LOUIS DUPONT

Implantation FPGA de l'algorithme de chiffrement à courbes elliptiques

Génération de clefs privées représentées directement en format w-NAF

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en génie électrique
pour l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2005

Résumé

Le chiffrement d'une communication sur un canal quelconque pose un problème de taille. Un émetteur doit en effet transmettre au récepteur une information lui permettant de décoder une communication chiffrée. Le canal d'information n'étant souvent pas physiquement sécurisé, cette information préliminaire doit être transmise sans que les interlocuteurs n'aient à se soucier qu'un autre acteur puisse intercepter cette information.

Différents algorithmes ont été développés afin de rendre possible cet échange préliminaire. Parmi les algorithmes communéments utilisés, la cryptographie à courbe elliptique permet de maximiser la sécurité d'une communication avec un minimum d'échange préliminaire d'information.

La cryptographie à courbe elliptique repose sur la multiplication d'un point sur cette courbe par un scalaire. Cette opération est relativement lourde au niveau logiciel. Le développement d'un co-processeur spécialisé pour cette opération devient alors pertinent.

Ce mémoire résume le développement de pareil co-processeur. Ce dernier a été développé sur FPGA en minimisant les ressources logiques utilisées tout en maximisant la fréquence d'horloge opérationnelle. De plus, le nombre d'opérations sur la courbe elliptique a été minimisé en représentant l'entier multipliant le point sur la courbe elliptique sous sa forme numérique w-NAF.

Ce mémoire propose également une façon inédite pour générer aléatoirement un entier sous sa forme w-NAF en minimisant les ressources logiques nécessaires pour pareille opération.

Abstract

The encryption of a communication on a given channel may appear hazardous. An interlocutor must transmit to another one enough information allowing both interlocutors to encrypt or decrypt the communication. Since the communication channel is visible to potentially malicious actors, this preliminary information must be exchanged without worrying about others intercepting it.

Several algorithms were developed making that exchange possible. Among the most commonly used algorithms, elliptic curve cryptography provides the highest strength per bit.

Elliptic curve cryptography is based on the multiplication of a point on this curve by a scalar. This operation is relatively complex when implemented in software. The use of a specialized co-processor becomes an interesting approach to perform this operation.

This thesis describes the development of such a co-processor. It has been developed targeting a FPGA, minimizing the use of logical resources while maximizing the operating frequency. Moreover, the number of operations on the elliptic curve have been minimized by representing the scalar multiplying the point of the elliptic curve in its w-NAF form.

A method randomly generating an integer in its w-NAF representation is also proposed. This method can be implemented in hardware using a minimum of logical resources.

Avant-propos

Je remercie en premier lieu les docteurs Sébastien Roy et Jean-Yves Chouinard pour leur soutien financier de même que leur expertise. Ces travaux ont été menés grâce au programme de bourses de recherche du conseil de recherches en sciences naturelles et en génie du Canada (CRSNG).

Ces travaux ont également été rendus possible grâce à l'équipement fourni par la société canadienne de micro-électronique (SMC) à travers son réseau de recherche sur les systèmes sur puce (SOCRN).

Finalement, j'aimerais remercier M. Minh Quang Nguyen pour son précieux soutien technique ainsi que les docteurs Paul Fortier et Olivier Sentieys (Université de Rennes) pour avoir accepter la tâche d'évaluation de ce mémoire.

Table des matières

Liste des tableaux

Table des figures

Liste des algorithmes

Nomenclature

$\mathcal{O}$	Point à l'infini sur une courbe elliptique. Toutes les droites verticales coupant une droite elliptique convergent vers ce point. $\mathcal{O}$ est l'élément neutre de l'addition
λ	Pente d'une droite coupant une courbe elliptique
$\overline{P}$	Si P est le premier point d'intersection d'une droite verticale avec une courbe elliptique, alors $\overline{P}$ est le second point d'intersection
a	Paramètre d'une courbe elliptque spécifié par le standard défini par le NIST [34]
$A(x)$	Polynôme défini sur $GF(2^N)$
A_i	Coefficient du polynôme $A(x)$ de poids x^i défini sur $GF(2)$
b	Paramètre d'une courbe elliptque spécifié par le standard défini par le NIST [34]
$B(x)$	Polynôme défini sur $GF(2^N)$
G	Élément générateur propre au protocole et donc connu de tous (point sur la courbe elliptique lorsque le protocole Diffie-Hellman est appliqué à un groupe elliptique)
$GF(2^N)$	Extension du corps de Galois 2 de cardinalité 2^N
$K(x)$	Polynôme tel que $R(x)-K(x)P(x) \in GF(2^N)$ pour effectuer l'opération $R(x) \bmod P(x)$.
k_A	Clef privée d'Alice (scalaire), connue uniquement d'Alice
k_B	Clef privée de Bob (scalaire), connue uniquement de Bob
l	Exposant résiduel sur x résultat de l'algorithme AIA (voir section 4.3.1)
N	Nombre de coefficients constituant un polynôme défini sur $GF(2^N)$
P	Point quelconque sur la courbe elliptique
p	Entier premier
P_A	Clef publique d'Alice disponible pour tous sur le canal (point sur la courbe elliptique lorsque le protocole Diffie-Hellman est appliqué à un groupe elliptique)
P_B	Clef publique de Bob disponible pour tous sur le canal (point sur la courbe elliptique lorsque le protocole Diffie-Hellman est appliqué à un groupe elliptique)

$R(x)$ Polynôme défini sur $GF(2^N)$

S_A Clef partagée obtenue par Alice, connue uniquement par Alice et Bob (point sur la courbe elliptique lorsque le protocole Diffie-Hellman est appliqué à un groupe elliptique)

S_B Clef partagée obtenue par Bob, connue uniquement par Alice et Bob (point sur la courbe elliptique lorsque le protocole Diffie-Hellman est appliqué à un groupe elliptique)

W Nombre d'automates cellulaires formant la chaîne d'automates cellulaires pour la génération de bits aléatoires

w Largeur d'une fenêtre que l'on peut faire glisser sur la représentation w-NAF (voir section 3.3.1) d'un entier sans que plus d'un digit ne soit présent à l'intérieur de cette fenêtre en tout temps

ECDH Protocole de Diffie-Hellman à courbe elliptique

Chapitre 1

Introduction

Plusieurs secteurs d'activités, allant des services gouvernementaux aux activités financières, dépendent de la capacité de transmettre de l'information confidentielle sur un canal non sécurisé. Une infrastructure à clef publique efficace devient nécessaire. Deux infrastructures sont offertes :

- Protocole RSA (Rivest, Shamir, Adleman)
- Protocole de Diffie-Hellman à courbe elliptique (ECDH).

Proposé en 1978 par Rivest, Shamir et Adleman [29], le protocole RSA se base sur la difficulté de factoriser un entier de grande taille. Whitfield Diffie et Martin Hellman [6] ont quant à eux proposé en 1976 un protocole d'échange de clef sur un canal sécurisé sur lequel se base l'algorithme ECDH. Le protocole Diffie-Hellman se base ainsi sur la difficulté de résoudre le problème du logarithme discret dans le groupe multiplicatif d'un entier modulo un nombre premier p.

Le groupe elliptique a été proposé simultanément par Neil Koblitz [16] et Victor Miller [23] en 1985 comme substitut du groupe multiplicatif pour le protocole Diffie-Hellman, donnant naissance au protocole Diffie-Hellman à courbe elliptique (ECDH).

À niveau de sécurité égal, les cryptosystèmes à courbes elliptiques sont basés sur des paramètres beaucoup moins lourds que les cryptosystèmes basés sur le protocole RSA. Il n'existe en effet pas d'algorithme efficace à ce jour pour résoudre le problème du logarithme discret sur le groupe elliptique tel que discuté à la section 2.4.2. L'allègement de la charge de calculs combiné à la diminution de la quantité d'information à envoyer sur le canal rend les cryptosystèmes à courbes elliptiques particulièrement attrayants dans le domaine des communications sans fils.

TAB. 1.1 – Nombre de bits requis pour représenter les clefs à des niveaux de sécurité équivalents pour les protocoles ECDH et RSA

ECDH	**RSA**
163	1024
283	3072
409	7680
571	15360

1.1 Standardisation

Plusieurs organismes régissent l'implantation des protocoles sécurisant les communications en industrie. Parmi eux, on trouve l'*Internet Engineering Task Force*(IETF), l'*American Bankers Association*, l'*International Telecommunication Union*, l'IEEE et le *National Institute of Standards and Technology* (NIST).

Depuis quelques années, ces organismes se sont dotés de standards régissant l'utilisation du groupe elliptique pour des applications en cryptographie. Par exemple, le *American Bankers Association* a développé le standard ANSI X9.63 pour le protocole ECDH. On retrouve également le standard IEEE P1363 ainsi que le FIPS 186-2 du côté du NIST.

Le NIST impose ainsi des paramètres de courbe précis (voir section 2.2.1), un point générateur (section 2.4.2) ainsi qu'un polynôme irréductible définissant l'extension du corps de Galois sur laquelle la courbe elliptique est définie (section 2.3).

1.2 Performances

Le tableau 1.1 compare, pour un niveau de sécurité identique, le nombre de bits requis pour représenter une clef à travers les protocoles RSA et ECDH [27].

En utilisant les résultats donnés par [10] et en comparant les temps de calcul des deux cryptosystèmes mentionnés précédemment, on en arrive à un gain de performance variant de 2.825 à 4.35 selon le processeur visé.

De plus, la diminution de la taille de la clef publique se traduit par une diminution du nombre de bits qu'il est nécessaire d'envoyer sur le canal lors de l'échange de clefs.

1.3 Contributions

Ce mémoire décrit une architecture générique de processeur microprogrammé se conformant au protocole ECDH. Différentes unités d'arithmétique polynomiale sur $GF(2^N)$ sont connectées à un bus arbitré par le microprogramme.

Le rendement du processeur est accéléré en représentant la clef privée sous sa forme w-. La clef privée étant générée aléatoirement, une méthode de génération de nombres aléatoires directement sous forme w-NAF est proposée.

Une architecture matérielle permettant de générer la clef privée à partir d'un générateur de bits aléatoires est également proposée.

1.4 Organisation du mémoire

Le mémoire est séparé en trois sections :

- Introduction à la cryptographie à courbes elliptiques.
- Architecture du processeur.
- Architecture des unités arithmétiques sur l'extension du corps de Galois $GF(2^N)$.

Une approche géométrique ainsi qu'algébrique est donnée pour présenter l'arithmétique sur le groupe elliptique au chapitre 2. L'approche algébrique est précisée lorsque la courbe est représentée sur l'extension du corps de Galois $GF(2^N)$. Finalement, le groupe elliptique est présenté comme substitut au groupe multiplicatif dans le cadre du protocole d'échange de clef Diffie-Hellman.

L'architecture du processeur est présentée au chapitre 3. Les différentes unités connectées au bus principal sont décrites brièvement. L'interface de la mémoire dynamique servant à conserver l'ensemble des points pré-calculés afin d'exploiter la représentation w-NAF de la clef privée est ensuite détaillée. L'architecture de l'unité de séquencement microprogrammée y est également décrite. La méthode proposée pour la génération de nombres aléatoires en représentation w-NAF suivie d'une description de

son implantation matérielle viennent conclure le chapitre.

Trois unités spécialisées pour l'arithmétique sur les corps de Galois sont reliées au bus principal du processeur. Chacune de ces unités est décrite au chapitre 4. L'unité de multiplication polynomiale ainsi qu'une unité de multiplication spécialisée pour des opérandes identiques sont d'abord détaillés. Le chapitre se termine sur l'unité d'inversion polynomiale. L'algorithme auquel se conforme chaque unité est expliqué au début de chaque section. L'architecture matérielle retenue pour chaque unité ainsi que les résultats d'implémentation correspondant suivent la description de l'algorithme.

Chapitre 2

Cryptographie à courbe elliptique

L'algorithme de chiffrement RSA est devenu, au cours des années, une référence en termes de sécurité des communications numériques. C'est un algorithme utilisant deux clefs. Une première clef publique permet de chiffrer le message et une seconde, connue seulement du destinataire, permet de le déchiffrer.

La clef en question est un chiffre qui, après opération mathématique, permet de chiffrer ou de déchiffrer une communication. Lorsque quelqu'un veut transmettre un message, il génère deux clefs. Une première clef publique, permettant de chiffrer le message, sera mise à la disposition de tout le monde. Une seconde, propre à la clef publique mais privée, servira à la déchiffrer. Par le biais d'un artifice mathématique, seule la personne possédant la clef privée sera en mesure de décoder n'importe quelle communication cryptée avec la clef publique.

La cryptographie à courbe elliptique(ECC), basée sur le même principe de clefs, a gagné en popularité récemment. L'étude de courbes elliptiques a permis de définir certaines opérations mathématiques pratiques pour le chiffrement.

2.1 Interprétation géométrique de l'opération d'addition

Une courbe elliptique prend la forme [4] suivante :

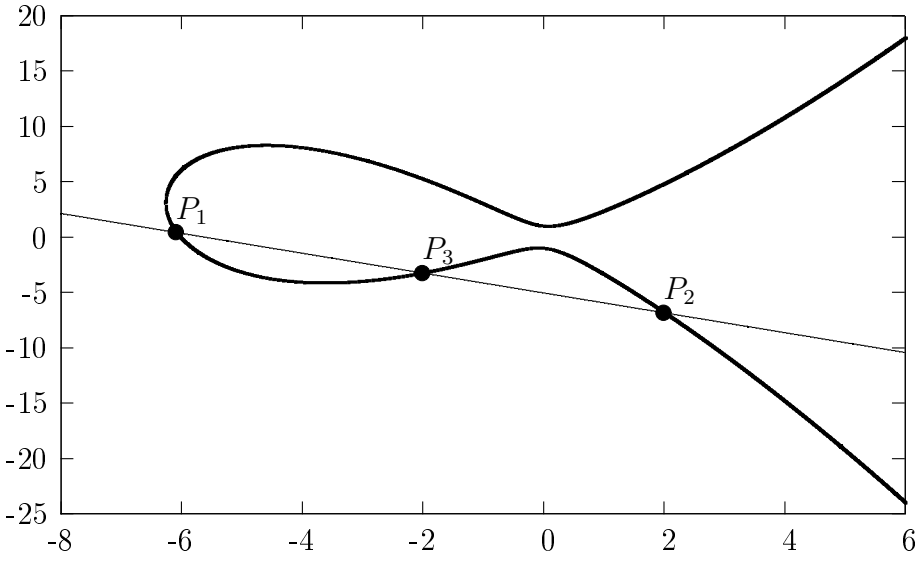

FIG. 2.1 – Droite coupant le courbe elliptique définie par $y^2 + xy = x^3 + 6x^2 + 1$ en trois points

$$y^2 + axy + by \quad = \quad x^3 + cx^2 + dx + e. \tag{2.1}$$

Les points sur une courbe elliptique forment un groupe. Certaines opérations mathématiques classiques ont été définies sur ce groupe en exploitant une propriété particulière de la courbe. Cette propriété veut que toute droite coupant une courbe elliptique en au moins deux points la coupe forcément en un troisième tel qu'illustré à la figure 2.1. Ce résultat est démontré à la section 2.2.1.

Théorème 2.1 *Toute droite coupant une courbe elliptique en au moins deux points, la coupe forcément en un troisième, exception faite des cas traités par les théorèmes 2.2 et 2.3.*

Le théorème 2.1 sera démontré à la section 2.2.1.

