tentent de convertir le message à un point sur la courbe elliptique. Bien que l'arithmétique impliquée dans la cryptographie à courbe elliptique soit complexe que d'autres algorithmes cryptographiques, la conversion de messages simples en points sur la courbe elliptique a toujours été difficile. Dans cette partie, nous discutons notre schéma de chiffrement et déchiffrement.

### 2.5.1 Description de schéma

Ce schéma est composé de plusieurs étapes, en premier lieu, convertir le texte clair à un point de la courbe elliptique en multipliant la valeur décimale du texte en clair avec la clé secrète partagée *K* de ECDH. Par exemple, nous savons que la valeur du message 'Bonjour' est $P_m(x, y)$ donc elle sera additionnée avec *K* et le résultat est $C_m$=$P_m$+K où K est le point partagé *K (a, b)*. Ce schéma est un schéma simple et rapide.

### 2.5.2 Algorithme de chiffrement et déchiffrement proposée

**Algorithme 2.6: Méthode de chiffrement proposée**

Entrée : Message de chaine de caractère, clé secret partage K (a, b) ;

Sortie :$C_m$=$P_m$+K ;

1. Convertir le message en binaire $M_{bin}$ (ASCII/8 bit) ;
2. Convertir $M_{bin}$ en décimal $M_{déc}$ ; // $M_{déc}$ est le x de point $P_m$
3. Calculer y de $P_m$ ;
4. $C_m$=$P_m$+K ;

Retourner $C_m$ ;

**Algorithme 2.7: Méthode de déchiffrement propose**

Entrée : $C_m(x_1,y_2)$ ;

Sortie :M message claire ;

1. $P_m(x,y)$=$C_m$ −K ;
2. Convertir x en binaire $M_{bin}$ (ASCII/8 bit) ;
3. Convertir $M_{bin}$ en chaine de caractère ;

Retourner M ;

### 2.5.3 Example et explication

Dans l'exemple ci-dessous la courbe elliptique prise *SECP160k2* est une courbe recommandée par NIST et respecte ces règles. Les paramètres sont :

*a :146150163733090291820368483271628301965163755428 8.*

*b :1032640608390511495214075079957864673410201913530.*

*G :(473058756663038503608844550604547710019657059949, 145400849536995165806079869847939590832745324523 0).*

*P :146150163733090291820368483271628301965163755429 1.*

Dans cet exemple, nous avons pris un message clair « Bonjour à tous » est utilisé comme entrée. Le processus global fonctionne comme suit :

Du côté de l'expéditeur :

Apres l'échange de clés ECDH notre résultat est la clé secrète partagée

*K= (853205140651388551691211629545805849481881171235, 190309367476447819220373422222250082631753442025).*

Étape 1 : Convertir le message « Bonjour à tout » en valeur ASCII 8 bits et placez-les dans une liste. Il produit la chaîne de bits suivante : *bin=01000010 01101111 01101110 01101010 01101111 01110101 01110010 00100000 01100001 00100000 01110100 01101111 01110101 01110100.*

Étape 2 : Convertir la valeur *bin* en décimal : *x=134746753190365998889652044322955*6.

Étape 3 : Calculer *y* a partir de *x* : *y=174746123655479125858458575448945*56.

Etape 4 : Calculer $C_m = P_m + K$ ;

$C_m$ *= (137531861420151036434604123876332806744113005869*2, *751268208864084434559361193815353737644274357113).*

Étape 5 : En fin, Envoyer $C_m$ au récepteur.

Du côté de récepteur :

Étape 1 : Calculer $P_m(x, y) = C_m - K$ ;

$P_m$ *= (134746753190365998889652044322955*6, *174746123655479125858458575448945*56).

Étape 2 : Convertir la valeur *x* de $P_m$ en binaire : *bin=01000010 01101111 01101110 01101010 01101111 01110101 01110010 00100000 01100001 00100000 01110100 01101111 01110101 01110100.*

Étape 3 : Finalement, convertir la valeur *bin* en texte : « Bonjour à tout ».

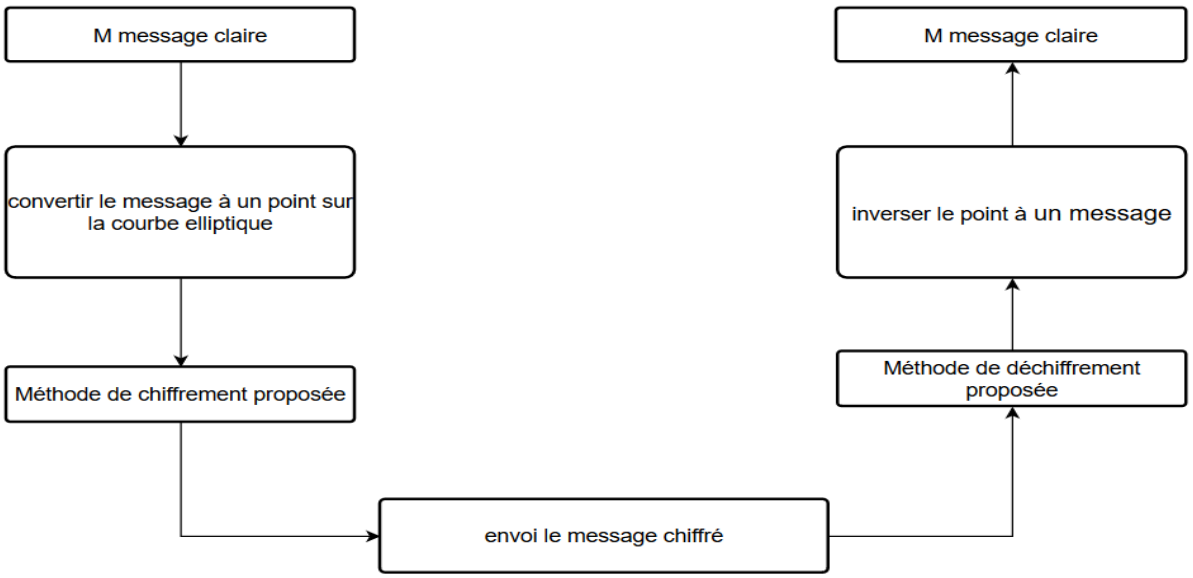



**Figure 2.8: Schéma de chiffrement et déchiffrement proposé**

## 2.6 Conclusion

Dans ce chapitre, nous avons défini la cryptographie dans la littérature sur les courbes elliptiques ensuite nous avons présenté et explique Radix-$2^r$ en détaille en le validant. Présentation de l'architecture de Radix-$2^r$ avec Diffie-Hellman. Enfin nous avons proposé une méthode de chiffrement et déchiffrement.

# Chapitre 3 : Implémentation et test de ECDH_Radix-2r sur FPGA

## 3.1 Introduction

Nous avons montré dans le chapitre précédent le problème d'optimisation de multiplication scalaire dans les courbes elliptiques et l'arithmétique de Radix-$2^r$ qui se représente sa solution. Dans ce chapitre, nous allons détailler la plateforme de conception et test de ECC avec Radix-$2^r$ et l'implémentation logiciel et matériel, nous commençons notre chapitre par expliquer notre plateforme, par la suite, nous présentons l'implémentation logiciel et matériel et nous terminerons par un jeu de test.

## 3.2 Plateforme d'implémentation et test logicielle

Avant de présenter les résultats d'implémentation logicielle de notre ECC basé sur Radix-$2^r$, nous décrivons ici l'environnement de travail et les outils utilisés pour l'implémentation et le développement de notre application.

### a) Caractéristiques de l'ordinateur

**Tableau 3.1: Les caractéristiques de l'ordinateur.**

| **Unité** | **Caractéristiques** |
|---|---|
| **Processeur** | AMD A12-9700P RADEON R7, 10 COMPUTE CORES 4C+6G 2.50 GHz |
| **Mémoire RAM** | 12.0 Go |
| **Disque dur** | 1 Tb HDD |
| **Système d'exploitation** | Windows 10-64 bits |

### b) Langage de programmation

- ***Java :*** est un langage de programmation à usage général, orienté objet dont sa syntaxe est proche du C. Il se caractérise par sa grande sécurité, la richesse de ses bibliothèques et sa simplicité . . . etc., qui font de lui un langage redoutable puissant et performant. Pour cela nous avons choisi java comme langage d'implémentation de notre application.

- ***C++ :*** C'est un langage de programmation créé par Bjarne Stroustrup dans les années 80 permettant la programmation sous de multiples paradigmes, c'est â dire est une programmation orienté objet. C'est un des langages de programmation les plus utilisés dans les applications où la performance est critique grâce à la bonne capacité d'implémentation, d'ailleurs il est connu comme un langage proche au langage de la machine ainsi le langage C.

### 3.2.1 Présentation d'échange de clés avec ECDH_ Radix-$2^r$

Notre but consiste à réaliser une application deux client et serveur dans un réseau local, cette application est une simulation de protocole Diffie-Hillman basée sur la cryptographie sur les courbes elliptiques avec Radix-$2^r$.

Cette fenêtre de notre application représente l'échange de clés entre les deux client Bob et Alice et qui se déroulent dans le serveur.