Il y a deux exceptions à cette règle :

Théorème 2.2 *Toute droite verticale ne coupe une courbe elliptique qu'en deux points.*

Théorème 2.3 *Toute droite tangente à une courbe elliptique ne coupe cette dernière qu'en deux points.*

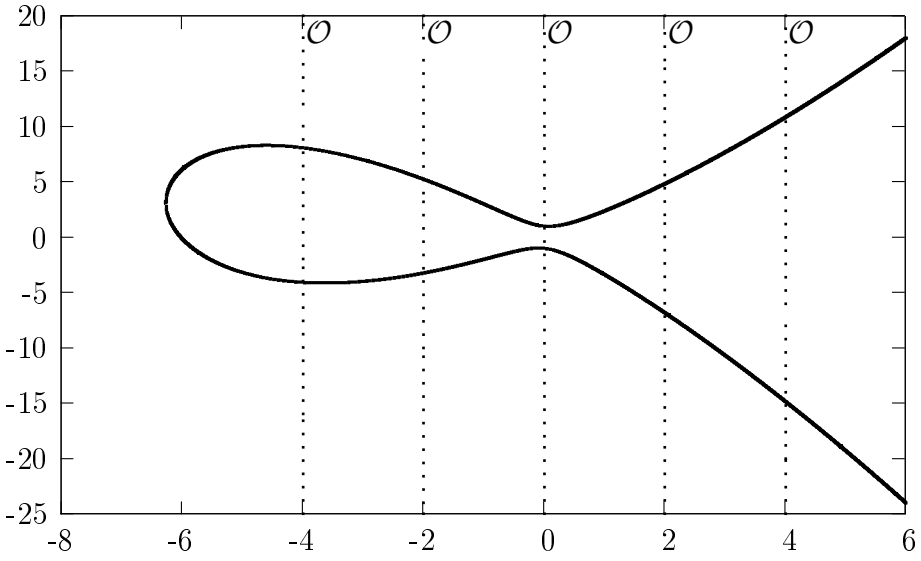

FIG. 2.2 – Illustration de l'emplacement de l'élément neutre noté $\mathcal{O}$

La seconde exception sera traitée plus loin. Afin de contourner la première, on ajoute un dernier point au groupe. Ce point est un élément neutre et noté $\mathcal{O}$. Il est situé à l'infini selon l'axe vertical, où aboutissent toutes les droites parallèles tel qu'illustré à la figure 2.2.

L'emplacement de ce point est évidemment abstrait d'un point de vue géométrique, mais les manipulations arithmétiques s'y rattachant sont très familières. L'addition de deux points quelconques sur la courbe peut être définie à partir de la prémisse suivante :

Soient trois points P_1, P_2 et P_3, les trois points d'intersection d'une droite quelconque avec une courbe elliptique tel qu'illustré à la figure 2.1. Alors :

$$P_1 + P_2 + P_3 = \mathcal{O}.$$

$\mathcal{O}$ étant élément neutre, on peut réécrire l'équation comme :

$$\begin{aligned} P_1 + P_2 &= \mathcal{O} + \overline{P_3}, \\ P_1 + P_2 &= \overline{P_3}. \end{aligned}$$

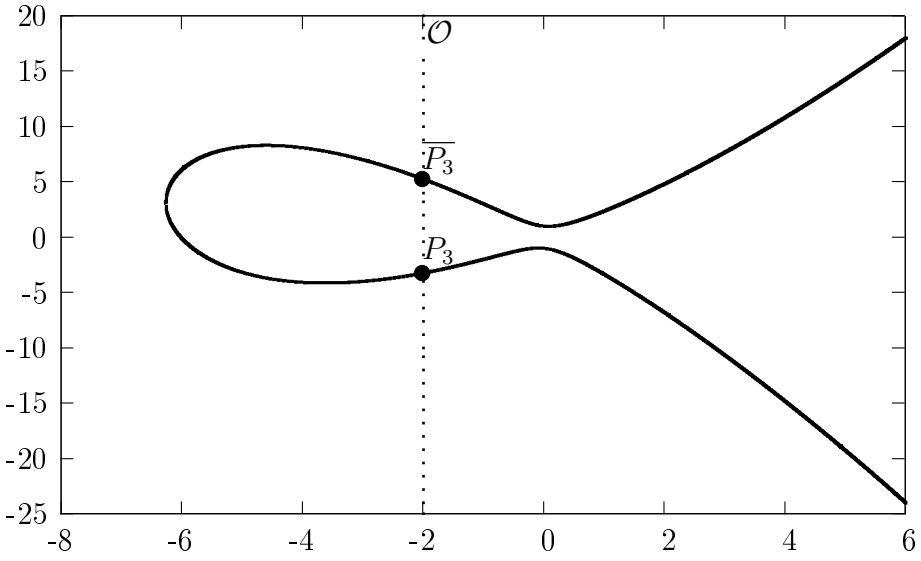

FIG. 2.3 – Emplacement de $\overline{P_3}$ à partir de l'emplacement de P_3

Pour trouver l'emplacement de $\overline{P_3}$, on utilise à nouveau la propriété d'élément neutre de $\mathcal{O}$:

$$\begin{aligned} P_3 + \overline{P_3} &= \mathcal{O}, \\ P_3 + \overline{P_3} + \mathcal{O} &= \mathcal{O}. \end{aligned}$$

P_3, $\overline{P_3}$ et $\mathcal{O}$ sont donc situés sur une même droite selon la prémisse de base. Cette droite étant la droite verticale passant par P_3, $\overline{P_3}$ devient le second point d'intersection avec cette dernière tel qu'illustré à la figure 2.3. Tous les éléments sont maintenant en place pour interpréter géométriquement l'opération d'addition de deux points distincts tel que présentée à la figure 2.4.

Le cas d'une droite tangente à la courbe est illustré à la figure 2.5. On note qu'à mesure qu'une droite converge vers la tangente, deux des points d'intersection entre la droite et la courbe ($P_1^{'}$ et $P_1^{''}$) convergent en un seul point(P_1). On considère donc que la tangente coupe deux fois la courbe en ce point. Ainsi, $P_1^{'}$, $P_1^{''}$ et P_2 sont sur la même droite à la figure 2.5. On peut donc écrire :

$$\begin{aligned} P_1^{'} + P_1^{''} + P_2 &= \mathcal{O}, \\ P_1^{'} + P_1^{''} &= \overline{P_2}. \end{aligned}$$

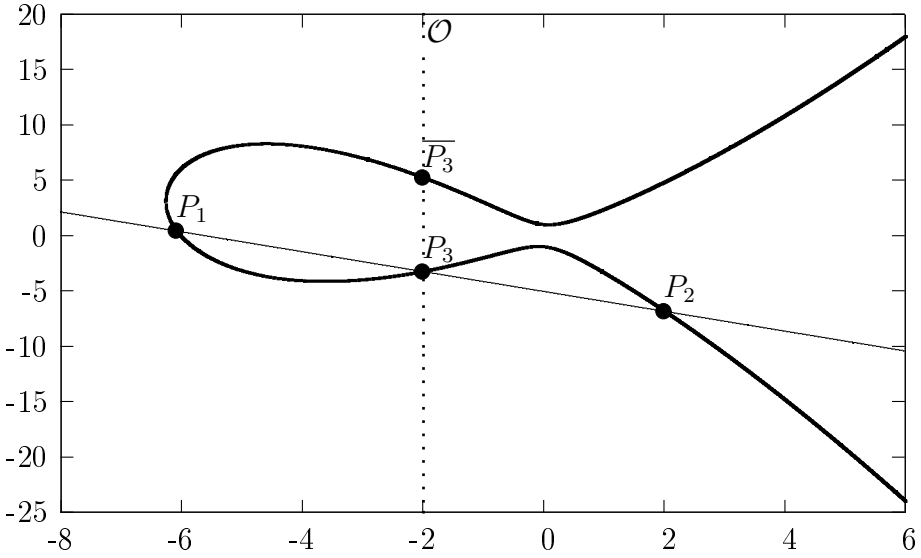

FIG. 2.4 – Interprétation géométrique de l'opération d'addition de deux points distincts sur une courbe elliptique

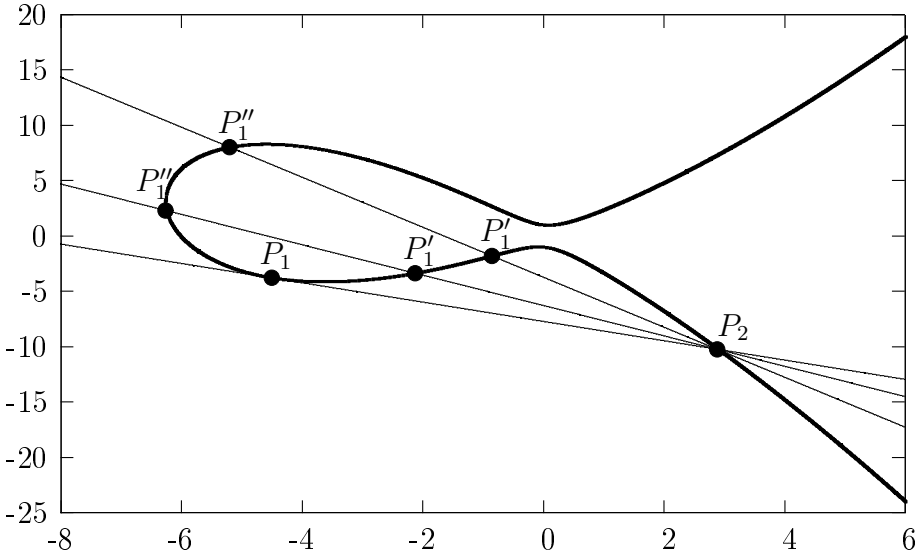

FIG. 2.5 – Traitement d'une droite tangente à la courbe

2.2 Interprétation algébrique de l'arithmétique elliptique

2.2.1 Addition de deux points

Les bases permettant d'interpréter géométriquement l'addition de deux points sur une courbe elliptique sont maintenant posées. On procède ensuite à une analyse algébrique de l'addition. Afin de respecter le standard imposé par le NIST [34], la courbe elliptique donnée en (2.1) sera réduite à la forme donnée en (2.2) ci-bas. Posons d'abord :

$$\begin{aligned} P_1 &= (x_1, y_1)\,, \\ P_2 &= (x_2, y_2)\,, \end{aligned}$$

lesquels constituent deux points sur la courbe, i.e.

$$y^2 + xy = x^3 + ax^2 + b. \tag{2.2}$$

Comme établi, $\mathcal{O}$ est élément neutre de l'addition. On a donc :

$$P_1 + \mathcal{O} = P_1. \tag{2.3}$$

Si P_1 et P_2 sont situés sur une même droite verticale (tel que c'est le cas pour P_3 et $\overline{P_3}$ à la figure 2.3), autrement dit que

$$\begin{aligned} x_1 &= x_2, \\ y_1 &\neq y_2, \end{aligned}$$

alors

$$\begin{aligned} P_1 + P_2 + \mathcal{O} &= \mathcal{O}, \\ P_1 + P_2 &= \mathcal{O}. \end{aligned}$$

Pour trouver la coordonnée en y de $\overline{P_1}$ à partir de P_1, on a :

$$\begin{aligned} y^2 + xy &= x^3 + ax^2 + b, \\ (-1)\, y^2 + (-x)\, y + \left(x^3 + ax^2 + b\right) &= 0, \\ y &= \frac{x \pm \sqrt{\Delta}}{-2} \\ -y - x &= \frac{-x \mp \sqrt{\Delta}}{-2} + \frac{2x}{-2}, \\ -y - x &= \frac{x \mp \sqrt{\Delta}}{-2}, \\ \Rightarrow -\left(x_1, y_1\right) &= \left(x_1, -y_1 - x_1\right). \end{aligned} \tag{2.4}$$

Si la droite n'est pas verticale, deux cas se posent :

- Les deux points à additionner sont distincts.
- Les deux points à additionner sont identiques.

La pente de la droite reliant trois points sur la courbe elliptique est notée λ. Si les points additionnés, P_1 et P_2, sont distincts, on a simplement :

$$\begin{aligned} \lambda &= \frac{\Delta y}{\Delta x}, \\ \lambda &= \frac{y_2 - y_1}{x_2 - x_1}. \end{aligned} \tag{2.5}$$

Si on additionne un point, P_1, avec lui-même, on va plutôt chercher la pente de la tangente à la courbe en P_1. On s'intéresse donc à la dérivée de la courbe. Ainsi, on a :

$$\begin{aligned}
\frac{\delta\left(y^2+xy\right)}{\delta y} &= 2y+x,\\
\frac{\delta\left(x^3+ax^2+b-xy\right)}{\delta x} &= 3x^2+2ax-y,\\
\frac{\delta y}{\delta x} &= \frac{3x^2+2ax-y}{2y+x},\\
\lambda &= \frac{3x_1^2+2ax_1-y_1}{2y_1+x_1}.
\end{aligned} \tag{2.6}$$

Il s'ensuit que la droite reliant les trois points est donnée par l'équation :

$$y = \lambda x + y_0. \tag{2.7}$$

En substituant (2.7) dans (2.2) on obtient :

$$\begin{aligned}
y^2+xy &= x^3+ax^2+b,\\
x^3+ax^2+b-y^2-xy &= 0,\\
x^3+ax^2+b-\left(\lambda x+y_0\right)^2-x\left(\lambda x+y_0\right) &= 0,\\
x^3+ax^2+b-\left(\lambda^2x^2+2\lambda y_0x+y_0^2\right)-\left(\lambda x^2+y_0x\right) &= 0,\\
x^3+\left(a-\lambda^2-\lambda\right)x^2+\left(-y_0-2\lambda y_0\right)x+\left(b-y_0^2\right) &= 0.
\end{aligned} \tag{2.8}$$

Le polynôme de degré trois ainsi obtenu possède trois racines vérifiant l'équation. Si deux de ces racines sont réelles, la troisième est également forcément réelle. On démontre ainsi qu'une droite coupant une courbe elliptique en au moins deux points la coupe en exactement trois points notés $P_1=(x_1,y_1)$, $P_2=(x_2,y_2)$ et $P_3=(x_3,y_3)$, conformément au théorème 2.1. x_1, x_2 et x_3 étant les racines du polynôme, ce dernier peut être réécrit sous la forme :

$$\begin{aligned}
\left(x-x_1\right)\left(x-x_2\right)\left(x-x_3\right) &= 0,\\
x^3-\left(x_1+x_2+x_3\right)x^2+\left(x_1x_2+x_1x_3+x_2x_2\right)x-\left(x_1x_2x_3\right) &= 0.
\end{aligned} \tag{2.9}$$

En faisant correspondre les coefficients devant x^2 dans (2.8) et (2.9), on obtient :

$$\lambda^2 + \lambda - a = x_1 + x_2 + x_3. \tag{2.10}$$

Connaissant P_1 et P_2, il est donc possible de retrouver $\overline{P_3} = (x_3, \overline{y_3})$, le troisième point sur la droite :

$$\begin{aligned} x_3 &= \lambda^2 + \lambda - a - x_1 - x_2, && (2.11)\\ \overline{y_3} &= \lambda x_3 + y_0, \\ y_0 &= y_1 - \lambda x_1, \\ \overline{y_3} &= \lambda x_3 + y_1 - \lambda x_1, \\ \overline{y_3} &= \lambda (x_3 - x_1) + y_1. && (2.12) \end{aligned}$$

La coordonnée en x de $P_3 = P_1 + P_2$ est identique à celle de $\overline{P_3}$. En substituant (2.12) dans (2.4), on trouve pour y_3 :

$$\begin{aligned} y_3 &= -\overline{y_3} - x_3, \\ y_3 &= -(\lambda (x_3 - x_1) + y_1) - x_3, \\ y_3 &= \lambda (x_1 - x_3) - y_1 - x_3. && (2.13) \end{aligned}$$

2.2.2 Multiplication d'un point par un scalaire

Une seconde opération est définie sur le corps. Il s'agit de la multiplication d'un point P sur la courbe par un scalaire k.

L'opération est définie comme l'addition du point P avec lui-même k fois. L'opération est distributive. Ainsi, on a :

$$(k_1 + k_2) P = (k_1 P) + (k_2 P).$$

L'opération est également commutative. On a donc :

$$k_1\left(k_2 P\right) \;=\; k_2\left(k_1 P\right).$$

2.3 Courbe elliptique définie sur un corps de Galois

L'implantation des opérations sur une courbe elliptique requiert de définir cette dernière sur un corps fini, appelé corps de Galois. Pour des raisons pratiques, le corps choisi sera de la forme $GF(2^N)$. Les opérations sur un corps de cette forme se prêtent bien à une implantation matérielle. L'arithmétique polynomiale est employée avec des coefficients appartenant à l'ensemble $\{0, 1\}$.