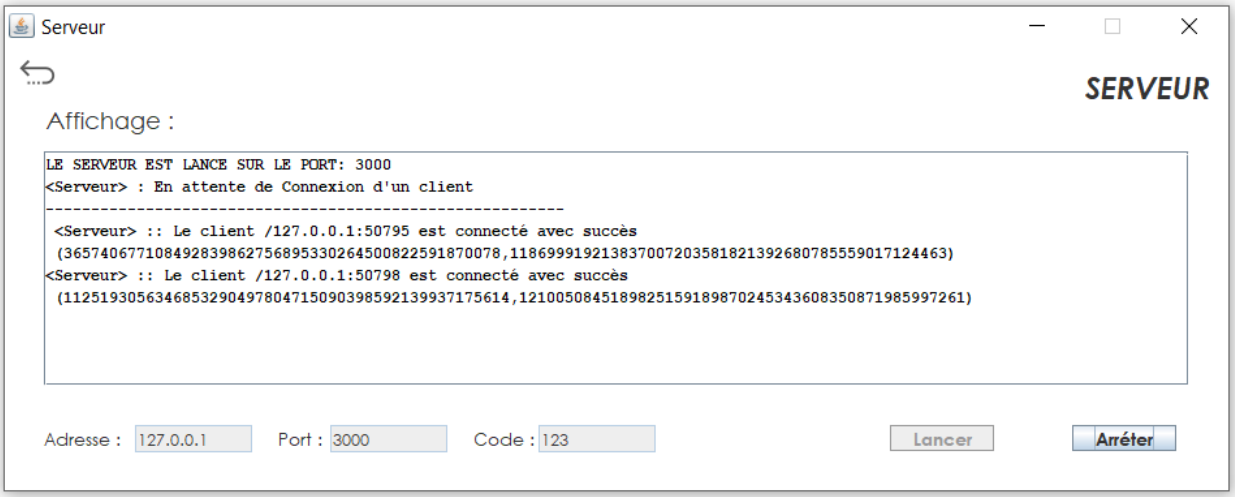



**Figure 3.1: Le serveur au temps d'échange les clés publiques**

### 3.2.2 Présentation du chiffrement et déchiffrement avec ECDH_Radix-$2^r$

Cette interface c'est la fenêtre de la part la cliente Alice qui montre l'envoie et la réception des messages.

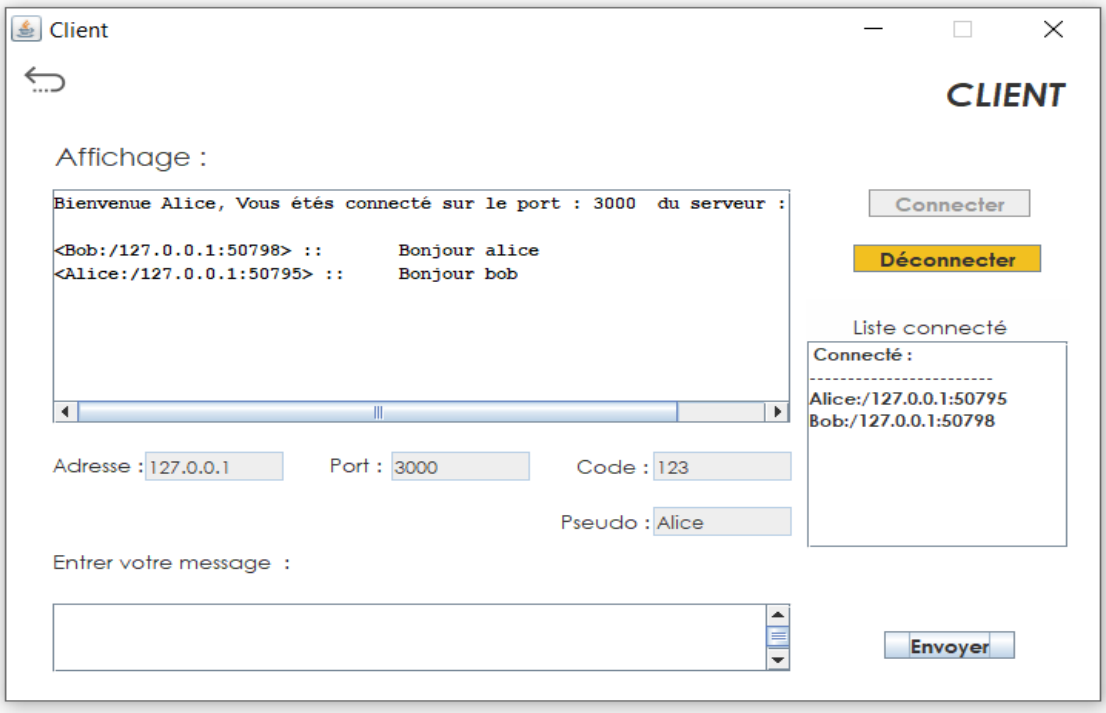



**Figure 3.2: La cliente Alice envoie le message en temps réel**

Cette interface c'est la fenêtre de la part le client Bob qui fait la même chose que Alice dans l'autre côté, il en train d'envoyer et la recevoir des messages.

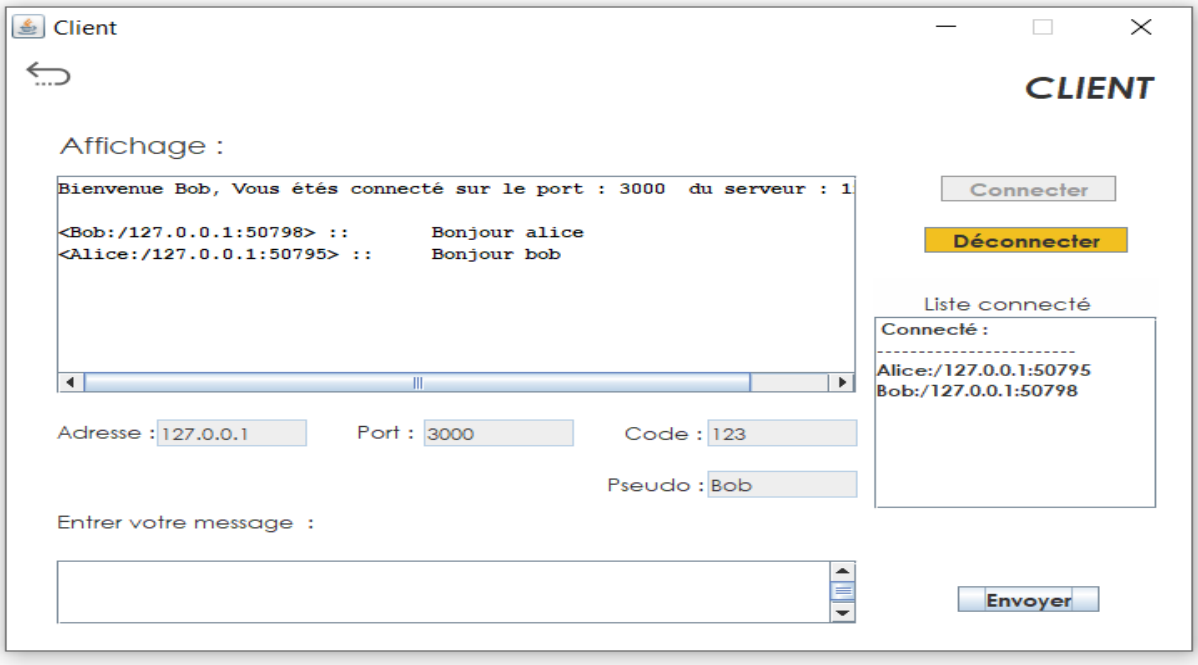



**Figure 3.3: Le client Bob envoie le message en temps réel**

Cette dernière interface montre les messages chiffrés de la communication entre Bob et Alice dans la fenêtre de serveur.

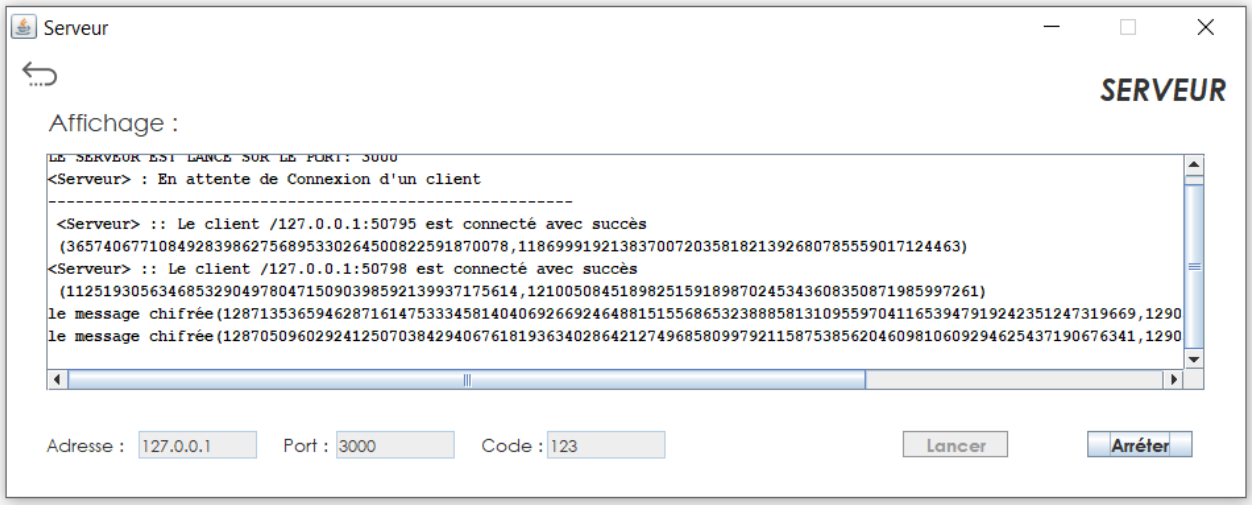



**Figure 3.4: Le serveur au temps d'échange les messages chiffrés**

### 3.2.3 Test et performances

Les tests effectués visent à évaluer les performances de notre solution en termes de temps, taille et nombre de ligne de code. Nous donnons ici la comparaison de notre résultat d'exécution ECC avec Radix-$2^r$ et ECC avec méthode binaire voir le tableau ci-dessous.

**Tableau 3.2 : Comparaison du temps d'exécution et d'accélération.**

| **Type d'optimisation** | **Multiplication Binaire** | **Multiplication Radix-$2^r$** |
|---|---|---|
| **Temps d'exécution en seconde** | 231654 | 4782 |
| **Taille de code dans mémoire en octets** | 107004796 | 137887744 |
| **Nombre de lignes de code** | 4901 | 5058 |

## 3.3 Implémentation et test matérielle de ECDH sur Vivado HLS

### 3.3.1 plateforme de conception et test utilisant Vivado HLS

Après avoir testé nos architectures proposées de ECC et ces applications de Diffie-Hellman et le chiffrement en terme d'implémentation logicielle. Dans cette section, ont présente nos implémentations matérielles de ces derniers dans le circuit FPGA. Suite à cela, nous utilisons le logicielle Vivado HLS de Xilinx qui nous permet de convertir nos architectures logicielles proposé en matérielle suivant certain guide et optimisation dans le code source originale.

Suite à cela, voici quelques guides que nous avons utilisé à suivre :

- Il faut séparer la fonction principale a implémenté a d'autre code qui vas utiliser après que dans les tests qui vas être appelé le test Bench.
- Les entrés et sortie de fonction principale doit être attaché avec un bus de communication matérielle dite AXI compatible aux systèmes sur puce Zynq-ARM,
- Puisque les arguments sont des tableaux a dimension, le bus de communication doit aussi connecter avec des mémoires matériels interne de FPGA,
- Les boucles indépendantes, peut être modifie pour exécuter en parallèle ou pipeline,
- Les tableaux d'une seule ou multiple dimension ne devront pas être utilisés comme des entrée/sortie en même temps, il faut les séparer,
- Finalement, Vivado HLS disposeras des fonction nommé PRAGMA peut être insérer directement dans le code pour certain conversion complexe qui nécessite plus de codage manuel.

### 3.3.2 Résultat de l'implantation matérielle

Le Tableau 3.3 illustre les résultats d'implémentation de ECC basé sur la multiplication de point de Montgomery avec Vivado HLS. Les résultats représentent les ressources matérielles utilisé une fois programmer dans le FPGA en terme de bloc élémentaire comme le table de consultation et les registres utilisé dans des opérations de calcules, utilisation de mémoire pour enregistrer les résultat temporaire ou finale, ou l'utilisation des bloc prête en cas d'opération mathématique complexe comme la multiplication et division. Finalement, le module complet ECC implémenté sur FPGA consomme 58 des blocs mémoire RAM (soit 20 % de ressource), 9359 des registres (soit 8 % de ressource), finalement 26724 de LUT (soit 50 % de ressource). Nous notons ici qu'on a utilisé 0 bloc de DSP, car toutes les opérations de MUL/DIV sont implémenté manuellement. Un dernier résultat, est la fréquence de fonctionnent matérielle de notre module ECC estimé à 148,32 Mhz (soit un période de 6,742 nano second).

**Tableau 3.3 : Résultats d'implémentation de ECC sur FPGA**

| | **BRAM_18K** | **DSP** | **FF** | **LUT** |
|---|---|---|---|---|
| **DSP** | - | - | - | - |
| **Instance** | 56 | - | 9229 | 26686 |
| **Mémoire** | 2 | - | 128 | 6 |
| **Ressource totale** | 58 | 0 | 9359 | 26724 |
| **Ressource existant** | 280 | 220 | 106400 | 53200 |
| **Ressource utilisées** | **20 %** | **0 %** | **8 %** | **50 %** |

Cependant, les tableaux suivants illustrent plus de détaille de ressource matérielle des sous blocs des différents niveaux d'abstraction de ECC (Figure 2.1).

Le premier tableau illustre les ressources matérielles de multiplication de point de ECC selon l'algorithme de Montgomery 2.2. Il est clair que l'opération ADD consomme plus de ressource que le DBL, puisque ont trouvé l'opération de DBL intégrer aussi. Une chose que le Radix-$2^r$ n'as pas.

**Tableau 3.4 : Résultats d'implémentation matérielle de la multiplication scalaire dans ECC**

| **Fonction de point** | **LUT** | **FF** | **BRAM** |
|---|---|---|---|
| **Multiplication de point** | 25293 | 8663 | 56 |
| **Double point DBL** | 7285 | 2592 | 14 |
| **Addition point ADD** | 17035 | 5769 | 34 |

Le dernier tableau illustre les ressources en détaille de chaque opération de multiplication de point soit DBL ou ADD en terme d'utilisation des opérations élémentaires les plus complexe comme la multiplication et l'inversion. Ces résultats confirment le tableau précédent sur le poids d'optimisé l'utilisation de ces opérations comme la multiplication. Dans la multiplication basée sur Montgomery, ces résultats résume *L-1* (*L=163*) opérations de DBL et ADD, tandis que dans le Radix-$2^r$ est réduit.

**Tableau 3.5: Résultats d'implémentation matériel des fonctions arithmétiques élémentaires utilisées dans le doublement et l'addition**

| **Fonctions élémentaires** | | **LUT** | **FF** | **BRAM** |
|---|---|---|---|---|
| **Double (DBL)** | **Multiplication** | 2369 | 970 | 4 |
| | **Inversion** | 3347 | 1442 | 4 |
| **Addition (ADD)** | **Multiplication** | 3582 + 7285 DBL | 1460 + 2592 DBL | 6 + 14 DBL |
| | **Inversion** | 3347 | 1442 | 4 |

### 3.3.3 Résultat de simulation matérielle sur FPGA

#### a) Génération de clé ALICE

***- Simulation logicielle :***

Clé prive de Alice : 2F4653B826675E49726B53E49955999BE394760F291871F

Clé publique de Bob : (C4956255 7A69CAE6 CCBF828D A22D2DA 5DA3A22A 2, D958B5 19B53581 AFE1ADD B683A20F 67ACC15D 3)

***- Résultat de simulation matérielle sur FPGA :***

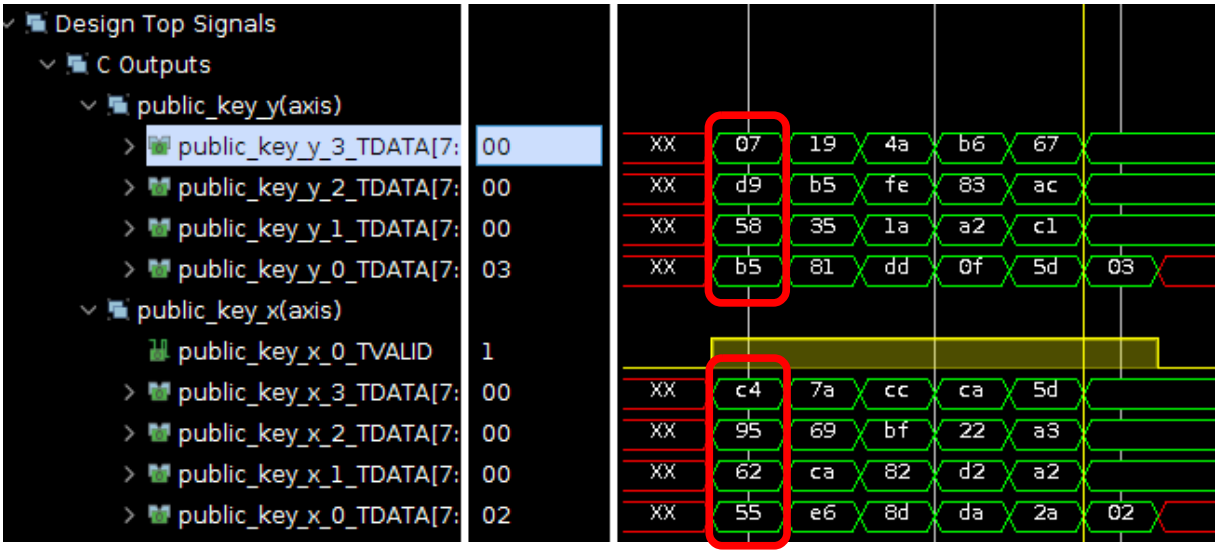



**Figure 3.5: Résultat de simulation matérielle sur FPGA**

**b) Génération de clé Bob**

***- Simulation logicielle :***

Clé prive de Bob : 8F698F5A1DC22D843EC0CAF8945239977D514FDA58AE16

Clé publique de Bob : (37B50455 64AFC9A FADDF6E4 960BA89 FE9E7F1 3, E0A3B814 EDC3F0E7 D9685415 EF567C2D 2F447F51 7)