Sur ce type de corps, l'addition est équivalente à une soustraction. On peut donc réécrire (2.4), (2.5), (2.11) et (2.13) comme :

$$\begin{aligned} -\left(x_1, y_1\right) &= \left(x_1, y_1 + x_1\right), \\ \lambda &= \frac{y_2 + y_1}{x_2 + x_1}, && (2.14) \\ x_3 &= \lambda^2 + \lambda + a + x_1 + x_2, && (2.15) \\ y_3 &= \lambda(x_3 + x_1) + y_1 + x_3. && (2.16) \end{aligned}$$

Une autre propriété intéressante sur ce corps est que l'addition entre deux nombres binaires revient à une opération de ou-exclusif bit à bit entre les deux nombres. L'addition d'un nombre avec lui-même donne donc invariablement 0. Ceci permet de simplifier (2.6) pour obtenir :

$$\begin{aligned} \lambda &= \frac{3x_1^2 + 2ax_1 - y_1}{2y_1 + x_1}, \\ \lambda &= \frac{x_1^2 + (x_1^2 + x_1^2) + (ax_1 + ax_1) - y_1}{(y_1 + y_1) + x_1}, \\ \lambda &= \frac{x_1^2 - y_1}{x_1}, \\ \lambda &= \frac{x_1^2 + y_1}{x_1}, \\ \lambda &= \frac{y_1}{x_1} + x_1. && (2.17) \end{aligned}$$

Lorsque l'on additionne un point avec lui-même, ce qui implique que $x_1 = x_2$, (2.15) peut-être ramené à :

$$x_3 = \lambda^2 + \lambda + a. \tag{2.18}$$

2.3.1 Base polynomiale

Les éléments de l'extension du corps de Galois 2 sont généralement représentés sur l'une de ces deux bases :

1. la base normale optimale,
2. la base polynomiale.

La base normale optimale est surtout utilisée lorsque l'opération de mise au carré intervient fréquemment. L'opération de mise au carré sur une base normale optimale correspond en effet à une rotation vers la gauche de tous les bits composant le corps de Galois.

L'opération de mise au carré intervient particulièrement fréquemment lorsque les points sur la courbe elliptique sont représentés sous forme projective (voir annexe A). L'opération d'inversion n'intervient alors que si un point doit être converti de sa forme projective à sa forme affine. Par contre, les opérations de multiplication interviennent beaucoup plus souvent.

L'utilisation de coordonnées projectives devient ainsi pertinente si l'opération d'inversion sur le corps de Galois est beaucoup plus longue que l'opération de multiplication. On estime que si le ratio de temps nécessaire pour réaliser chacune de ces opérations dépasse 10 [1], le choix des coordonnées projectives devient pertinent.

En choisissant un algorithme récent pour l'opération d'inversion (voir section 4.3), le ratio entre les temps de traitement logiciel des deux opérations devient trop petit pour que le choix des coordonnées projectives devienne pertinent[25]. De plus, l'ajout d'une coordonnée augmente la quantité de ressources logiques devant être dévolues au cryptosystème. Les coordonnées affines ont donc été retenues, et par le fait même, la base polynomiale.

Dans une base polynomiale, chaque bit constituant la quantité représentée par le corps de Galois correspond à un coefficient du polynôme. Par exemple, on a :

$$011010001 \quad \rightarrow \quad x^7 + x^6 + x^4 + 1 \text{ sur } GF(2^9)$$

2.3.2 Multiplication sur une base polynomiale

Un corps de Galois représenté sur une base polynomiale ressemble beaucoup à une quantité binaire traditionnelle. La mécanique de la multiplication est identique à une multiplication entre deux quantités binaires. Seule l'opération d'addition survenant à travers cette mécanique diffère. L'opération d'addition traditionnelle avec propagation de retenue y est remplacée par une opération XOR bit à bit.

Contrairement à la multiplication traditionnelle, la quantité résultante ne peut déborder (la quantité résultante est invariablement représentable sur le nombre de bits correspondants à l'extension choisie du corps de Galois). En effet, une fois la multiplication effectuée, le résultat est réduit par le biais de l'opération modulo. Le second opérande de l'opérateur modulo est un polynôme ne pouvant être factorisé (polynôme irréductible) de degré N.

Un tel polynôme garantit l'existence d'un inverse multiplicatif (voir section 2.3.3) pour chaque quantité. Plusieurs polynômes irréductibles existent pour une extension donnée du corps de Galois 2. Les sections 4.1 et 4.2 détaillent les critères intervenant dans le choix du polynôme irréductible le plus approprié.

2.3.3 Inversion sur une base polynomiale

Une division intervient pour trouver la valeur du paramètre λ dans les équations (2.14) et (2.17). Le concept de fraction n'existe pas sur un corps de Galois. Pour un corps de Galois donné, il existe un nombre fini d'éléments et le résultat de n'importe quelle opération sur ce corps appartient également à ce corps.

La division est ainsi décomposée en une multiplication et une inversion, i.e.

$$\frac{N(x)}{D(x)} = N(x) \times \frac{1}{D(x)}. \tag{2.19}$$

L'opération de multiplication étant définie, il reste à trouver la valeur de $\frac{1}{D(x)}$. Il suffit d'exploiter l'équation suivante :

$$D(x) \times \frac{1}{D(x)} = 1, \tag{2.20}$$

qui indique que $\frac{1}{D(x)}$ est le polynôme par lequel il faut multiplier $D(x)$ pour obtenir 1. L'algorithme pour déterminer ce polynôme est donné à la section 4.3.

2.4 Applications à la cryptographie

L'échange d'information sécuritaire entre deux interlocuteurs sur un canal non sécurisé fait appel à des algorithmes de chiffrement connus (ex : AES [35]). Afin que les deux interlocuteurs puissent utiliser un tel algorithme pour chiffrer ou déchiffrer des données, les deux intervenants doivent avoir accès à une information commune et ignorée des autres.

Cette information peut être échangée au préalable sur un canal sécurisé. Malheureusement, pareil canal n'est pas toujours à la portée des interlocuteurs. Un problème se pose alors. Comment les deux intervenants peuvent-ils utiliser un canal de communications accessible à tous pour transmettre cette information commune sans qu'aucun intervenant externe ne puisse y accéder ?

2.4.1 Protocole d'échange de clés Diffie-Hellman

L'information commune en question est un entier appelé *clef partagée*. Le protocole d'échange de clés de Diffie-Hellman [6] propose une solution à ce problème. Ce protocole repose sur la relative facilité de l'opération d'exponentiation par rapport à son inverse : le logarithme sur un corps fini ou logarithme discret.

Une terminologie propre aux schémas cryptographiques doit être introduite. Supposons que deux interlocuteurs (Alice et Bob) veulent procéder à un échange d'information

sur un canal non sécurisé. Un autre intervenant, Ève, peut tenter malicieusement d'accéder à cette information. Afin de s'entendre sur une clef commune, Alice et Bob vont respecter le protocole d'échange de clef Diffie-Hellman.

- Alice et Bob génèrent chacun une clef privée aléatoire (k_A et k_B).
- Alice et Bob calculent leur clef publique respective (P_A et P_B) en élevant à la puissance k_a et k_b un élément d'un groupe G connu de tous, y compris d'Ève :

$$P_A = G^{k_A}, \tag{2.21}$$
$$P_B = G^{k_B}. \tag{2.22}$$

- Alice et Bob s'envoient leurs clefs publiques respectives. Ève a accès aux deux clefs publiques.
- Alice et Bob élèvent à la puissance k_a et k_b la clef publique reçue.

$$S_A = P_B^{k_A} = G^{k_B k_A}, \tag{2.23}$$
$$S_B = P_A^{k_B} = G^{k_A k_B}. \tag{2.24}$$
$$\Rightarrow S = S_A = S_B \tag{2.25}$$

En respectant le protocole de Diffie-Hellman, Alice et Bob en arrivent à partager une même clef ($S = S_A = S_B$). Afin qu'Ève ne puisse arriver à trouver S, trouver k en ne connaissant que G et G^k doit être un problème très difficile. Ce calcul est désigné *problème du logarithme discret*.

Paradoxalement, calculer G^k en connaissant G et k doit être suffisamment facile pour qu'Alice et Bob puissent calculer leur clef partagée ainsi que leurs clefs publiques respectives.

2.4.2 Intégration de l'arithmétique sur les courbes elliptiques au protocole d'échange de clef Diffie-Hellman

L'arithmétique sur les courbes elliptiques se prête bien à l'implantation du protocole d'échange de clef Diffie-Hellman. L'exponentiation est remplacée par la multiplication d'un point G connu publiquement sur une courbe elliptique, également publiquement connue, par un scalaire généré aléatoirement k.

Le problème du logarithme discret repose sur la difficulté de trouver k connaissant kG et G, si k est très grand. Paradoxalement, connaissant k et G, il est relativement facile de trouver kG. Il suffit en effet d'utiliser la propriété de distributivité de la multiplication. En décomposant k en une somme de puissances de 2 on obtient :

$$\begin{aligned} kP &= \left(\sum_{i=0}^{N-1} k_i 2^i\right) P \text{ où } k_i \in [0,1]\,, \\ kP &= \sum_{i=0}^{N-1} k_i \left(2^i P\right) \text{ où } k_i \in [0,1]\,. \end{aligned}$$

Le calcul de $2^i G$ étant facile (il suffit de doubler G i fois) trouver kG à partir de k et G est de complexité $O(N)$ alors que trouver k à partir de kG et G est de complexité exponentielle en $O(2^N)$. Il faut en effet additionner G à lui même k fois avant de trouver kG. Il existe des algorithmes plus rapides pour extraire k (Pollard rho [8], MOV [20] et Anomalous [31]), mais aucun algorithme connu à ce jour n'arrive à réduire la complexité du problème en dessous de $O(2^{\frac{N}{2}})$.

La clef publique est ainsi représentée par un point formé de deux coordonnées. Ainsi, une paire de coordonnées est envoyée sur le canal. Il est intuitif d'observer que cette approche est quelque peu redondante puisque pour une coordonnée en x donnée, il n'existe que deux coordonnées en y possibles. Il est donc raisonnable de penser qu'il suffirait d'un bit pour représenter la coordonnée en y, réduisant pratiquement de moitié la nombre de bits envoyés sur le canal. Malheureusement, cette approche fait l'objet d'un brevet de la compagnie Certicom *US Patent 6,141,420* [37] et ne sera donc pas couverte dans ce mémoire.

2.5 Conclusion

La cryptographie à courbe elliptique est basée sur l'arithmétique des groupes elliptiques. Les points sur la courbe forment les éléments de ce groupe. À ces points s'ajoute un dernier point, noté $\mathcal{O}$.

L'arithmétique elliptique repose sur l'opération d'addition. Lorsque définies sur l'extension du corps de Galois $GF(2^N)$, les formules pour obtenir les coordonnées du point P_3 résultant de l'addition des points P_1 et P_2 sont données en (2.14), (2.15) et (2.16).

Lorsque les points à additionner sont identiques, (2.14) et (2.15) sont remplacés par (2.17) et (2.18).

La multiplication d'un point P sur la courbe elliptique par un scalaire k consiste à additionner P avec lui-même k fois. La cryptographie à courbe elliptique repose sur la facilité à trouver kP connaissant k et P par opposition à la difficulté de trouver k en connaissant P et kP.

Chapitre 3

Processeur Cryptographique

Le complexité d'un circuit capable de se conformer au protocole d'échange de clef Diffie-Hellman est assez imposante. Le circuit présenté dans ce chapitre prend la forme d'un processeur spécialisé. Un survol des différentes unités reliées au bus principal est d'abord effectué à la section 3.1.

Afin d'alléger la charge de calcul du processeur, la clef privée est représentée sous sa forme w-NAF. La clef étant générée aléatoirement, une méthode de génération de nombres aléatoires en représentation w-NAF est donnée à la section 3.3, suivie de son implantation matérielle à la section 3.3.2.

Afin d'exploiter la représentation w-NAF de la clef privée, un certain nombre de points doivent être pré-calculés. Ces points sont conservés dans une structure centrée sur une mémoire dynamique. L'interface de cette mémoire est présentée à la section 3.4, suivie d'une annexe à cette structure spécifiant si un point est à $\mathcal{O}$.

3.1 Survol du processeur

Le circuit a été décomposé en ses différentes unités fonctionnelles reliées entre elles par un bus trois-états de façon analogue à un processeur complexe traditionnel tel qu'illustré à la figure 3.1. La largeur du bus est fixée par la taille des coordonnées des points sur la courbe elliptique.

Trois des unités sont dédiées à des fonctions spécifiques ayant trait à l'arithmétique

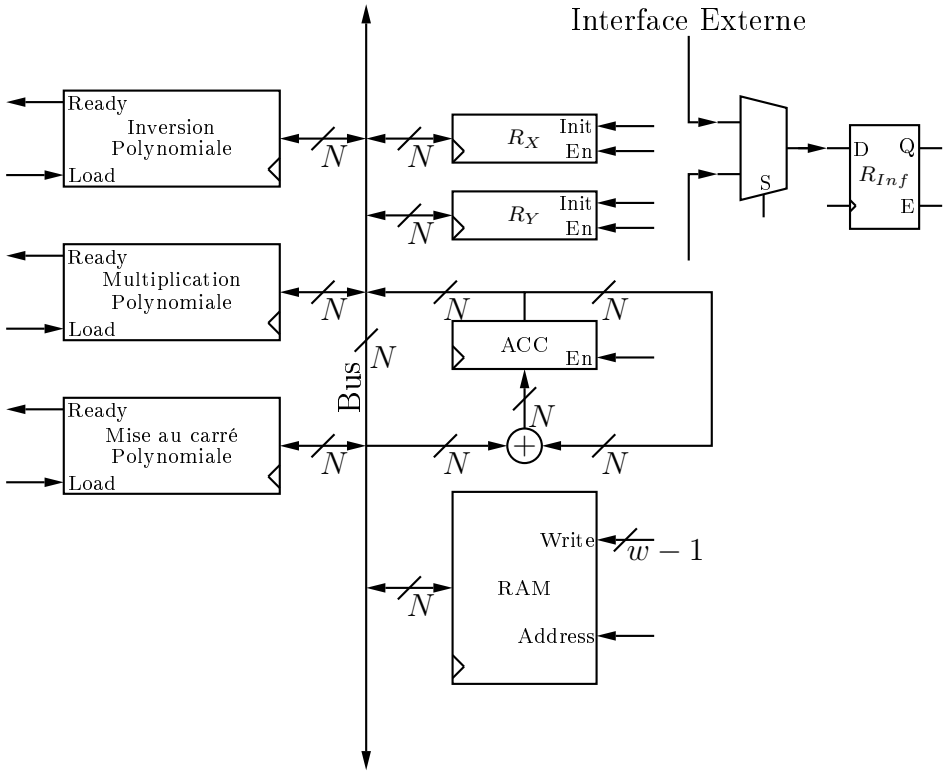

FIG. 3.1 – Schéma du processeur

polynomiale. Leur mécanisme est détaillé au chapitre 4. À ces unités se rajoutent deux registres, R_X et R_Y, chargés de sauvegarder les coordonnées d'un point entre les différentes opérations sur la courbe elliptique. Une bascule sert d'annexe à ces deux registres pour statuer si le point courant est à $\mathcal{O}$. Ces deux registres sont initialisés avec les coordonnées du point G. Un dernier registre, Acc, sert d'accumulateur pour les opérations d'addition sur une extension du corps de Galois 2. L'opération d'addition sur ce corps revient en fait à une opération XOR bit à bit tel que démontré à la section 2.3. Finalement, une mémoire de type dynamique (RAM) sert de table de correspondance pour des points pré-calculés conformément à l'algorithme de multiplication d'un point sur une courbe elliptique par un entier représenté sous forme w-NAF (voir section 3.3.1). L'interface de cette mémoire RAM est décrite à la section 3.4.

L'unité de synchronisation du processeur est décrite à la section 3.2.

3.2 Séquenceur

Les circuits illustrés aux figures 3.1, 3.12 de même que 3.13 font appel à différents signaux de synchronisation que ce soit pour réguler le flot de données sur le bus ou encore pour contrôler les différentes unités.

Ces signaux sont générés par une unité de séquencement. Une architecture de type microprogrammée a été retenue. La complexité du protocole d'échange de clef Diffie-Hellman ainsi que le caractère souvent répétitif se rattachant à sa mécanique privilégie ce choix par rapport à une approche plus traditionnelle faisant appel à une machine à états.

Cette mécanique de synchronisation dépend de plusieurs signaux venant du processeur. Ainsi, le flot du microprogramme doit, par exemple, prendre un chemin différent si les points à additionner sont identiques ou si un des points est à $\mathcal{O}$. L'unité de synchronisation doit également être en mesure d'attendre que certaines unités aient terminé leur travail en plus de réagir aux signaux d'interface de la puce. Le microprogramme doit donc pouvoir faire des branchements conditionnels.