***- Résultat de simulation matérielle sur FPGA :***

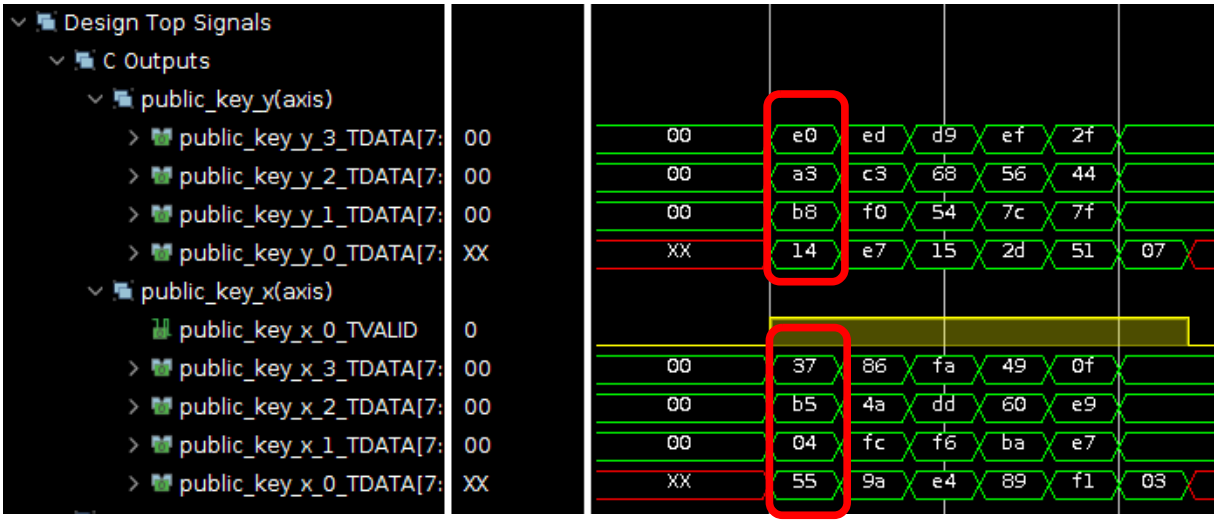



**Figure 3.6: Résultat de simulation matérielle sur FPGA**

## 3.4 Implémentation et test matérielle de ECDH sur SoC

Nous proposons deux plateformes de test et validation matérielle. La première plateforme consiste à l'intégration nos solutions dans un système embarqué basé sur le système sur puce (SoC) de type Zynq et processeur ARM avec différent périphériques. Tandis que la deuxième plateforme exploite un nouveau système de test développer par le CDTA pour déboguer les blocs conçus sans passé par un SoC. Ont noté ici que la deuxième plateforme est aussi compatible à SoC de Zynq et peut être aussi intégré avec son bus de communication commun.

La figure 3.7 illustre le système sur puce SoC de type Zynq-7000 utilisé pour développer notre système embarqué sécurisé avec ECC. Il est composé de deux partie : la première partie est basé sur le processeur ARM Cortex®-A9 double cœur (667 MHz) totalement reconfigurable et un bus d'interconnexion de type AMBA AXI-4 qui peut communiquer avec les périphériques déjà existé dans le system comme DDR, ETH, PCI … ect. Tandis que la deuxième partie est la partie reprogrammable de FPGA qui contient toutes nos blocs conçus comme le ECC. Finalement ce système permet de créer des conceptions basées sur Linux, Android, Windows ou tout autre système d'exploitation/système d'exploitation en temps réel (RTOS).

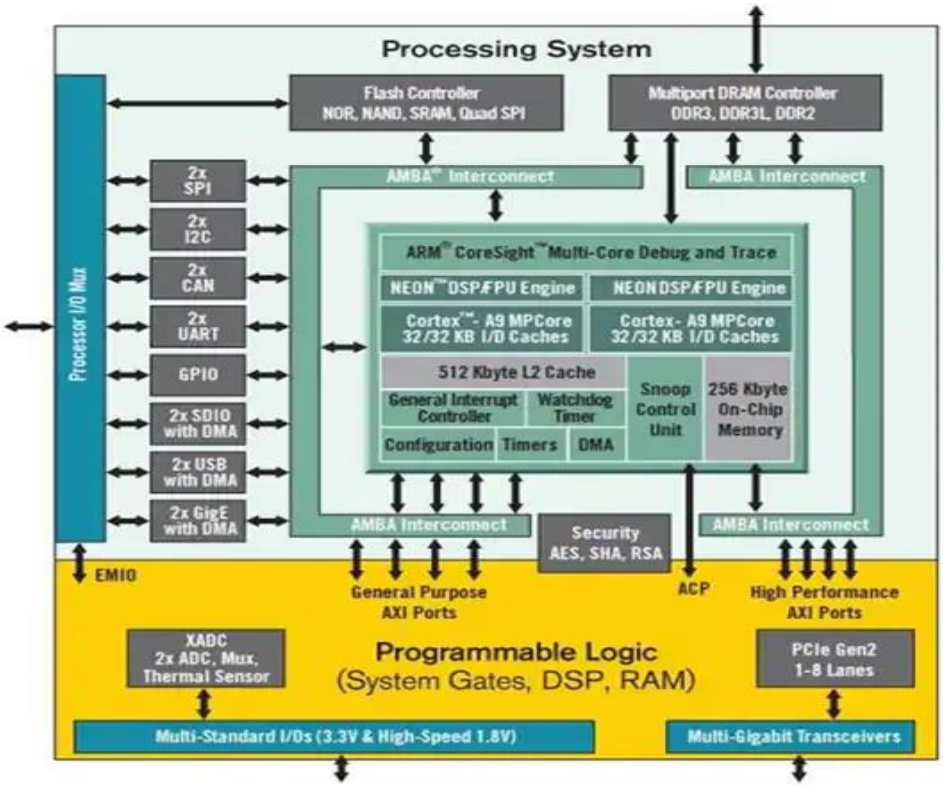



**Figure 3.7 : Système sur puce Zyn-7000 basé sur processeur ARM Cortex®-A9 double cœur**

### 3.4.1 Méthodologie de conception avec SoC

Cette plateforme de conception et test consiste a utilisé une méthodologie de conception de système embarqué basé sur un système sur puce SoC de type Zynq présenter par le vendeur de FPGA Xilinx. Elle est divisée en deux parties : la première partie résume la conception matérielle de système embarqué basé sur un système sur puce utilisant le logicielle Vivado de Xilinx. Ces outils consiste a sélectionné les blocs matériels qui constitue le SoC comme le processeur ARM, DDR et nos modules de chiffrement ECC. Une fois le système est construit, nous passons à l'étape de conversion nommé synthèse de ces blocs décris en langage de hauts niveaux a des blocs électroniques logiques basées sur la technologie de FPGA choisis puis générant finalement un fichier binaire « Bitsream » qui vas être programmé dans le FPGA.

La deuxième partie de plateforme consiste au développement de partie commande logicielle de SoC. Ensuite, nous utilisons Vivado SDK pour mapper les adresses des modules utilisées avec l'application de test. Un fois l'application est compilée avec le compilateur arm-gcc, la sortie vas être combiné avec le Bitsream de matérielle générer, formulant finalement un fichier binaire qui seras programmé sur FPGA.

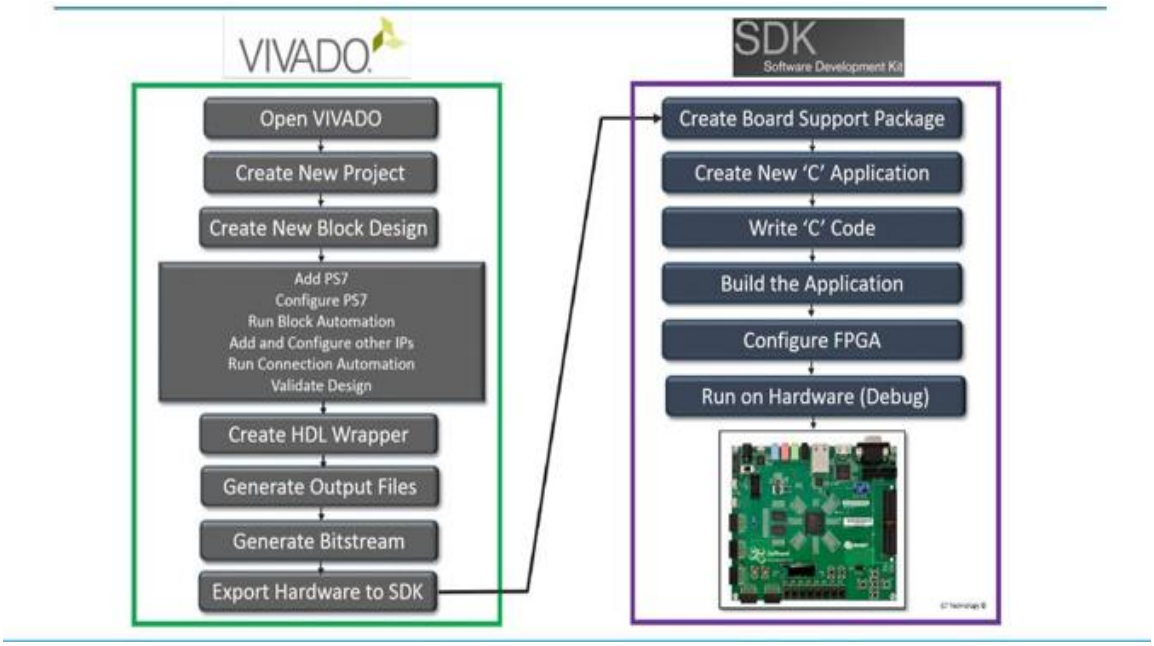



**Figure 3.8: Plateforme de conception de système embarqué avec Zynq et Vivado**

Pour les expériences, la plateforme de test est conçue et mise en œuvre à l'aide des outils Xilinx Vivado et de deux cartes prototypes FPGA, à savoir la carte ZYBO.

### 3.4.1.1 Plateforme matérielle

La nouvelle plateforme présente un matériel alternatif et un concept de test de Zynq qui a été proposé dans des précédents travaux de recherche **[29]**. Cependant, maintenant, la plateforme devient totalement indépen1dante de n'importe qu'il processeur (Zynq) et elle est entièrement reconfigurable avec un logiciel principal. Il est également basé sur AXI-4, ce qui le rend flexible et facile à intégrer avec la plateforme Zynq pour les applications SoC.

Le système sur puce SoC proposé est basé sur un processeur ARM Cortex A9 duelle core communiquant avec plusieurs périphériques comme une mémoire DDR4, une unité de transfert série UART vers le PC, des entrée et sortie vers horloge, mise a zéros de carte et commande mécanique (switch) et un bus de communication de type AXI-4 qui relier toutes les blocs suivants le master et le slave.

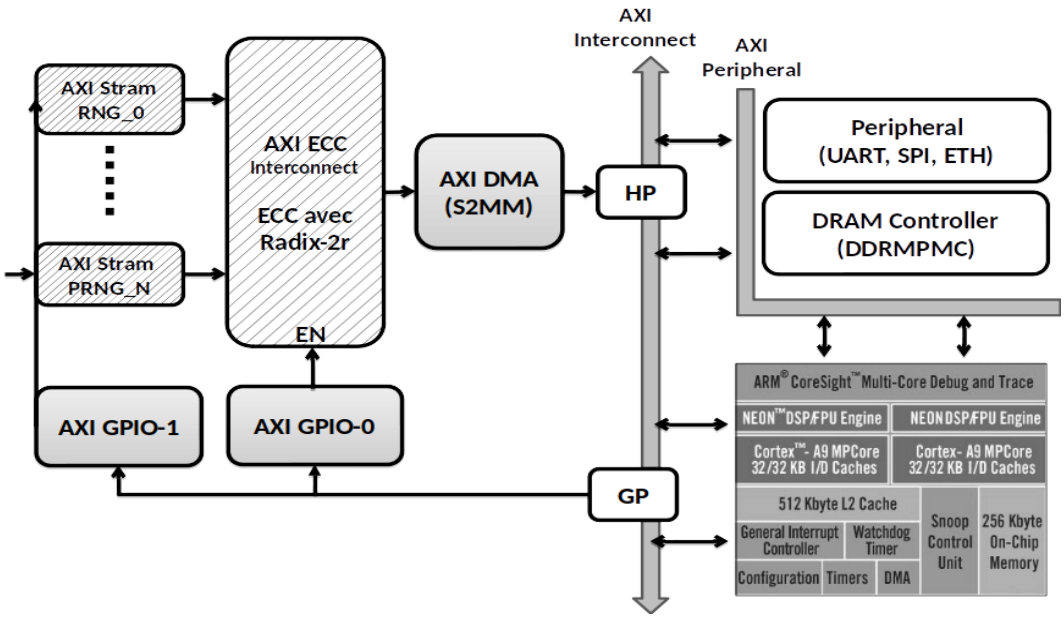



**Figure 3.9: Architecture matériel du SoC ARM de type Zynq avec ECC**

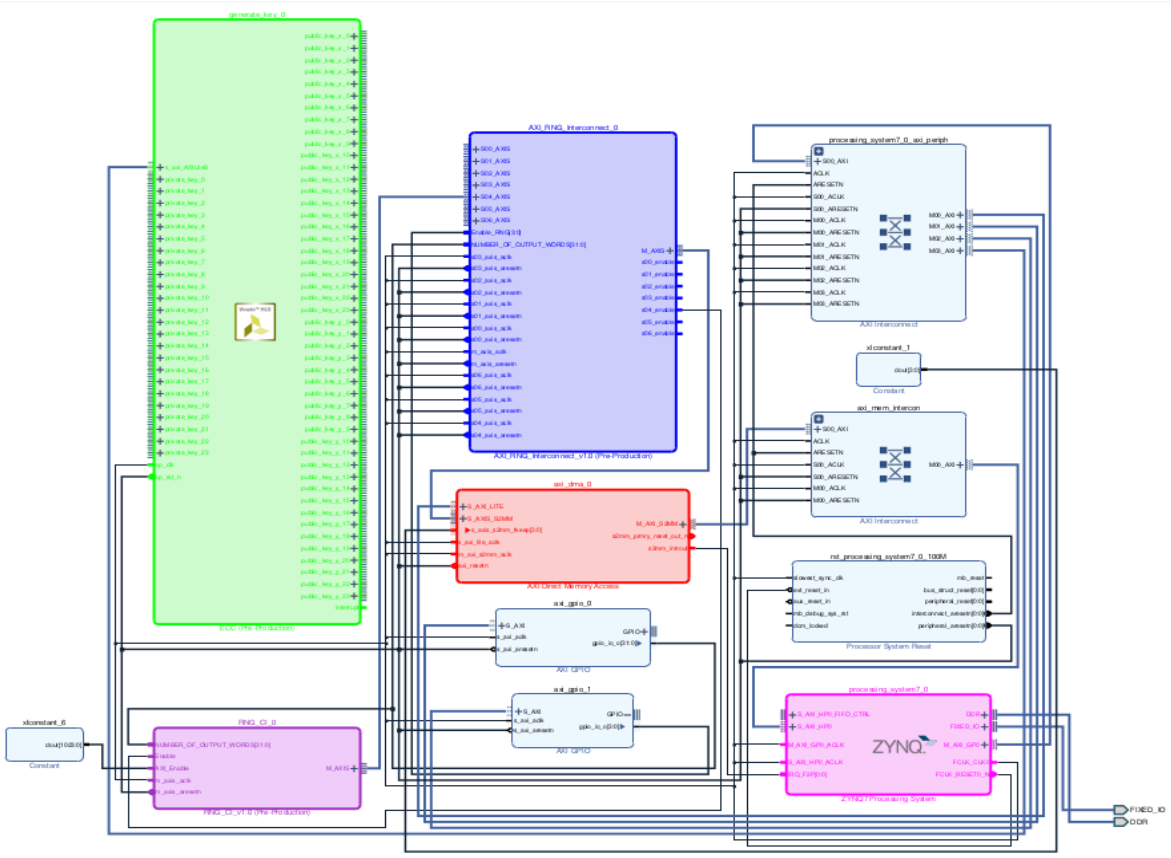


**Figure 3.10: Implémentation matérielle de module de chiffrement ECC**

Une fois le système sur puce est construit avec notre module de ECC, l'outil généré un tableau d'adresse (Figure 3.11) de tous les éléments utilisés et qui vont être utilisé dans le prochain outil nommé SDK pour construire notre application (ECC = 0x43C00000).

| Name | Interface | Slave Segment | Master Base Address | Range | Master High Address |
|---|---|---|---|---|---|
| Network 0 | | | | | |
| /axi_dma_0 | | | | | |
| /axi_dma_0/Data_S2MM (32 address bits : 4G) | | | | | |
| /processing_system7_0 | S_AXI_HP0 | HP0_DDR_LOWOCM | 0x0000_0000 | 512M | 0x1FFF_FFFF |
| /processing_system7_0 | | | | | |
| /processing_system7_0/Data (32 address bits : 0x40000000 [ 1G ]) | | | | | |
| /axi_dma_0 | S_AXI_LITE | Reg | 0x4040_0000 | 64K | 0x4040_FFFF |
| /axi_gpio_0 | S_AXI | Reg | 0x4120_0000 | 64K | 0x4120_FFFF |
| /axi_gpio_1 | S_AXI | Reg | 0x4121_0000 | 64K | 0x4121_FFFF |
| /generate_key_0 | s_axi_AXILiteS | Reg | 0x43C0_0000 | 64K | 0x43C0_FFFF |

**Figure 3.11: Table d'adresse des éléments utilisés**

### 3.4.1.2 Implémentation de l'application sur SoC

Le kit de développement logiciel (SDK) de Xilinx fournit un environnement pour créer des plateformes logicielles et des applications pour les processeurs embarqués Xilinx. Le SDK fonctionne avec les conceptions matérielles créées avec Vivado et il est basé sur le standard open source Eclipse. Les fonctionnalités du SDK incluent (Figure 3.12) : un éditeur de code C/C++ riche en fonctionnalités et environnement de compilation, le driver de processeur ARM et les différents blocs matérielles de SoC utilisé comme les adresses (fichier xml), initialisation et les interruptions et finalement les outils de compilation et débogage utilisant la toolchain de ARM et une console d'affichage des entrées et sorties.