Des appels de sous-routines imbriquées doivent également être possibles à même le microprogramme. L'architecture retenue pour l'unité de séquencement a été proposée par B. W. Bomar [5]. En plus de permettre au microprogramme de faire des branchements conditionnels de même que des appels à des sous-routines imbriquées, l'architecture vise spécialement une implantation FPGA.

Deux architectures sont proposées. Une première version non-pipelinée fait appel à une micromémoire dont le bus d'adresse n'est pas pipeliné. La micromémoire n'est par conséquent pas isolée du reste du circuit par un point de synchronisation.

La version pipelinée de l'architecture isole entièrement la micromémoire du reste du circuit, diminuant les risques d'affecter le chemin critique. En contrepartie, tout branchement conditionnel doit être effectué une micro-instruction à l'avance. D'autres restrictions s'appliquent. Certaines microinstructions doivent être ajoutées afin d'introduire une latence volontaire lorsqu'il n'est pas possible d'effectuer un branchement conditionnel une microinstruction à l'avance.

La version non-pipelinée du séquenceur n'a pas pu être utilisée dans l'implantation visant le FPGA Altera Statix EP1S80B956C6. Toutes les mémoires statiques (ROM) disponibles sur ce FPGA imposent un point de synchronisation au niveau du bus

d'adresses. Les deux approches ont été développées, la version non-pipelinée étant possiblement plus appropriée lorsque le FPGA visé le permet puisqu'elle n'introduit aucune microinstruction excédentaire.

3.2.1 Séquenceur non-pipeliné

L'architecture de la version non-pipelinée du séquenceur est illustrée à la figure 3.2. Elle repose sur une micromémoire asynchrone de type statique (ROM), uniquement accessible en lecture, contenant l'ensemble des microinstructions. En sortie de cette micromémoire se trouve un registre à partir duquel partent tous les signaux de synchronisation du processeur, ainsi qu'un certain nombre de signaux contrôlant le flot du microprogramme.

Plusieurs signaux en provenance du processeur peuvent influencer le flot du microprogramme. La fonction de chacun de ces signaux est détaillée à la section 3.2.3. Ces signaux servent d'entrée à un multiplexeur (M_1) dont la sortie est connectée au module de sélection de la prochaine adresse. Le fonctionnement de ce dernier est décrit à la section 3.2.1. Chaque microinstruction peut ainsi choisir un de ces signaux comme condition de branchement.

La première entrée du multiplexeur M_1 est particulière. Le séquenceur est en effet pourvu d'une structure permettant au microprogramme d'effectuer des boucles. Un compteur à rebours tient le compte du nombre de boucles à effectuer. À chaque itération, le compteur est décrémenté.

La condition de sortie de la boucle est vue comme une condition de branchement. Si la valeur du compte courant est devenue négative, donc que le bit le plus significatif du compteur passe à 1, le séquenceur détecte qu'il est temps de sortir de la boucle. Le bit le plus significatif du compteur est ainsi branché sur la première entrée du multiplexeur M_1, permettant au microprogramme d'effectuer un branchement en fonction de la valeur du compteur.

Le séquenceur permet également d'effectuer des boucles imbriquées. Pour y arriver, les valeurs des boucles externes doivent être sauvegardées dans une pile. Les détails du fonctionnement de la pile sont donnés à la section 3.2.1.

Une autre pile sert à sauvegarder l'adresse de retour lors d'un appel de sous-routine. Elle est connectée sur le microPC, registre pointant toujours sur l'instruction suivante en micromémoire. C'est cette adresse qui sera empilée lors de l'appel de sous-routine

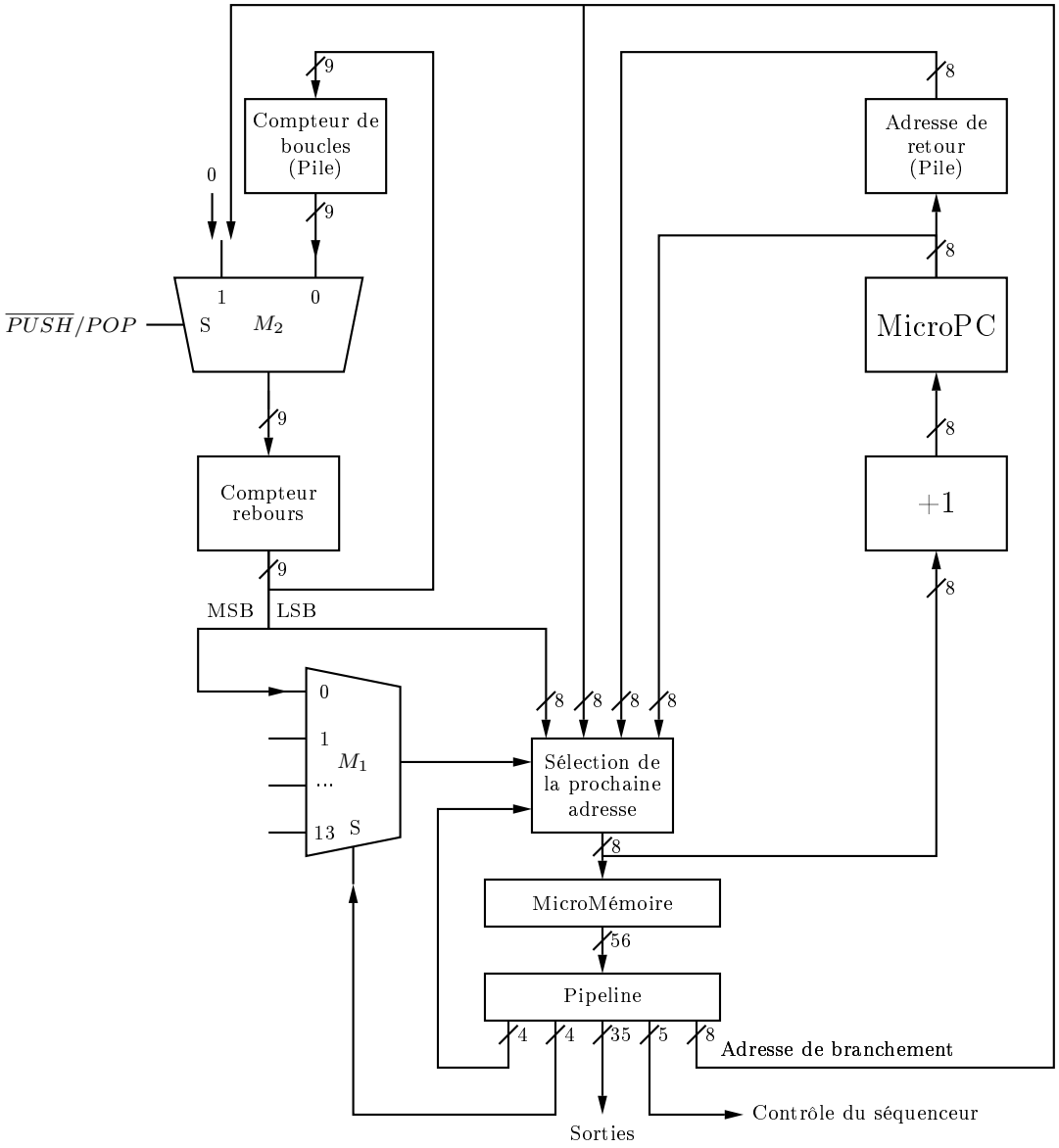

FIG. 3.2 – Unité de séquencement non pipelinée

pour être récupérée sur le dessus de la pile à la fin de l'exécution de la sous-routine.

Quatre options peuvent ainsi être sélectionnées en ce qui concerne l'adresse de la prochaine microinstruction à exécuter :

- Le microPC peut être sélectionné. Il pointe constamment sur l'instruction suivant séquentiellement l'instruction courante en micromémoire.
- L'adresse sur le dessus de la pile peut être sélectionnée lors d'un retour de sous-routine.
- L'adresse peut être spécifiée directement à partir du micromot.
- L'adresse peut être spécifiée à partir du compteur de boucles permettant de faire des branchements à deux voies.

Module de sélection de la prochaine adresse

Le microprogramme doit être capable de spécifier quelle adresse choisir parmi les quatre options énumérées à la section précédente. De plus, le microprogramme doit être capable de spécifier deux options différentes en fonction de la sortie du multiplexeur M_1 (voir figure 3.2).

Le module illustré à la figure 3.3 remplit précisément cette fonction. Deux multiplexeurs, M_1 et M_2, travaillent en parallèle pour obtenir l'adresse de la prochaine microinstruction à exécuter. M_1 calcule la prochaine adresse si la sortie du multiplexeur des conditions est à 1 alors que M_2 en fait autant si la sortie est à 0. Les bits de sélection de ces deux multiplexeurs sont spécifiés à même le micromot.

Pendant que les signaux se propagent à travers ces deux multiplexeurs, la sortie du multiplexeur des conditions a le temps de se stabiliser. L'unité travaille ainsi de façon analogue à un additionneur à retenue conditionnelle, calculant les prochaines adresses associées aux deux possibilités de sortie du multiplexeur des conditions, puis choisissant une des deux options une fois la sortie stabilisée.

Le multiplexeur M_3 est connecté au signal de remise à zéro (Rst) du cryptosystème, forçant le microprogramme à débuter à l'adresse 0. Une autre possibilité aurait été de s'assurer que le micromot dans le registre à la sortie de la micromémoire choisisse invariablement le MicroPC après une remise à zéro du système, en fixant le champs de sélection de la prochaine adresse à une valeur précise. Malheureusement, la configuration du bloc de mémoire ROM fourni par la compagnie Altera ne permet pas de définir la valeur initiale que prendra le registre en sortie du bloc mémoire.

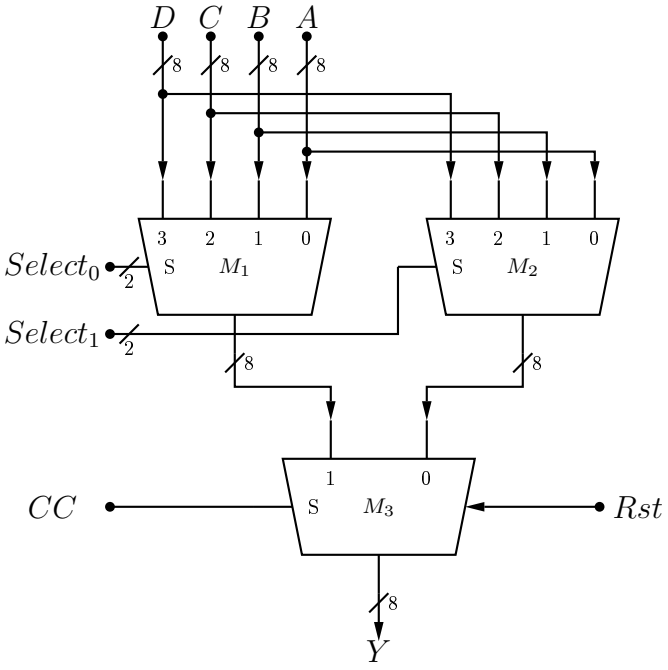

FIG. 3.3 – Module de sélection de la prochaine adresse

Implantation de la pile

Deux structures de type LIFO sont nécessaires à l'implantation de boucles ainsi que de sous-routines imbriquées. À cette fin, la pile illustrée à la figure 3.4 a été développée.

La dernière donnée empilée est disponible directement sur le dessus de la pile. Deux signaux de contrôle, En et $\overline{Push}/Pop$, permettent d'empiler ou de désempiler une donnée.

Un des avantages de cette structure est qu'elle permet de définir une profondeur de pile arbitraire sans affecter le chemin critique, contrairement à une architecture faisant appel à un pointeur de pile par exemple.

3.2.2 Définition du micro-mot

En plus des différents signaux s'assurant que le processeur se conforme au protocole d'échange de clef Diffie-Hellman, différents champs du micromot sont réservés au contrôle du séquenceur.

La plupart des bits sont réservés au contrôle du flot du séquenceur. Quatre bits sont

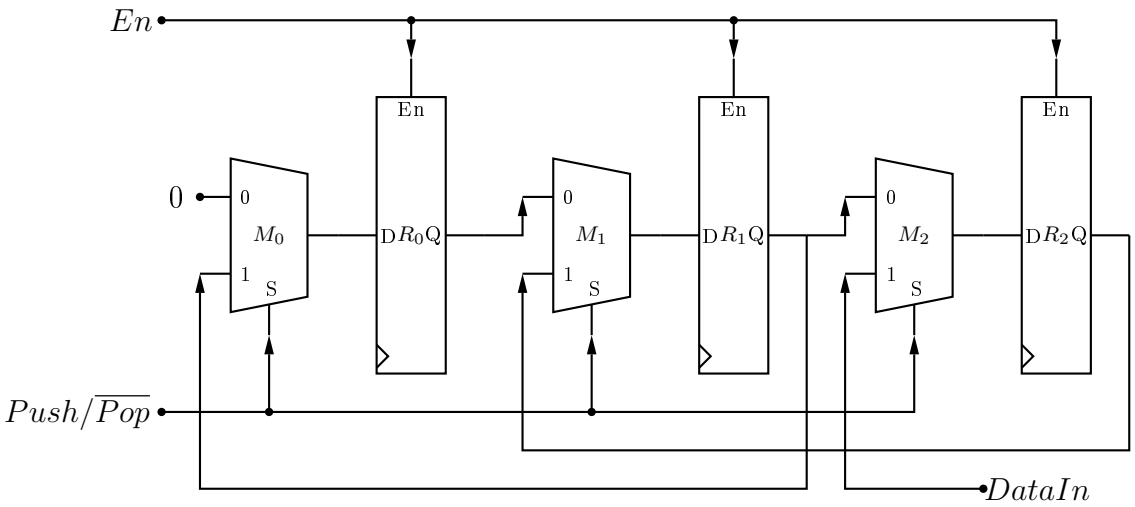

FIG. 3.4 – Architecture de la pile

réservés pour le contrôle du module de sélection de la prochaine adresse et cinq servent au contrôle du compteur et des piles que l'on retrouve à travers le séquenceur.

Un champ de quatre bits est également réservé pour sélectionner la condition de branchement. Ce champ doit également prendre une valeur particulière lors des intructions de boucles.

Flot du microprogramme

Neuf signaux provenant du micromot sont réservés exclusivement au flot du microprogramme. En assignant une valeur précise à chacun de ces signaux, une microinstruction peut choisir la prochaine adresse parmi onze options, chacune d'elles étant détaillée au tableau 3.1.

Chaque option au niveau du flot du microprogramme est définie à partir d'une suite de signaux contrôlant le séquenceur. Le tableau 3.2 définit chacune des étiquettes données par le tableau 3.1 en fonction de la valeur à assigner à chacun des signaux de contrôle.

3.2.3 Instructions conditionnelles

En plus de servir à spécifier le flot du microprogramme, le champ *Condition Select* du micromot sert à spécifier une condition de branchement. Treize conditions de bran-

TAB. 3.1 – Description des différents flots au niveau du microprogramme offerts par le séquenceur

Étiquette	**Description**
Cont	Continue à la prochaine microinstruction
LdCt	Charge le compteur de boucle avec la valeur définie par le champs *Adresse de branchement* du micromot et continue à la prochaine microinstruction. *Adresse de branchement* doit être fixé au nombre de boucles à effectuer soustrait de deux.
PshLdCt	Sauvegarde la valeur du compteur de boucle sur la pile, charge le compteur avec le champs *Adresse de branchement* du micromot et continue à la prochaine microinstruction
PopCt	Charge le compteur de boucle avec la valeur sur la pile, pop la pile et continue à la prochaine microinstruction
Loop	Si la valeur du compteur n'est pas négative, prends le branchement. Sinon, continue à la prochaine microinstruction. Dans tous les cas, décrémente le compteur
Br	Branchement inconditionnel à *Adresse de branchement*
CBF	Branchement conditionnel à *Adresse de branchement* si la condition de branchement est 0, sinon continue à la prochaine microinstruction.
CBT	Branchement conditionnel à *Adresse de branchement* si la condition de branchement est 1, sinon continue à la prochaine microinstruction.
TWB	Branchement à la valeur du compteur de boucle si la condition de branchement est 0, sinon branchement à *Adresse de branchement*
Call	Appel de sous-routine située à *Adresse de branchement*
Ret	Retour de sous-routine

TAB. 3.2 – Description des signaux de contrôle pour obtenir un flot particulier au niveau du microprogramme

Flot	**Next Address Select**	**Loop Stack $\overline{\text{Enable}}$**	**Sub-Routine Stack $\overline{\text{Enable}}$**	**$\overline{\text{Push}}$ Pop**	**Loop Counter Enable**	**Loop Counter Load**	**Condition Select**
Cont	F	1	1	X	0	0	X
LdCt	F	1	1	0	1	1	X
PshLdCt	F	0	1	0	1	1	X
PopCt	F	0	1	1	1	1	X
Loop	D	1	1	X	1	0	0
Br	5	1	1	X	0	0	X
CBF	D	1	1	X	0	0	C
CBT	7	1	1	X	0	0	C
TWB	4	1	1	X	0	0	C
Call	5	1	0	0	0	0	X
Ret	A	1	0	1	0	0	X

chement ont été retenues. Les signaux associés à chacune de ces treize conditions de branchement sont définis au tableau 3.3.