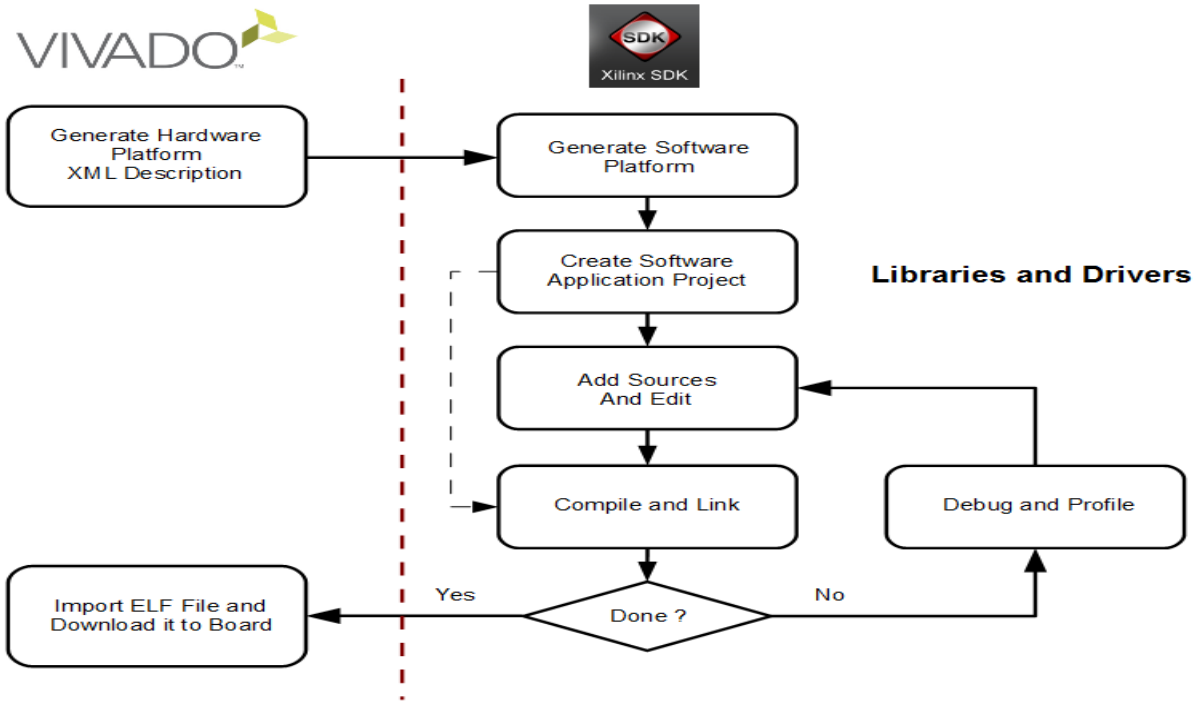



**Figure 3.12: Plateforme de conception de l'application test de ECC sur FPGA**

Cela dit, le firmware se compose de trois parties principales et peut être résumé comme suit : Premièrement, la communication UART établit une communication série entre la plateforme et le PC (ouvre et ferme un port série virtuel USB). Afin d'établir cela, le firmware doit être synchronisé avec la configuration matérielle de l'UART implémentée dans FPGA. La deuxième partie est la configuration des registres internes de plateforme des différentes opérations pour la configuration initiale. En d'autres termes, il exécute une série d'opérations de lecture et d'écriture en parallèle.

Concernant l'opération d'écriture, il cible les registres internes pour la configuration, comme suit : définit s'il écrit ou lit des opérations dans des états internes, définit la latence, sélectionne quel type de clés à générer de l'application Diffie-Hillman (privé ou partagé). L'opération de lecture, par exemple, capture tout type de réponses de la plateforme après l'opération d'écriture. Enfin, la troisième partie exécute le ECC dans Diffie-Hellman comme un générateur de clé partagé *(Q= ka*P)* en gardant à l'esprit que l'écriture et la lecture s'exécutent en parallèle.

### 3.4.1.3 Résultats de l'implémentation

Le tableau 3.13 illustre les résultats de l'implémentation matérielle de de ECC sur SoC. Nous notons que ECC consomme plus de ressource par a port a d'autre blocs utilisés.

| Name | Slice LUTs (17600) | Slice Registers (35200) | F7 Muxes (8800) | Slice (4400) | LUT as Logic (17600) | LUT as Memory (6000) | Block RAM Tile (60) |
|---|---|---|---|---|---|---|---|
| design_1_wrapper | 10838 | 8856 | 4 | 3389 | 10296 | 542 | 31 |
| design_1_i (design_1) | 10838 | 8856 | 4 | 3389 | 10296 | 542 | 31 |
| axi_dma_0 (design_1_axi_dma_0_0) | 846 | 1292 | 0 | 376 | 778 | 68 | 1 |
| axi_gpio_0 (design_1_axi_gpio_0_1) | 127 | 382 | 0 | 94 | 127 | 0 | 0 |
| axi_gpio_1 (design_1_axi_gpio_1_0) | 37 | 48 | 0 | 14 | 37 | 0 | 0 |
| axi_mem_intercon (design_1_axi_mem_intercor | 334 | 371 | 0 | 127 | 306 | 28 | 0 |
| AXI_RNG_Interconnect_0 (design_1_AXI_RNG_Int | 152 | 101 | 0 | 71 | 152 | 0 | 0 |
| generate_key_0 (design_1_generate_key_0_0) | 8748 | 5965 | 4 | 2561 | 8364 | 384 | 30 |
| processing_system7_0 (design_1_processing_s | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| processing_system7_0_axi_periph (design_1_p | 578 | 660 | 0 | 228 | 517 | 61 | 0 |
| rst_processing_system7_0_100M (design_1_rst | 18 | 37 | 0 | 15 | 17 | 1 | 0 |

**Figure 3.13: Les résultats d'implémentation de ECDH**

Cependant, la consommation de puissance est de 1,746W de tout le système, soit 1,625W (93%) consommation de puissance dynamique en cours d'exécution et 0,121W (7%) en statique ou en mise en veille. Finalement, la Figure 3.15 illustre que notre module ECC consomme 0,102W soit 6 % de puissance dynamique totale.