3.2.4 Version pipelinée

Le séquenceur décrit à la figure 3.2 n'isole aucunement la micromémoire du reste du circuit. Il est en effet possible de parcourir, à partir d'une des entrées du multiplexeur des condition (M_1), tout le chemin jusqu'à la sortie de la micromémoire sans croiser le moindre point de synchronisation.

Dans certaines configurations de circuit, ce délai peut s'ajouter au chemin critique, diminuant la fréquence d'opération maximale du circuit. Afin d'éviter le problème, une seconde architecture a été développée dans [5].

L'architecture est présentée à la figure 3.5. Elle ressemble à s'y méprendre au circuit représenté à la figure 3.2 sauf que cette fois-ci, le *MicroPC* est placé juste avant la micromémoire, isolant ainsi cette dernière par deux points de synchronisation.

Une telle architecture présente certains désavantages. Tous les branchements doivent

TAB. 3.3 – Entrées du multiplexeur des conditions

Condition Select	**Condition de Branchement**	
1	InvPolReady	Inversion polynomiale terminée (actif "low").
2	MulPolReady	Multiplication polynomiale terminée (actif "low").
3	SqrtPolReady	Mise au carré polynomiale terminée (actif "low").
4	LUIsInf	Le point sélectionné dans la table de correspondance est $\mathcal{O}$ (actif "low").
5	RIsInf	Le point dans R est $\mathcal{O}$ (actif "low").
6	AccIsZero	La valeur contenue dans l'accumulateur du processeur est 0 (actif "low").
7	DigitIsZero	Le chiffre extrait de la clef privée est de magnitude nulle (actif "low").
8	DigitSign	Le chiffre extrait de la clef privée est négatif (actif "low").
9	DigitIsReady	Un nouveau chiffre à été extrait de la clef privée (actif "low").
10	IsLastDigit	Le dernier chiffre extrait de la clef privée est le chiffre de poids le plus significatif (actif "low").
11	SetX	Signal d'interface externe à la puce permettant d'écrire la coordonnée en X de la clef publique (actif "low").
12	SetY	Signal d'interface externe à la puce permettant d'écrire la coordonnée en Y de la clef publique (actif "low").
13	NewExchange	Signal d'interface externe à la puce permettant d'amorcer un nouvel échange de clef conformément au protocole Diffie-Hellman (actif "low").

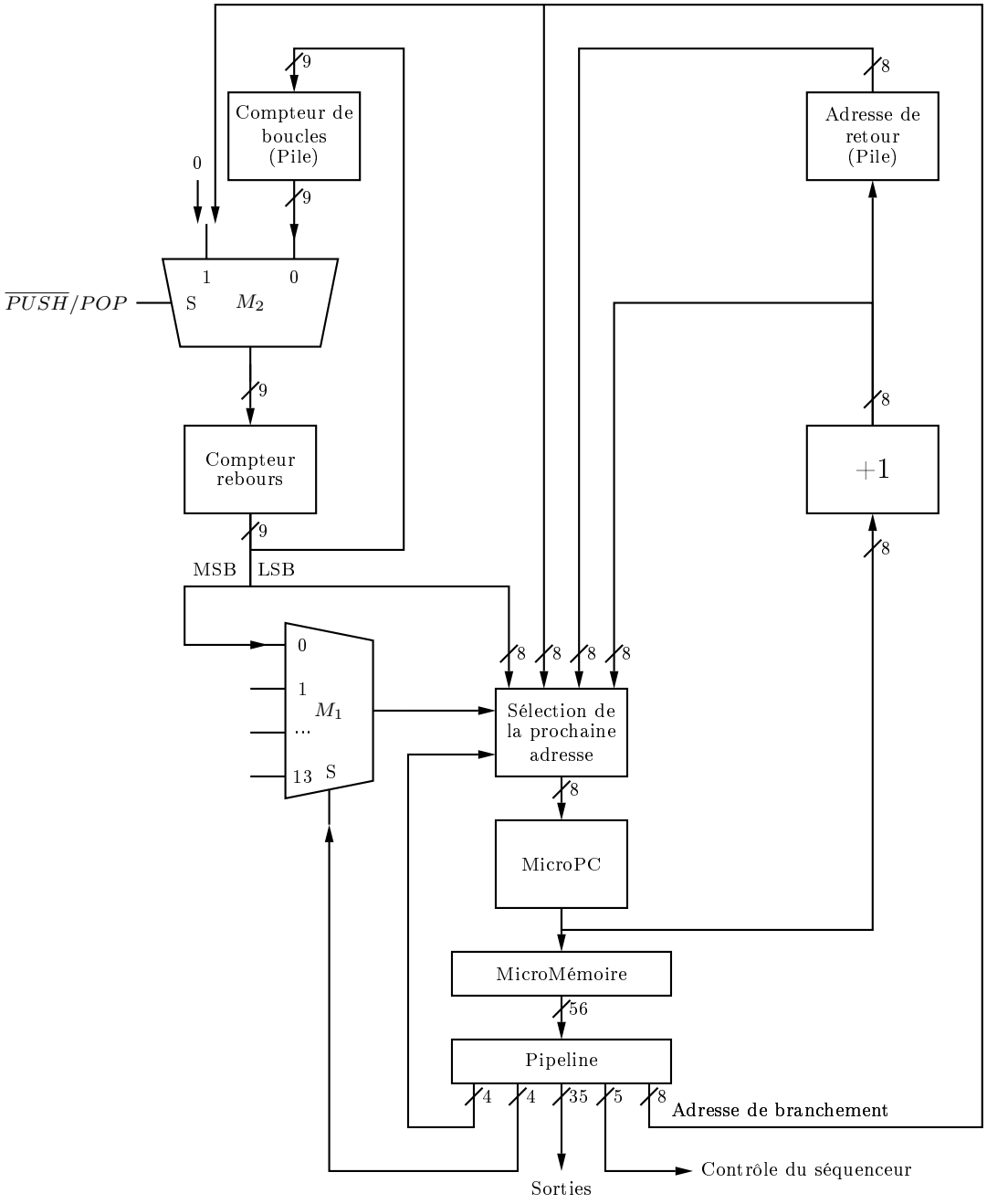

FIG. 3.5 – Unité de séquencement pipelinée

être calculés une microinstruction à l'avance. Autrement dit, le branchement se prendra une microinstruction plus loin. Si un travail peut être effectué durant le cycle d'attente du branchement, le temps de traitement n'est pas nécessairement affecté. Mais si aucun travail ne peut être fait, une microinstruction latente doit être insérée, affectant le temps de traitement.

La version non-pipelinée de l'architecture n'affecte pas nécessairement la fréquence d'opération du processeur décrit à la section 3.1. Des points de synchronisation ont en effet été introduits à chacune des entrées du multiplexeur des conditions, isolant dans une certaine mesure la micromémoire.

Malheureusement, la version non-pipelinée du séquenceur n'a pas pu être retenue puisque aucune mémoire statique (ROM) sans point de synchronisation en entrée n'était disponible sur le FPGA visé. Toutefois, les résultats de synthèse présentés au tableau 5.1 tendent à démontrer que l'architecture non-pipelinée diminue la fréquence d'opération du processeur. Ces même résultats tendent à démontrer que l'architecture pipelinée serait plus appropriée, l'augmentation de la latence du processeur étant plus que compensée par l'augmentation de la fréquence d'opération maximale. Le microcode associé aux deux architectures est fourni à l'annexe B.

3.3 Unité de génération de la clef privée

3.3.1 Représentation w-NAF

Le nombre d'additions associées à la multiplication d'un point P sur une courbe elliptique par un scalaire k dépend du nombre de uns dans la représentation binaire de k. Le scalaire k peut-être représenté sous différentes formes afin de minimiser le nombre de chiffres différents de zéro dans sa représentation. La forme 2-NAF en est un exemple :

$$k = \sum_{i=0}^{N} k_i 2^i \mid k_i \epsilon \{-1, 0, 1\},\ k_i k_{i+1} = 0. \tag{3.1}$$

Il a été démontré que n'importe quel entier peut être représenté sous cette forme, sans que 2 chiffres différents de zéro ne soient adjacents [28]. La représentation d'un entier en respectant cette propriété minimise le nombre de chiffres différents de zéro [28]. Par exemple, 29 peut être représenté par :

$$29 = 11101 = 100\overline{1}01 = 32 - 4 + 1 = 29. \tag{3.2}$$

Il a par ailleurs été démontré à la section 2.3 que la soustraction d'un point sur une courbe elliptique (lorsque $k_i = \overline{1}$) est très facile, rendant l'utilisation de chiffres signés facile d'implantation.

Le principe peut être généralisé à w-NAF, où le paramètre w représente la largeur d'une fenêtre que l'on peut glisser sur la représentation de l'entier, sans que plus d'un chiffre différent de zéro ne soit présent à l'intérieur de la fenêtre en tout temps. L'entier est alors représenté par :

$$\begin{aligned} k &= \sum_{i=0}^{n} k_i 2^i \mid k_i \epsilon \{0, \pm(2j+1) | 0 \leq j \leq 2^{w-2} - 1)\}, \\ \sum_{l=1}^{w-1} k_i k_{i+l} &= 0, \end{aligned} \tag{3.3}$$

c'est-à-dire un chiffre de plus que pour la représentation binaire. Par exemple, pour w=3, 83 peut être représenté par :

$$83 = 1010011 = 100\overline{3}0003 = 128 - 48 + 3 = 83.$$

Le nombre de chiffres différents de zéro diminue à mesure que w augmente, diminuant le nombre d'additions lors de la multiplication d'un point P par un scalaire k. En contrepartie, le nombre de points devant être pré-calculés augmente. Le nombre de points devant être pré-calculés augmente de façon exponentielle avec w alors que le nombre moyen d'additions diminue de façon linéaire. Ainsi, les performances du cryptosystème en viennent à diminuer lorsque w est trop grand.

L'algorithme 3.1 montre comment calculer kP à partir de la représentation w-NAF de k.

Génération d'un nombre aléatoire

Les algorithmes permettant le passage d'une représentation binaire à une représentation w-NAF présentent un obstacle majeur. Pour chaque chiffre extrait, on doit

Algorithme 3.1 Calcul de kP

entrée : k : un entier sur N chiffres représenté sous sa forme w-NAF
P : un point sur la courbe elliptique
sortie : $R = kP$

$R \leftarrow \mathcal{O}$
pour $i = N - 1$ down to 0 **faire**
  $R \leftarrow R + R$
  $R \leftarrow k_i P + R$
fin pour
retourne R

propager une retenue à travers tous les bits de la représentation binaire de b (163 bits en pratique). Un circuit capable d'effectuer la conversion est, par conséquent, relativement lourd. De plus, le premier chiffre utilisable est le dernier à être extrait.

Joye et Tymen [13] ont proposé une méthode permettant de contourner partiellement le problème. Dans les schémas classiques de chiffrement à courbe elliptique, l'entier multipliant un point est généré aléatoirement. Il suffit donc de générer ce nombre directement dans sa représentation w-NAF et ainsi contourner le problème de la conversion. Aucune propagation de retenue n'est nécessaire.

Malheureusement, la méthode proposée se limite à la représentation 2-NAF d'un entier. L'approche présentée ici permet de généraliser la génération de nombre aléatoire à w-NAF.

Généralisation à w-NAF

On applique la transformation suivante à la sortie d'un générateur de bits aléatoires, en commençant par le bit le moins significatif :

$$\left\{ \begin{array}{r} \underbrace{s\,(m_{w-3} \ldots m_1\, m_0)\,1}_{w \text{ bits}} \\ 0 \end{array} \right\} \xrightarrow{T_1} \left\{ \begin{array}{r} \underbrace{(0 \ldots 0\,0)\,K}_{w \text{ chiffres}} \\ 0 \end{array} \right\},$$

où M représente la magnitude de K et s son signe. Si s est 1, alors K est négatif. La magnitude de K est donnée par $2M + 1$.

Cette première transformation est notée T_1. Si le chiffre le plus significatif différent de zéro est négatif, on concatène un autre chiffre prenant la valeur 1 à gauche de la

représentation. Cette seconde transformation est notée T_2. Par exemple :

$$\{\underline{111}\ \underline{101}\ \underline{0}\ \underline{011}\} \xrightarrow{T_1} \{\underline{00\overline{3}}\ \underline{00\overline{1}}\ \underline{0}\ \underline{003}\} \xrightarrow{T_2} \{\underline{1}\underline{00\overline{3}}\ \underline{00\overline{1}}\ \underline{0}\ \underline{003}\}.$$

Il sera démontré que bien que $T = T_1 \bullet T_2$ ne produise par la représentation w-NAF de l'entier en entrée, les valeurs représentées par l'ensemble de sortie constituent une permutation des valeurs représentées par l'ensemble d'entrée.

La transformation proposée, comme tous les algorithmes de conversion connus vers la forme w-NAF, extrait les chiffres un à un, du moins significatif au plus significatif. Cette contrainte représente un handicap pour l'algorithme 3.1 qui utilise les chiffres du plus significatif au moins significatif.

Une autre façon de représenter k, MOF (Mutual Opposite Form), a été proposée par K. Okeya et al. [26]. Comme c'est le cas pour la forme w-NAF, l'algorithme de conversion à la forme MOF exige de propager une retenue à travers la représentation de l'entier, nécessitant un circuit relativement lourd pour se conformer à l'algorithme.

La forme MOF est avantageuse puisque, contrairement à la forme w-NAF, elle permet d'extraire le chiffre le plus significatif en premier. Cependant, dans le contexte d'un processeur cryptographique, l'extraction des chiffres du moins significatif au plus significatif n'est pas problématique. L'opération est effectuée parallèlement à l'algorithme 3.1, ne générant aucun délai d'attente pour connaître le prochain chiffre.

Le premier chiffre est calculé alors que le processeur se charge de remplir une table de correspondance avec des points pré-calculés. Dans certains cas, cette table pourrait être remplie d'entrée de jeu. En effet, le point G étant connu à l'avance, une table de correspondance séparée basée sur mémoire statique pourrait être implantée pour le point G. Dans pareil cas, la forme MOF pourrait se révéler avantageuse.

La table en question devrait être implantée en plus d'une table basée sur une mémoire dynamique pour la multiplication de la clef privée par la clef publique de l'interlocuteur. Pour cette raison, l'architecture du circuit proposé dans ce mémoire n'utilise qu'une seule table de correspondance basée sur une mémoire dynamique. La même table est utilisée à chaque étape du protocole d'échange de clef Diffie-Hellman.

Démontration de la permutation

Une partie de la preuve consiste à démontrer que T induit obligatoirement une représentation w-NAF. Par construction, T_1 induit forcément une représentation w-NAF. Chaque chiffre différent de zéro est en effet suivi de w-1 zéros lors de la transformation. Il est donc impossible que 2 chiffres différents de zéro se retrouvent dans une même fenêtre de w chiffres. La magnitude de K étant représentée sur w-2 chiffres, cette dernière ne peut dépasser

$$2(2^{w-2} - 1) + 1,$$

conformément à (3.3).

Un problème pourrait survenir si le chiffre le plus significatif et différent de zéro est négatif. T_2 intervient alors et on ajoute un 1 à gauche de la représentation. Or, si le chiffre négatif est situé dans les w-1 bits les plus significatifs, au plus w-2 zéros le séparera du 1 ajouté, violant la propriété de non-adjacence.

Cette situation ne peut survenir que si un 1 apparaît en entrée dans les w-2 bits les plus significatifs. Par construction, le chiffre généré dans pareil cas ne peut être négatif. Le bit de signe est en effet situé w-1 bits plus loin, à l'extérieur du nombre représenté en entrée. Le bit de signe est donc forcément zéro et le chiffre généré, positif. Par exemple, dans le pire des cas, le bit le plus significatif en entrée sera 1 et interprété comme bit de signe :

$$\begin{aligned} & \underbrace{1\,(m_{w-3}\,\ldots\,m_1\,m_0)\,1}_{w \text{ bits les plus significatifs}}, \\ \xrightarrow{T_1} \quad & \underbrace{(0\,\ldots\,0\,0)\,\overline{2M+1}}_{w \text{ chiffres les plus significatifs}}, \\ \xrightarrow{T_2} 1 \quad & \underbrace{(0\,\ldots\,0\,0)\,\overline{2M+1}}_{w \text{ chiffres les plus significatifs}}. \end{aligned}$$

La propriété de non-adjacence est bel et bien respectée. D'autre part, la représentation générée tient, au maximum, sur un chiffre de plus que sa représentation binaire, ce qui est encore une fois conforme aux propriétés de la représentation w-NAF.

Muir et Stinson [24] ont démontré qu'un entier possède une et une seule représentation satisfaisant les contraintes imposées par la représentation w-NAF. À partir de cette prémisse et sachant que 2 séquences différentes en entrée ne peuvent générer une même représentation w-NAF, il suffit de démontrer que toute représentation générée

représente bien un entier compris entre 0 et $2^N - 1$ où N est le nombre de bits en entrée. Si c'est le cas, les valeurs représentées par l'ensemble de sortie sont forcément une permutation des valeurs représentées par l'ensemble d'entrée.

Par construction, le chiffre le plus significatif est toujours positif. Pour démontrer que la valeur représentée est toujours positive, il suffit de considérer le pire cas, lorsque le premier chiffre est 1 et que tous les autres chiffres pouvant être différents de zéro prennent la valeur la plus négative possible. Pour que la valeur représentée soit positive, l'inégalité suivante doit être vérifiée :

$$\begin{aligned}
\sum_{i=0}^{k-1}(2(2^{w-2}-1)+1)2^{wi} &\leq 2^{kw}, \\
(2^{w-1}-1)\sum_{i=0}^{k-1}2^{wi} &\leq 2^{kw}, \\
(2^{w-1}-1)\frac{2^{kw}-1}{2^w-1} &\leq 2^{kw}, \\
(2^{w-1}-1)(2^{kw}-1) &\leq (2^w-1)2^{kw}.
\end{aligned}$$

En observant les termes du produit de chaque côté de l'inégalité, on peut constater que cette dernière est forcément vraie. Il reste maintenant à démontrer que la valeur représentée est inférieure à 2^N.

La démonstration est faite que si le chiffre le plus significatif est positif, la valeur représentée est positive. Par symétrie, si le chiffre le plus significatif est négatif, la valeur représentée est forcément négative. T_2 intervient alors et on ajoute alors un 1 de poids 2^N à la représentation. Dans pareil cas, la valeur finale est donc forcément inférieure à 2^N.

Le cas où seule le transformation T_1 intervient est facile à démontrer. Il suffit de descendre à l'échelle d'une fenêtre et d'observer que la valeur générée est forcément inférieure à la valeur maximale que peut prendre la représentation binaire sur cette même fenêtre.

En effet, si le chiffre généré est positif, il prendra exactement la même valeur que la

représentation binaire correspondante :

$$\begin{aligned} s &= 0, \\ M &= (m_{w-3}...m_1\, m_0), \\ 2M &= (m_{w-3}...m_1\, m_0)\, 0, \\ 2M+1 &= (m_{w-3}...m_1\, m_0)\, 1 = s\, (m_{w-3}...m_1\, m_0)\, 1. \end{aligned}$$

La transformation T_1 ne peut ainsi générer aucun débordement par rapport à la valeur binaire en entrée, démontrant que la valeur générée est forcément inférieure à 2^N.

3.3.2 Implantation matérielle

Une implantation matérielle générique de la méthode proposée pour générer un entier aléatoire en représentation w-NAF est donnée ici. Dans le cadre de ce mémoire, les paramètres de ce circuit sont demeurés génériques. Il est à noter toutefois que le circuit peut être légèrement optimisé pour des valeurs spécifiques de w.

Seuls les signaux d'interface essentiels sont décrits dans ce mémoire. En pratique, d'autres signaux devraient être ajoutés. Ainsi, un signal signifiant si oui ou non le chiffre qui vient d'être extrait est le dernier (signifiant que le processus de multiplication est terminé) a été ajouté dans le circuit associé à ce mémoire.

Un autre signal d'interface, signifiant si oui ou non le prochain chiffre a été extrait est prêt à être transféré en sortie, pourrait également être ajouté. Toutefois, un tel signal serait inutile en pratique puisque l'extraction des chiffres est effectuée en parallèle et requiert beaucoup moins de coups d'horloge que les autres opérations associées à la multiplication d'un point sur la courbe elliptique par un scalaire.

Architecture du circuit

L'implantation matérielle proposée repose sur l'utilisation d'un registre à décalage tel qu'illustré à la figure 3.6. Le registre à décalage est initialisé à une valeur particulière, essentielle au bon fonctionnement du circuit pour trois raisons.

Dans un premier temps, l'entrée aléatoire de la transformation T est mise en place dans le registre à décalage. Un multiplexeur s'assure ainsi de sélectionner une entrée

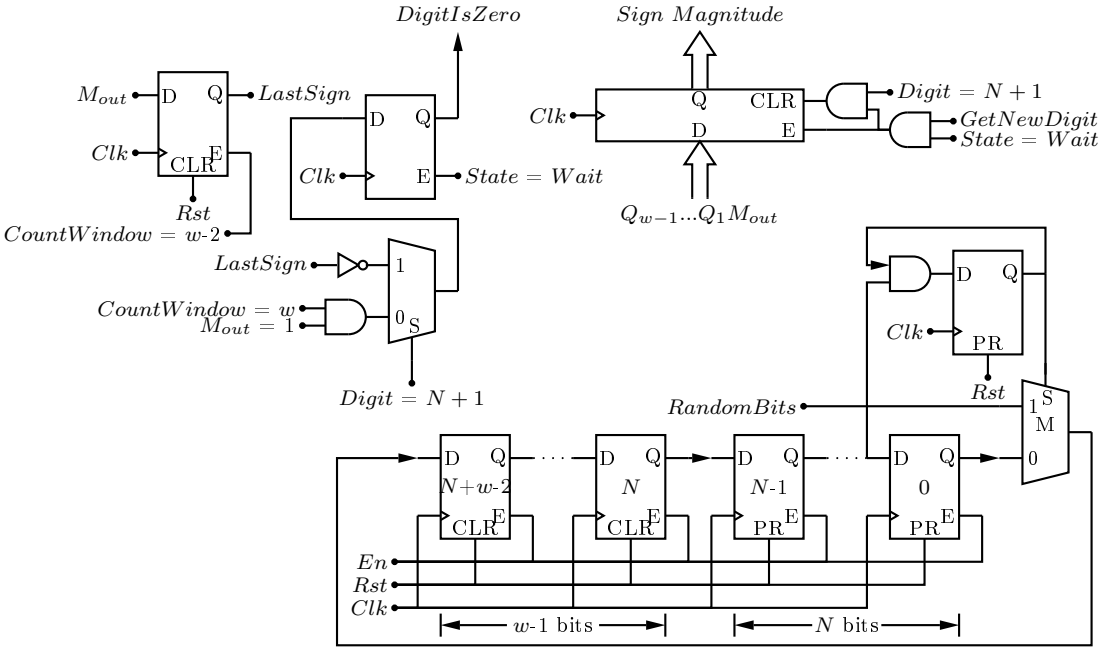

FIG. 3.6 – Implantation matérielle

aléatoire au registre à décalage. Cette mécanique est illustrée à la figure 3.6. Dès que le premier bit initialisé à zéro atteint la fin du registre à décalage, un mécanisme n'ayant recours qu'à une bascule D et une porte ET s'assure que le bit de sélection du multiplexeur M soit forcé à zéro. À partir de ce moment, la sortie du registre à décalage devient son entrée, transformant le circuit en registre à décalage circulaire.

Ce mécanisme fait en sorte que le registre à décalage est, dans un premier temps, connecté à une entrée aléatoire puis devient un registre à décalage circulaire. Les w-1 bits laissés à zéro servent en premier lieu à assurer cette reconfiguration. Ces bits correspondent en fait aux bits débordant la représentation de l'entier en entrée. Cette mécanique permet d'éviter d'avoir recours à un mécanisme potentiellement complexe pour extraire la magnitude, ainsi que le signe, du dernier chiffre différent de zéro généré par T_1, si ce dernier est situé dans les N-w+1 chiffres les plus significatifs. Le chiffre peut ainsi être extrait comme tous les autres en utilisant les bits débordant la représentation de l'entier. Le troisième usage de ces w-1 bits sera détaillé ultérieurement.

La commande du circuit décrit à la figure 3.6 est assuré par une machine à états tel qu'illustrée à la figure 3.7.

Il est à noter que la transformation démarre dès que le premier bit généré aléatoirement est connu. Cette mécanique peut sembler problématique puisqu'elle implique que la magnitude du premier chiffre extrait ne sera pas connue avant que le chiffre en question ne soit extrait.

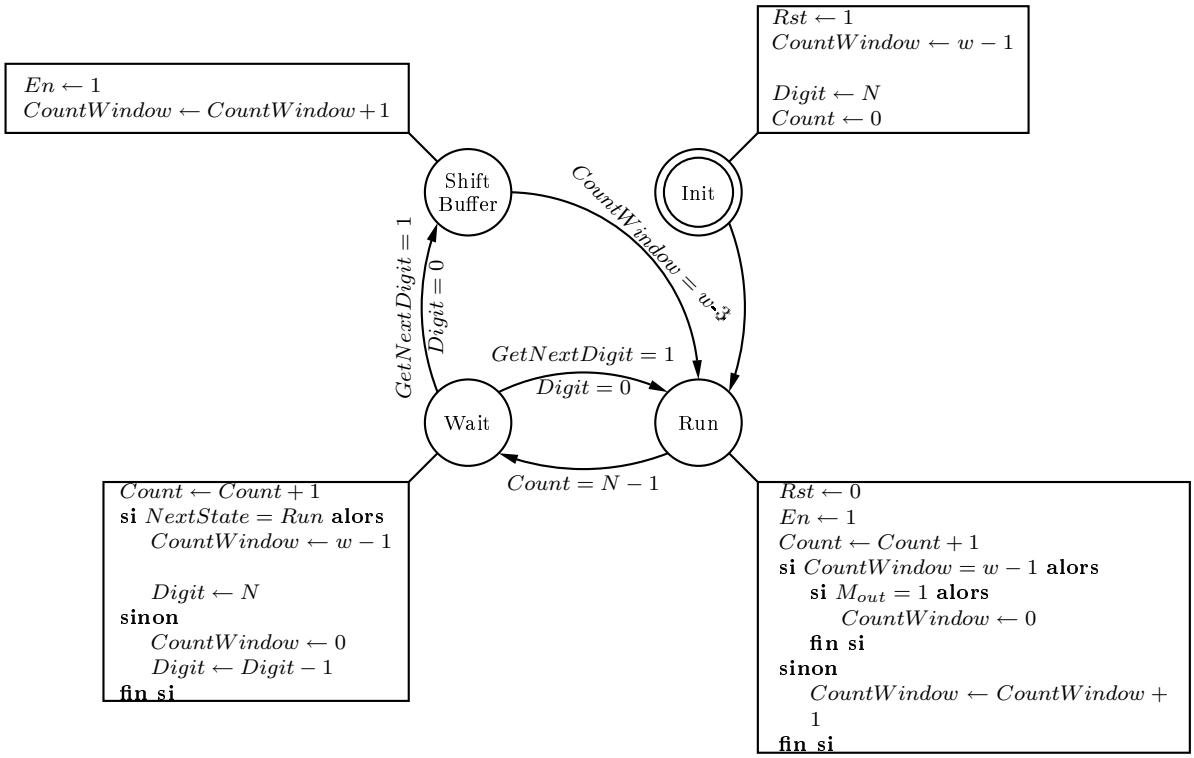

FIG. 3.7 – Machine à états utilisée pour la synchronisation du circuit.

Le premier chiffre extrait, soit le chiffre le plus significatif, est un cas particulier de la transformation T. Seules deux valeurs sont admises pour ce chiffre : zéro et un. T_2 choisit entre ces deux valeurs en fonction du signe du dernier chiffre extrait. La transformation peut ainsi commencer sans que la valeur en entrée ne soit connue en entier.

Une fois que la valeur aléatoire en entrée de T est entièrement connue, le registre à décalage devient un registre à décalage circulaire de façon à préserver cette valeur. Pour chaque chiffre extrait, la transformation T doit connaître tous les chiffres précédents puisque ces derniers sont extraits du plus significatif au moins significatif.

Après extraction d'un chiffre, tous les bits du registre seront décalés circulairement de $N + w - 2$ bits (sur un total $N + w - 1$ bits) de façon à ce que la transformation s'arrête sur le prochain chiffre à extraire. Un nouveau chiffre sera alors extrait avant de répéter exactement la même mécanique.

Ce mécanisme s'assure de toujours extraire le bon chiffre avec un minimum de gestion de la part de la machine à état. Mais en s'assurant de terminer la conversion au bon chiffre, on force la conversion à débuter au mauvais endroit. La conversion ne

commence en effet pas au bit de poids zéro à l'entrée mais au bit suivant le chiffre qui vient d'être extrait. À première vue, le mécanisme déterminant si oui ou non un chiffre est de magnitude nulle pourrait s'en trouver affecté.

Le mécanisme en question repose sur un compteur : $CountWindow$. Ce compteur s'assure que $w-1$ chiffres seront forcés à zéro après l'extraction d'un chiffre de magnitude non nulle. Afin de ne pas affecter la transformation, ce compteur doit revenir à sa valeur initiale au début de la transformation.

Le tampon de w-1 zéros trouve ici sa troisième utilité. Une fois le dernier chiffre extrait, la transformation T passe à travers ces w-1 zéros, incrémentant le compteur $CountWindow$ à chaque zéro jusqu'à concurrence de w-1 (la valeur initiale du compteur). Cette séquence s'assure ainsi que $CountWindow$ atteigne sa valeur initiale lorsque la transformation reprendra au bit le moins significatif.

Résultats d'implémentation

L'unité a été implémentée en choisissant un générateur de bits aléatoires basé sur une suite d'automates cellulaires (voir section 3.3.3). Le design a été implémenté pour différentes tailles de clefs privées correspondant aux extensions de corps de Galois recommandée par le NIST [34] en visant le FPGA Altera Statix EP1S80B956C6.

Le largeur de fenêtre w a été fixée de façon à minimiser le nombre moyen d'additions de points donné par :

$$2^{w-2} - 1 + \frac{N+1}{w+1}, \tag{3.4}$$

où N est le nombre de bit nécessaires pour représenter la valeur en entrée de T. Cette valeur n'est qu'approximativement proportionnelle au temps de calcul puisqu'elle ne tient pas compte du nombre d'additions d'un point avec lui même. Ne connaissant pas, à priori le ratio de temps de traitement entre les opérations d'addition et de doublage, l'équation (3.4) constitue une bonne approximation.

Les résultats obtenus sont présentés au tableau 3.4.

Le chemin critique du circuit est lié à la mécanique de gestion de la part de la machine à états. La fréquence d'opération maximale demeure ainsi élevée, indépendamment de la taille de la clef privée. L'unité ne risque pas de ralentir le processeur, le fréquence

TAB. 3.4 – Fréquence d'opération maximum et ressources logiques utilisées pour l'unité de génération de clef privée pour différentes tailles de clefs basées sur les extensions de $GF(2^N)$ recommandées par le NIST.

w	N	**Fréquence d'opération maximale**	**Éléments logiques**
5	163	299.22	286
5	233	248.08	531
5	283	276.32	632
6	409	272.78	884
6	571	273.90	1213

d'opération maximale du module d'inversion polynomiale étant inférieure d'un facteur allant de 2.2 à 3.4 (voir tableau 4.4).

3.3.3 Génération de bits aléatoires

Le circuit décrit à la figure 3.6 fait appel à un signal générant un bit aléatoire à chaque coup d'horloge. Plusieurs architectures matérielles ont été proposées dans la littérature que ce soit pour générer des nombres aléatoires [17] ou pseudo-aléatoires [2].