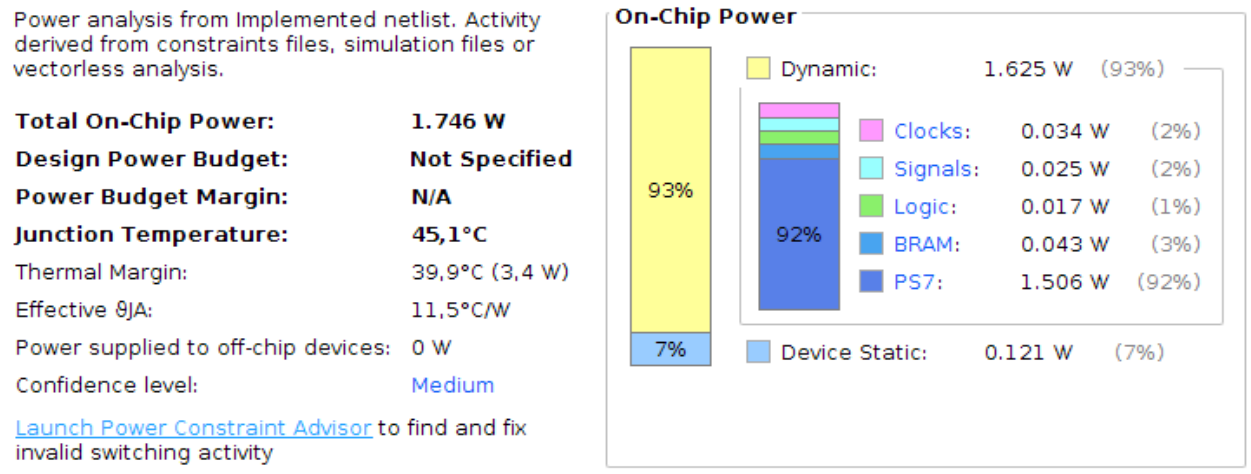



**Figure 3.14: Consommation de puissance dynamique et statique de SoC avec ECC**

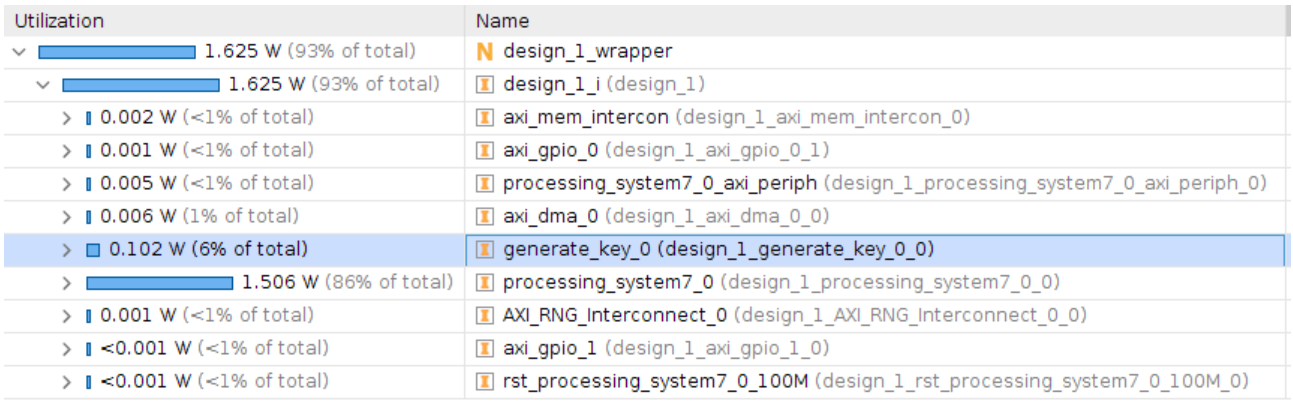



**Figure 3.15: Consommation de puissance dynamique de SoC avec ECC**

## 3.5 Comparaison et discussion

Dans cette partie nous allons voir les résultats précédents voir chapitre 1 (Tableau 1.3 ) et comparer avec notre travail, nous ne pouvons pas augmenter la fréquence d'exécution et les LUT et FF au même temps dans un programme, nous allons soit diminuer les LUT et FF avec une fréquence faible ou augmenter la fréquence et en parallèle les LUT et FF augment, sachant que la facture plus importante c'est la fréquence d'exécution (vitesse), notre travail offre une meilleure fréquence d'exécution par rapport aux autres implémentations.

**Tableau 3.6 : Table de cas d'implémentation ECC sur FPGA avec notre solution [30]**

| | **FPGA** | **Fréquence (Mhz)** | **LUT** | **FF** | **Courbe** |
|---|---|---|---|---|---|
| **Sinha Roy** | Virtex 5 | 0,105 | 10195 | - | 163 |
| **Ansari** | Virtex 2 | 0,0243 | 8300 | 1100 | 163 |
| **Liu** | Virtex 4 | 0,111 | - | - | 163 |
| **Luts** | Virtex 2 | 0,002 | 7362 | 1930 | 163 |
| **Loi** | Virtex 4 | 0,001 | 3815 | 1219 | 163 |
| **Sutter** | Virtex 5 | 0,05 | 22340 | - | 163 |
| **Chelton** | Virtex 4 | 0,05 | - | - | 163 |
| **Notre travaille** | **Zybo** | **1,4832** | **26724** | **9359** | **163** |

## 3.6 Conclusion

Dans ce chapitre, nous avons présenté l'environnement de travail logiciel et matériel et les différents résultats obtenus par l'implémentation de l'algorithme proposée. Pour finaliser cette partie nous avons ajouté des captures d'écran de notre application qui présentent les principales fonctionnalités de cette dernière.

# Conclusion générale

Toutes les solutions multiplication scalaire développées dans les courbes elliptiques ont prouvé qu'il y a toujours des vulnérabilités dans les systèmes embarqués et qui elle ne sont toujours pas suffisantes. C'est pour cela nous nous sommes intéressés au problème de l'implantation de la multiplication de point par un scalaire sur les courbes elliptiques.
Nous avons abordé ce problème aussi bien au niveau arithmétique et algorithmique une nouvelle solution, toutefois cette dernière permet un meilleur fonctionnement de performance en optimisant la consommation de ECC.

Nous avons travaillé dans deux environnement logiciel et matériel. Au niveau logiciel (java) nous avons créé deux clients et un serveur, après la création d'une topologie pour tester les performances de notre algorithme de chiffrement nous avons obtenu des résultats de déroulement dans le réseau.

L'exécution de notre application nous a aidé beaucoup à remarquer et voir les avantages dans le système jusqu'à arriver à un haut niveau de sécurité sachant que dans l'informatique tout ce qu'est nouveau est vulnérable alors ça reste une solution efficace en ce moment voyant les résultats obtenus et reste à améliorer toujours.

Dans notre travail nous avons traité quelque type d'attaque et nous avons choisi « Homme de milieu » pour tester notre contre mesure en implémentant les techniques nécessaires pour contourner les données en échange entre les clients dans un réseau.

# Perspectives

Les perspectives futures sont dans un premier temps tester physiquement la multiplication scalaire proposée Radix-$2^r$ avec ECC sur FPGA afin d'arriver aux résultats attendus, La deuxième perspectives consiste à réaliser l'architecture ci-dessous qui représente la création d'un Shell Linux embarqué sur le SoC Zynq avec un bloc sécurisé basé sur ECC et Radix-$2^r$ comme un pilote (driver).

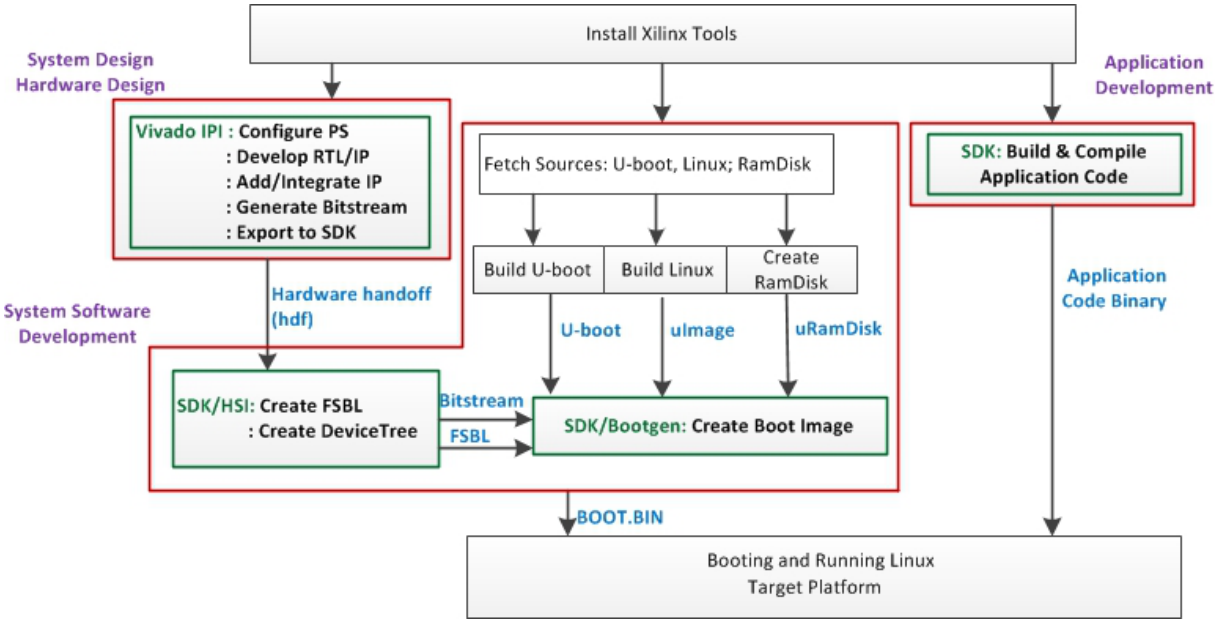



**Figure 4 : Plateforme de conception et test d'un système embarqué Linux sur FPGA**

# Bibliographies

**[1]** Guy Pujolle : Les Réseaux - Eyrolles (6ème Éd) 2008.

**[2]** Travis Spann. Fault induction and environmental failure testing, 2005.

**[3]** Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks, 2002.

**[4]** Adi Shamir and Aran Tromer. Acoustic cryptanalysis: On nosy people and noisy machines, 2004.

**[5]** Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. 2005.

**[6]** Neal Koblitz. Elliptic curve cryptosystems. Mathematics of computation, 48(177): 203–209, 1987.

**[7]** Victor S Miller. Use of elliptic curves in cryptography. In Advances in Cryptology—CRYPTO 85 Proceedings, pages 417–426. Springer, 1986.

**[8]** Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). International Journal of Information Security, 1(1) :36–63, 2001.

**[9]** Whitfield Diffie and Martin Hellman. New directions in cryptography. Information Theory, IEEE Transactions on, 22(6) :644–654, 1976.

**[10]** Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2): 120–126, 1978.

**[11]** Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and SheuelingChang Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In Marc Joye and Jean-Jacques Quisquater, editors, Cryptographic Hardware and Embedded Systems - CHES 2004, volume 3156 of Lecture Notes in Computer Science, pages 119–132. Springer Berlin Heidelberg, 2004.