Le circuit décrit en [17] n'est pas en mesure de générer un bit aléatoire à la fréquence voulue. Un générateur pseudo-aléatoire a par conséquent été retenu. Les circuits basés sur des registres à décalage à contre-réaction linéaire [15] sont souvent retenus pour les implantations VLSI en raison de leurs faibles complexités. Malheureusement, les propriétés statistiques d'un tel circuit le rendent inutilisable en cryptographie [21].

Le générateur choisi se base plutôt sur une automate cellulaire implémentant la règle 30 [38]. Un automate cellulaire est souvent comparé à un insecte dans un essaim. Chaque insecte obéit à des règles très simples mais en interagissant avec les autres insectes autour de lui, il émerge de l'essaim un comportement très complexe.

En pratique, une cellule d'un automate cellulaire prend une valeur binaire. Cette valeur dépend de sa propre valeur à l'itération précédente ainsi que de celle de ses deux voisins. La figure 3.8 donne, pour les huit configurations possibles, la valeur que doit prendre une cellule d'un automate cellulaire implémentant la règle 30 à l'itération

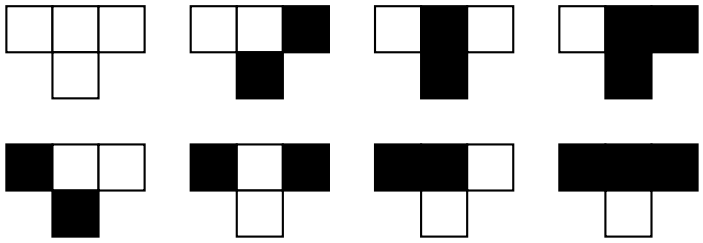

FIG. 3.8 – Règles 30 des automates cellulaires

suivante.

Le comportement d'une seule cellule est par conséquent très simple, mais lorsqu'on la fait interagir avec ses voisins, le comportement global du système devient presque aléatoire tel qu'illustré à la figure 3.9.

Les cellules à la figure 3.9 forment un anneau, le premier interagissant avec le dernier. En échantillonnant la valeur d'un seul des cellules. On obtient un bit pseudo-aléatoire à chaque coup d'horloge.

La valeur initiale du vecteur de cellules est importante. Par exemple, si toutes les cellules étaient initialisés à zéro, le système resterait dans le même état, itération après itération.

Comme tout générateur de nombres pseudo-aléatoires, la séquence de bits aléatoires est périodique. Des simulations ont permis de vérifier que la période est de

$$2^{W-1},$$

où W est le nombre de cellules constituant la chaîne. Le paramètre W a été choisi de façon à pouvoir échantillonner 2^N valeurs de N bits avant de revenir au début de la séquence. L'équation (3.5) démontre que la valeur visée pour le paramètre W est de $2^N + \log_2(N) + 1$.

$$\begin{aligned} 2^{W-1} &= N2^N, \\ 2^{W-1} &= 2^{\log_2(N)}2^N, \\ 2^{W-1} &= 2^N + \log_2(N), \\ W &= 2^N + \log_2(N) + 1. \end{aligned} \tag{3.5}$$

Le générateur ainsi obtenu a été soumis avec succès au *Statistical Test Suite for the*

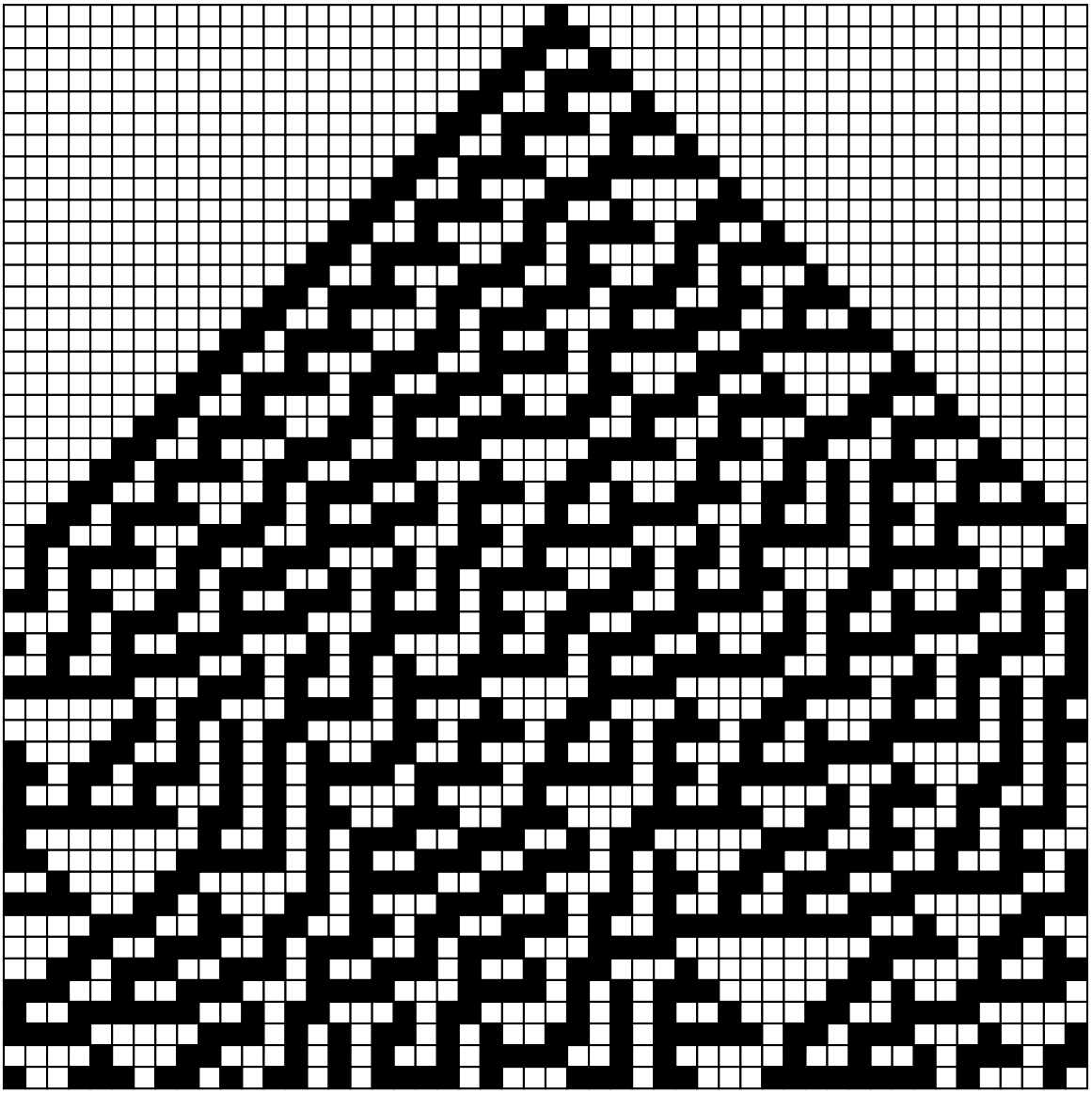

FIG. 3.9 – Comportement aléatoire d'un automate cellulaire

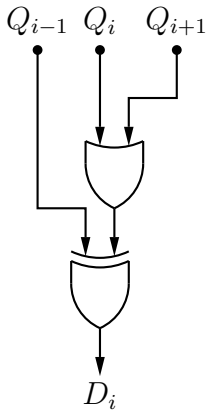

FIG. 3.10 – Cellule d'un automate cellulaire à base de portes logiques

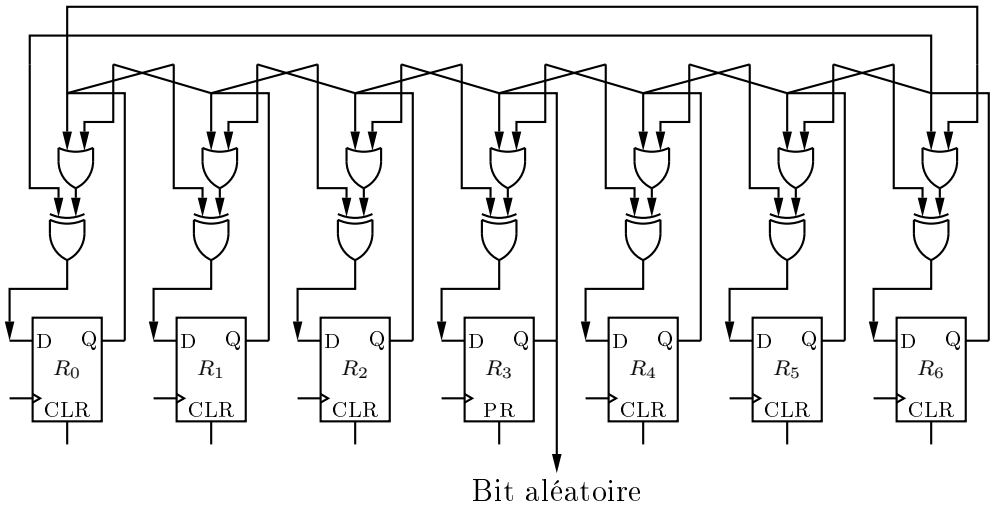

FIG. 3.11 – Générateur de bits aléatoires

Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications du *NIST* [36].

Implantation matérielle

Un automate cellulaire se révèle particulièrement facile à implanter au niveau matériel. Le circuit illustré à la figure 3.10 représente une seule cellule. Il peut être obtenu aisément en dérivant les règles énoncées à la figure 3.8 à l'aide d'une table de Karnaugh.

Chaque cellule peut être connecté avec ses voisins immédiats, formant le circuit de la figure 3.11. Le circuit fonctionne à une fréquence d'opération très élevée puisque l'on ne trouve, au maximum, que deux portes logiques entre deux points de synchronisation.

3.4 Table de correspondance

Un certain nombre de points doivent être pré-calculés afin d'accélérer la multiplication d'un point sur une courbe elliptique tel que démontré par l'algorithme décrit à la section 3.3.1. Ces points sont fonction du point à multiplier sur la courbe elliptique. Deux informations doivent être sauvegardées pour chaque point :

- les coordonnées en x et y du point,
- un bit supplémentaire statuant si le point est à $\mathcal{O}$.

3.4.1 Coordonnées

L'ajout d'une mémoire de type RAM permet de sauvegarder les coordonnées de ces points sans accaparer trop de ressources logiques.

L'interface de cette mémoire est illustrée à la figure 3.12. Le bus de données de la mémoire est connecté directement au bus principal du processeur. La largeur du bus de données est donc fixée par la taille de chacune des coordonnées des points (N).

Ainsi, on ne peut accéder qu'à une seule des coordonnées de point à la fois. Le bit le moins significatif du bus d'adresses est donc réservé pour sélectionner à laquelle des deux coordonnées (x ou y) on cherche à accéder.

Le reste du bus d'adresses permet de spécifier le point auquel on cherche à accéder. Le nombre de points dans la table de correspondance est fonction du paramètre w tel que mentionné à la section 3.3.1. La largeur de la partie la plus significative du bus d'adresses est ainsi fixée à $w-2$ conformément à l'équation (3.3). À cela on ajoute un bit pour sélectionner la coordonnée fixant la taille du bus d'adresses à $w-1$ bits.

Un multiplexeur permet d'adresser les points de trois différentes façons :

- Le point peut être sélectionné en fonction de la magnitude d'un chiffre appartenant à la clef privée (voir section 3.3).
- Un compteur est également disponible. Cette option est utile lors du remplissage de la table de correspondance avant qu'un point ne soit multiplié par la clef privée.
- Le point peut être spécifié directement par l'unité de séquencement.

L'unité de séquencement tel que décrite à la section 3.2 est de type microprogram-

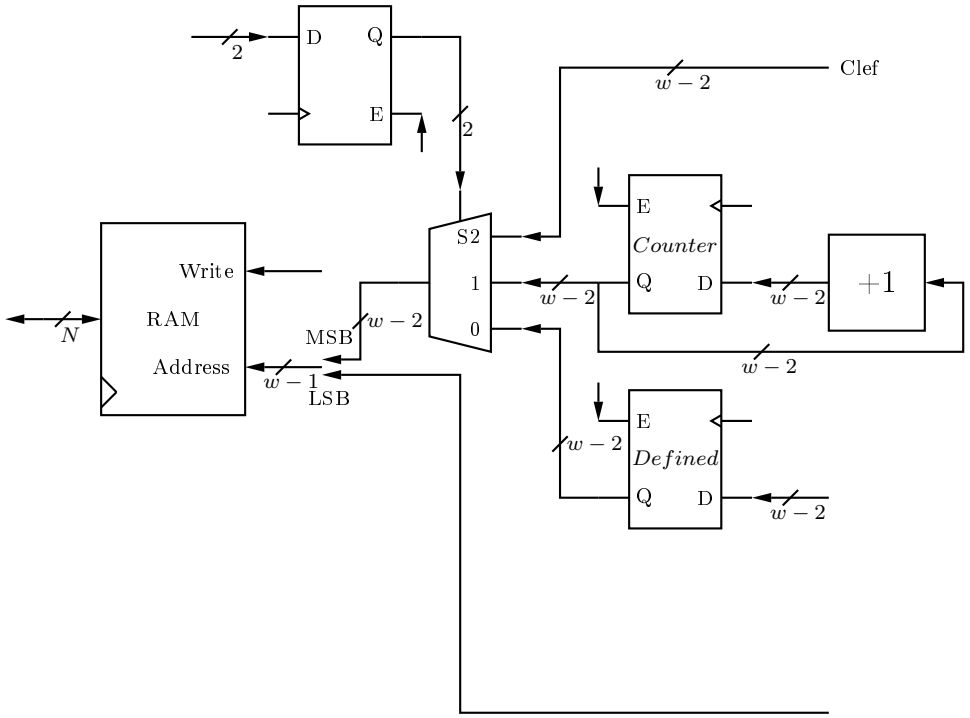

FIG. 3.12 – Interface de la table de correspondance

mée. Le protocole se prête en effet très bien à l'utilisation de sous-routines au niveau du séquencement (par exemple pour l'addition de deux points). Le point sélectionné dans la table de correspondance devient ainsi un argument de la sous-routine. Ce dernier se doit donc d'être sélectionné avant l'appel de la sous-routine.

Deux registres contrôlés à même le microcode sont ajoutés afin de pouvoir spécifier un point particulier avant l'appel de la sous-routine. Le premier sert à conserver l'adresse du point lorsque cette dernière est spécifiée à même le microcode (Registre *Defined* de la figure 3.12). L'autre est placé juste avant les bits de sélection du multiplexeur. L'ajout de ces deux registres permet ainsi d'écrire des routines microprogrammées sans avoir à spécifier de point particulier durant l'exécution de la sous-routine.

3.4.2 Points à $\mathcal{O}$

La mise en mémoire des coordonnées en x et y dans la table de correspondance ne suffit pas. Un bit d'information supplémentaire doit être ajouté pour spécifier si un point est à $\mathcal{O}$ ou pas.

L'accès à cette information se fait généralement de façon indépendante de l'accès

aux coordonnées du point. Une structure indépendante constituée de bascules plutôt que de mémoire RAM a donc été développée.

Cette structure est décrite à la figure 3.13. Elle fait appel aux mêmes $w-2$ bits connectés sur le bus d'adresse de la mémoire RAM (voir figure 3.12) comme bits de sélection du démultiplexeur et du multiplexeur. Deux autres signaux sont extraits directement du micromot afin de spécifier si l'on cherche à faire une écriture ainsi que la valeur à écrire.