**[12]** Yanbo Shou. Cryptographie sur les courbes elliptiques et tolérance aux pannes dans les réseaux de capteurs. Université de Franche-Comté, 2014.

**[13]** Certicom Research. Sec 2: Recommended elliptic curve domain parameters, September 2000.

**[14]** Nicholas Jansma and Brandon Arrendondo. Performance comparison of elliptic curve and rsa digital signatures, April 2004.

**[15]** Arjen K Lenstra and Eric R Verheul. Selecting cryptographic key sizes. Journal of cryptology, 14(4) :255–293, 2001.

**[16]** D. Hankerson et al., “Guide to Elliptic Curve Cryptography “, Springer, 2004.
National Institute of Standards and Technology (NIST), Digital Signature Standard, FIPS Publication 186-2, February 2000.

**[17]** Neal Koblitz. Elliptic curve cryptosystems. Mathematics of Computation, 48(177) :203_209, January 1987.

**[18]** Philip M Freidin. Logic block with look-up table for configuration and memory, May 9 1995. US Patent 5,414,377.

**[20]** D.R. Hankerson, S.A. Vanstone, and A.J. Menezes. Guide to elliptic curve cryptography. Springer-Verlag New York Inc, 2004.

**[21]** Montgomery speeding the pollard and elliptic curve methods of factorization. MathematicsofComputation,48:243-264,1987.

**[22]** A. Booth. Asigned binary multiplication technique. The Quarterly Journal of Mechanics and Applied Mathematics, 4:236-240,1951.

**[23]** M. JoyeandS. -M.Yen.Optimal left to right binary signed digit recoding. IEEE Transactions on Computers,49(7):740-748,2000.

**[24]** B. Phillips and N.Burgess. Minimal weight digit set conversions. IEEE Transactions on Computers, 53(6): 666-677,2004.

**[25]** H. Sam, and A. Gupta, "A Generalized Multibit Recoding of Two"s Complement Binary Numbers and its Proof with Application in Multiplier Implementation," IEEE Trans. on Computers, vol. 39, N° 8, August 1990.

**[26]** Abdelkrim. Kamel. Oudjida, "Binary Arithmetic for Finite-Word-Length Linear Controllers: MEMS Applications" Thèse de Doctorat, Ecole Doctorale Sciences Physiques pour l"Ingénieur et Microtechnique, Jan 2014.

**[27]** Abdelkrim. Kamel. Oudjida, "RADIX-2r: A New Multiple Constant Multiplication Algorithm", 27 May 2015.

**[28]** A.K. Oudjida and A. Liacha, "Rapid, Memory-Efficient, and Secure Algorithm for Scalar Multiplication in Elliptic Curve Cryptography ", paper under review in IEEE Trans. on Computers.

**[29]** BAKIRI, Mohammed, COUCHOT, Jean-François, et GUYEUX, Christophe. CIPRNG: A VLSI family of chaotic iterations post-processings for $\mathbb {F} _ {2} $-linear pseudorandom number generation based on zynq mpsoc. IEEE Transactions on Circuits and Systems I: Regular Papers, 2017, vol. 65, no 5, p. 1628-1641.

**[30]** GÖVEM, Burak, JÄRVINEN, Kimmo, AERTS, Kris, et al. A fast and compact FPGA implementation of elliptic curve cryptography using lambda coordinates. In: International Conference on Cryptology in Africa. Springer, Cham, 2016. p. 63-83.

**[31]** Bellemou, A. M., Benblidia, N., Anane, M., & Issad, M. (2020). Microblaze-based parallel implementations of elliptic curve scalar multiplication over $F_p$ on FPGA. International Journal of Internet Technology and Secured Transactions, 10(1/2), 171.

# Annexe mathématique

## I Arithmétique des Courbes Elliptiques

Nous étudions dans cette annexe les notions mathématiques dont nous avons besoin pour comprendre son fonctionnement.

### I.1 Groupe

En mathématique, un groupe est un couple $(E, \cdot)$ où $E$ est un ensemble et $\cdot$ est une loi de composition interne qui combine deux éléments $a$ et $b$ de $E$ pour obtenir un troisième élément $a \cdot b$. Il faut que la loi satisfasse les quatre axiomes ci-dessous **[12]**.

– Fermeture : $\forall (a, b) \in E \mid a \cdot b \in E$

– Associativité : $\forall (a, b) \in E \mid (a \cdot b) \cdot c = a \cdot (b \cdot c)$

– Élément neutre : $\exists e \in E \mid a \cdot e = e \cdot a = a$

– Symétrique : $\forall a \in E, \exists b \in E \mid a \cdot b = b \cdot a = e$

### I.2 Groupe abélien

Un groupe abélien, ou un groupe commutatif, est un groupe dont la loi de composition interne est commutative **[12]**. Un ensemble $E$ est un groupe commutatif lorsque

$\forall (a, b) \in E \mid a \cdot b = b \cdot a$

#### I.3.1.3 Groupe cyclique

Un groupe fini $G$ est cyclique si tout élément du groupe peut s'exprimer sous forme d'une puissance ou d'un multiple d'un élément particulier g, appelé le générateur du groupe, c'est-à-dire $G = (g) = \{g^n \mid n \in Z*\}$. Par exemple si G = $\{g^0, g^1, g^2, g^3, g^4, g^5\}$ et $g^6 = g^0$, alors G est un groupe cyclique. Tout groupe cyclique est abélien car $g^n\ g^m = g^{n+m} = g^{m+n} = g^m\ g^n$ .

L'ordre d'un élément e d'un groupe cyclique est le nombre entier n positif le plus petit tel que $ne = 0$ (en notation additive) ou $e^n = 1$ (en notation multiplicative) **[12]**. Reprenons le même groupe G du paragraphe précédent, par exemple l'ordre de l'élément $g^2$ est $3$ car l'élément neutre du groupe est $g^0 = 1$ et $(g^2)^3 = g^6 = 1$.

#### I.3.1.4 Anneau unitaire et commutatif

Un anneau unitaire, ou simplement anneau, est un ensemble $E$ muni de deux lois de composition, notées + (addition) et $\cdot$ (multiplication). $E$ est un anneau, si

– $(E, +)$ est un groupe commutatif

– La loi $\cdot$ est associative et distributive par rapport à la loi +.

– La loi $\cdot$ possède un élément neutre.

Il existe un élément neutre de la loi de composition +, noté 0 tel que $\forall a \in E \mid a+0 = 0+a = a$ et $\forall a \in E, \exists b \in E \mid a + b = 0$. Il existe un autre élément neutre pour la loi de composition ·, noté 1 tel que $\forall a \in E \mid a \cdot 1 = 1 \cdot a = a$.

Un anneau commutatif est un anneau dont la loi de composition · est commutative, c'est- à-dire $\forall (a, b) \in E \mid a \cdot b = b \cdot a$.

### I.3.1.5 Corps

Un corps est un ensemble E muni de deux lois de composition, notée respectivement + et ·. Il faut que les deux lois satisfassent les conditions suivantes :

– Le couple $(E, +)$ forme un groupe abélien, il existe un élément neutre, noté 0, tel que $\forall a \in E \mid a + 0 = 0 + a = a$.

– Le couple $(E\setminus\{0\}, \cdot)$ forme aussi un groupe abélien dont l'élément neutre est 1, $\forall a \in E \mid a \cdot 1 = 1 \cdot a = a$.

– La multiplication · est distributive pour l'addition, c'est-à-dire $\forall (a, b, c) \in E \mid a \cdot (b + c) = a \cdot b + a \cdot c$ et $(b + c) \cdot a = b \cdot a + c \cdot a$.

Autrement dit, un corps est un anneau dont les éléments non nuls forment un groupe abélien pour la multiplication **[12]**.

### I.3.1.6 Corps fini

Un corps fini $F$ est un corps dont le nombre d'éléments est fini. Le nombre d'éléments est l'ordre du corps, noté $q$, qui peut être représenté par la puissance d'un nombre premier $q = p^n$, où $p$ est un nombre premier, appelé la caractéristique du corps et $n \in Z^+$. Pour étudier la cryptographie sur les courbes elliptiques, il faut que nous comprenions les deux types de corps premier et binaire. Un corps est un corps premier, noté $F_P$ lorsque l'ordre du corps $q = p^n$ et p est un nombre premier. Le corps est constitué des nombres entiers $\{0, 1, 2, \ldots, p - 1\}$ et ∀a ∈ Z, $a \bmod p$ donne le reste unique $r$ qui est compris entre $[0, p - 1]$. Un corps fini de l'ordre $2^n$ est un corps binaire, noté $F_{2n}$, qui peut être construit en utilisant une représentation polynomiale **[12]**. Les éléments du corps sont des polynômes binaires dont les coefficients $a_i \in \{0, 1\}$ et les degrés sont inférieurs à $n$. C'est-à-dire $F_{2n} = \{a_{n-1}z^{n-1} + a_{n-2}z^{n-2} + \ldots + a_1 z + a_0 : ai \in \{0, 1\}\}$.