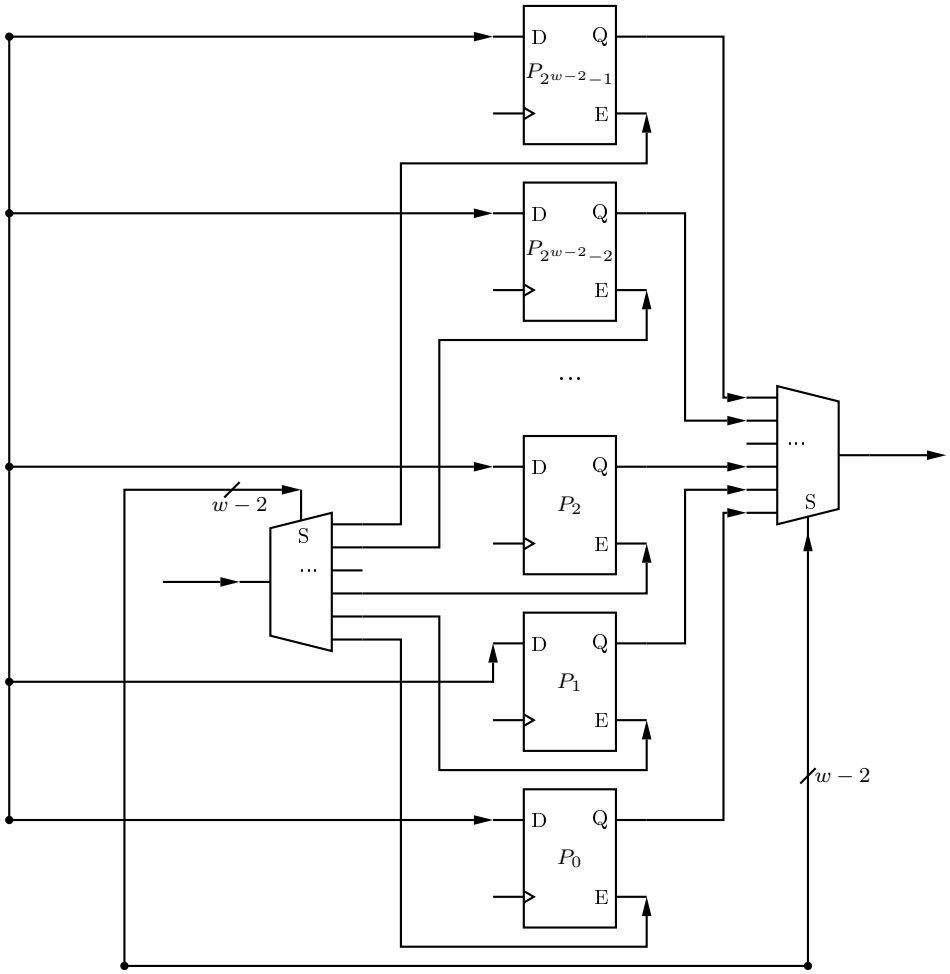

FIG. 3.13 – Annexe de la table de correspondance spécifiant si un point est à $\mathcal{O}$

3.5 Conclusion

Un processeur a été développé, capable de se conformer au protocole d'échange de clef Diffie-Hellman. Le processeur est décomposé en ses différentes unités décrites

brièvement à la section 3.1. Les unités sont reliées entre elles par un bus sur lequel transigent les coordonnées des points.

Le bus est arbitré par un séquenceur, également chargé de contrôler les différentes unités. La mécanique associée au protocole ECDH se prête bien à l'utilisation d'un séquenceur microprogrammé en raison du caractère répétitif des opérations s'y rattachant ainsi que de la complexité séquentielle de l'algorithme dans son ensemble.

Deux séquenceurs ont été développés à la section 3.2 selon les ressources disponibles sur le FPGA visé. Les deux séquenceurs ont été développés de façon à exploiter efficacement les ressources logiques disponibles sur un circuit FPGA. Les microinstructions associées aux deux séquenceurs permettent entre autres de faire des branchements conditionnels, des boucles, ainsi que des appels de sous-routines.

Afin d'accélérer le protocole d'échange de clef, la clef privée est représentée sous forme w-NAF. Une méthode pour générer la clef privée directement sous sa forme w-NAF est proposée à la section 3.3 ainsi que l'architecture matérielle associée à l'unité.

Afin de se conformer à l'algorithme 3.1, un certain nombre de points doivent être pré-calculés. Les coordonnées de ces points sont sauvegardées dans une table de correspondance basée sur une mémoire dynamique tel que décrite à la section 3.4. Une structure basée sur des bascules D est annexée à la mémoire dynamique spécifiant si un point est à $\mathcal{O}$ ou pas.

Chapitre 4

Unités arithmétiques sur GF(2^N)

4.1 Unité de multiplication polynomiale sur GF(2^N)

4.1.1 Algorithme de multiplication polynomiale

La multiplication polynomiale est tout-à-fait analogue à une multiplication binaire classique. Deux approches existent pour effectuer l'opération.

La multiplication de droite à gauche (Algorithme 4.1) est l'approche la plus intuitive.

Algorithme 4.1 Multiplication de droite à gauche

entrée : $A(x)$ un polynôme défini sur $GF(2^N)$
$B(x)$ un polynôme défini sur $GF(2^N)$
sortie : $R(x)$ un polynôme défini sur $GF(2^{2N-1})$

$R(x) \leftarrow 0$
pour $i = 0$ jusqu'à $N-1$ **faire**
  si $A_i = 1$ **alors**
    $R(x) \leftarrow R(x) + B(x)$
  fin si
  $B(x) \leftarrow xB(x)$
fin pour

Pour chaque $A_i = 1$, on ajoute simplement au résultat $x^i B(x)$. Le problème de la multiplication peut toutefois être attaqué différemment tel que présenté par l'algorithme 4.2.

Algorithme 4.2 Multiplication de gauche à droite

entrée : $A(x)$ un polynôme défini sur $GF(2^N)$
$B(x)$ un polynôme défini sur $GF(2^N)$
sortie : $R(x)$ un polynôme défini sur $GF(2^{2N-1})$

$R(x) \leftarrow 0$
pour $i = N-1$ jusqu'à 0 **faire**
  si $A_i = 1$ **alors**
    $R(x) \leftarrow R(x) + B(x)$
  fin si
  $R(x) \leftarrow xR(x)$
fin pour

Cette fois, le polynôme ajouté au résultat ne change pas. Le résultat est multiplié par x à chaque itération. La multiplication s'effectuant de gauche à droite et la multiplication étant une opération distributive, pour chaque $B(x)$ ajouté au résultat à l'itération i (lorsque $A_i = 1$), $x^i B(x)$ sera ultimement ajouté au résultat. Les deux algorithmes sont donc équivalents.

Cette approche devient particulièrement utile lorsque l'on doit réduire le polynôme résultant de la multiplication.

Réduction polynomiale

L'opération $A(x)$ modulo $P(x)$ consiste à soustraire $K(x)P(x)$ de $A(x)$ où $K(x)$ est choisi de façon à s'assurer que le résultat de l'opération soit de degré inférieur à N.

Une première approche pour obtenir ce résultat est détaillée par l'algorithme 4.3.

Tous les termes de degré supérieur à x^{N-1} sont éliminés un à un en commençant par le terme le plus significatif. Commencer par le bit le plus significatif permet de s'assurer que l'opération $R(x) \leftarrow R(x) + P(x)$ ne fera pas réapparaître un terme préalablement éliminé.

Cet aspect rend très intéressant l'algorithme 4.2 de multiplication de gauche à droite. En effet, les bits les plus significatifs sont connus en premier. L'étape de réduction peut donc se faire à même chaque itération en additionnant $P(x)$ au résultat si ce dernier est de degré N. Cette approche est intégrée à l'algorithme de multiplication donné en 4.4.

Algorithme 4.3 Réduction d'un polynôme

entrée : $A(x)$ un polynôme défini sur $GF(2^{2N-1})$
$P(x)$ un polynôme irréductible de degré N
sortie : $R(x)$ un polynôme défini sur $GF(2^N)$

$R(x) \leftarrow A(x)$
$P(x) \leftarrow x^{N-2}P(x)$
pour $i = 2N-2$ jusqu'à N **faire**
  si $R_i = 1$ **alors**
    $R(x) \leftarrow R(x) + P(x)$
  fin si
  $P(x) \leftarrow x^{-1}P(x)$
fin pour

Algorithme 4.4 Multiplication de deux polynôme sur $GF(2^N)$

entrée : $A(x)$ un polynôme défini sur $GF(2^N)$
$B(x)$ un polynôme défini sur $GF(2^N)$
$P(x)$ un polynôme irréductible défini sur $GF(2^{N+1})$ de degré N
sortie : $R(x)$ un polynôme défini sur $GF(2^N)$

$R(x) \leftarrow 0$
pour $i = N-1$ down to 0 **faire**
  $R(x) \leftarrow xR(x)$
  si $A_i = 1$ **alors**
    $R(x) \leftarrow R(x) + B(x)$
  fin si
  si $R_N = 1$ **alors**
    $R(x) \leftarrow R(x) + P(x)$
  fin si
fin pour
retourne $R(x)$

Un exemple de multiplication $(x^5+x^4+x)\times(x^5+x^3+x^2)$ en choisissant (x^6+x+1) comme polynôme irréductible sur $GF(2^6)$ est illustré à la figure 4.1.

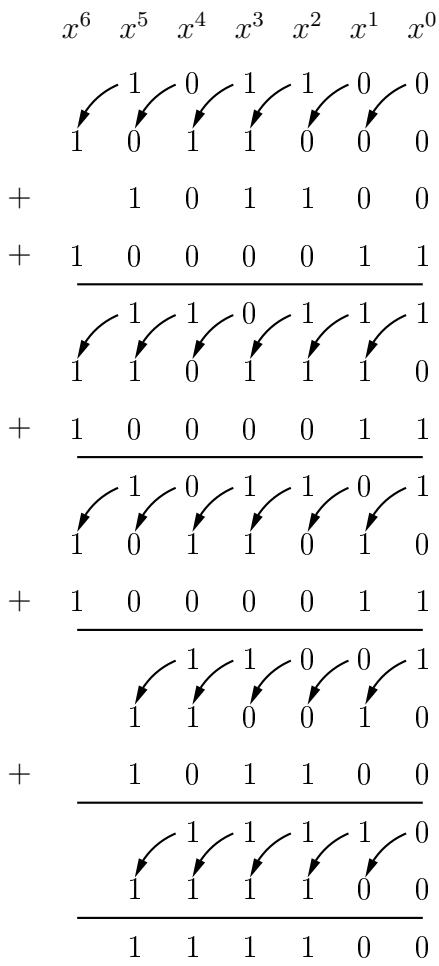

FIG. 4.1 – Exemple de multiplication de deux polynômes avec réduction entrelacé

4.1.2 Implantation matérielle

La version séquentielle d'un circuit se conformant à l'algorithme 4.4 est présentée ici. À chaque coup d'horloge, le circuit doit effectuer le travail équivalent à une itération de l'algorithme 4.4, soit deux additions conditionnelles successives.

Le problème peut être facilement décomposé en unités d'un seul bit. L'unité est définie spécifiquement en fonction de la valeur du bit correspondant au niveau du polynôme irréductible $P(x)$. Ce dernier en effet est connu à l'avance et fait donc parti des paramètres du circuits, au même titre que le point G.

L'unité correspondante à un 0 au niveau du polynôme irréductible n'a pas à se

charger d'additionner $p(x)$ au résultat le cas échéant où $R_N = 1$ puisque $P_i = 0$. Seule la première partie de l'algorithme 4.4 est implantée. Ainsi, 1 doit être additionné au résultat si le bit le plus significatif de A est à 1 et que le bit correspondant au niveau de B est également à 1.

Cette mécanique est réalisée à l'aide d'une simple porte ET. Une porte XOR est placée en sortie pour réaliser l'opération d'addition tel qu'illustré à la figure 4.2.

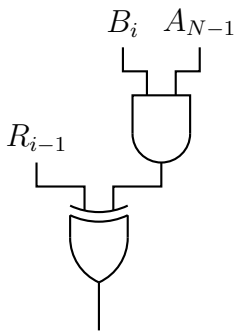

FIG. 4.2 – Unité de multiplication lorsque $P_i = 0$

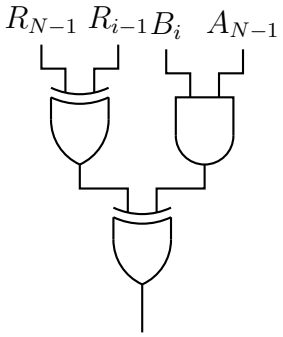

FIG. 4.3 – Unité de multiplication lorsque $P_i = 1$

Une seconde unité doit être utilisée lorsque le bit correspondant au niveau du polynôme irréductible P_i est à 1. Cette fois, une seconde addition conditionnelle doit être faite en fonction de la valeur de R_N. Une seconde porte XOR doit donc être ajoutée à l'unité pour effectuer cette addition.

Suivant l'algorithme 4.4, cette porte XOR devrait intuitivement être placée à la sortie de la première. Or, il est possible d'observer que la table de vérité du circuit demeure identique si la porte XOR en question est placée en entrée, de façon à travailler en parallèle avec la porte ET tel qu'illustré à la figure 4.3. Cette approche fait passer le délai de propagation de 3 portes logiques à 2.

La première unité ($i = 0$) de multiplication est un cas particulier. En effet, R_{-1} correspond au bit entrant lors du décalage. R_{i-1} est donc forcé à '0' au niveau de la première unité. On sait également que le bit le moins significatif du polynôme irréductible est forcément 1. Si ce n'était pas le cas, x serait facteur du polynôme et ce dernier ne serait plus irréductible.

En combinant ces deux éléments, on obtient le circuit de la figure 4.4. On note que le circuit est en fait identique à celui de la figure 4.2 à la différence près que l'entrée R_{i-1} est remplacée par R_{N-1}. L'unité en question peut ainsi être réutilisée.

Chaque unité peut être connectée au bit correspondant du registre R de façon à

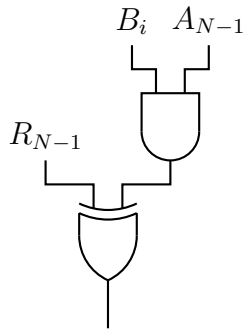

FIG. 4.4 – Unité de multiplication ($i = 0$)

effectuer un décalage vers la gauche de tous les bits conformément à l'algorithme 4.4. A est également placé dans un registre à décalage, encore une fois conformément à l'algorithme 4.4. Le circuit résultant est illustré à la figure 4.5.

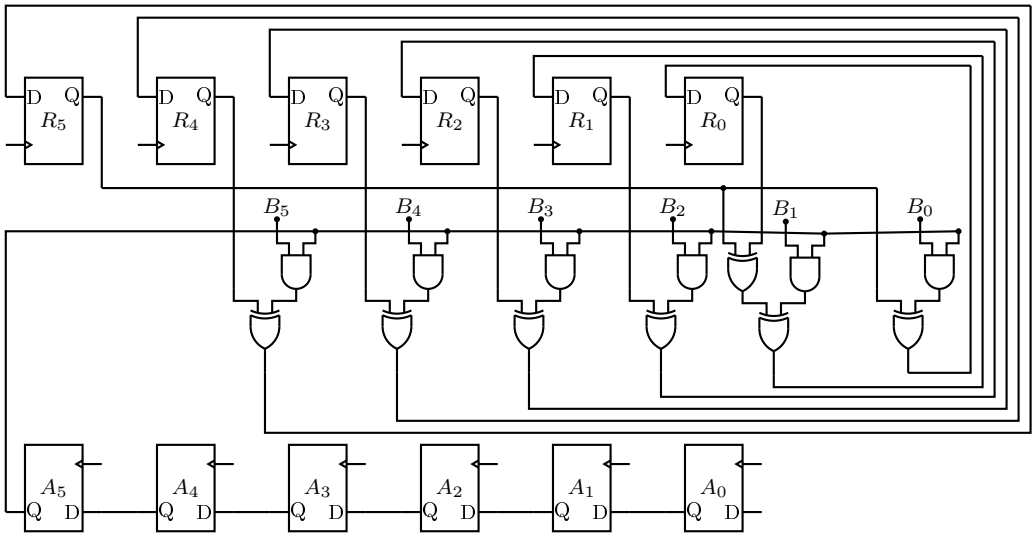

FIG. 4.5 – Multiplicateur polynomial de latence N correspondant au polynôme x

Un compteur associé à une machine à états a également été ajouté au circuit de façon à ce que ce dernier effectue un nombre d'itérations approprié. La machine à état permet également à l'unité de communiquer avec le reste du processeur.

Mise en cascade

Le plus long délai de propagation du circuit décrit à la section précédente est minime comparativement au délai de propagation de l'unité d'inversion polynomiale décrite à la section 4.3.2. Ainsi, la quantité de travail effectuée par l'unité de multiplication polynomiale en un seul coup d'horloge peut être augmentée sans affecter le chemin

critique du processeur.

Il est ainsi possible d'effectuer plusieurs itérations par coup d'horloge. À cette fin, plusieurs étages d'unités telles que décrites aux figures 4.2, 4.3, ainsi que 4.4 peuvent être mises en cascade. Par exemple, le circuit illustré à la figure 4.6 effectue 3 itérations de l'algorithme 4.4 par coup d'horloge.

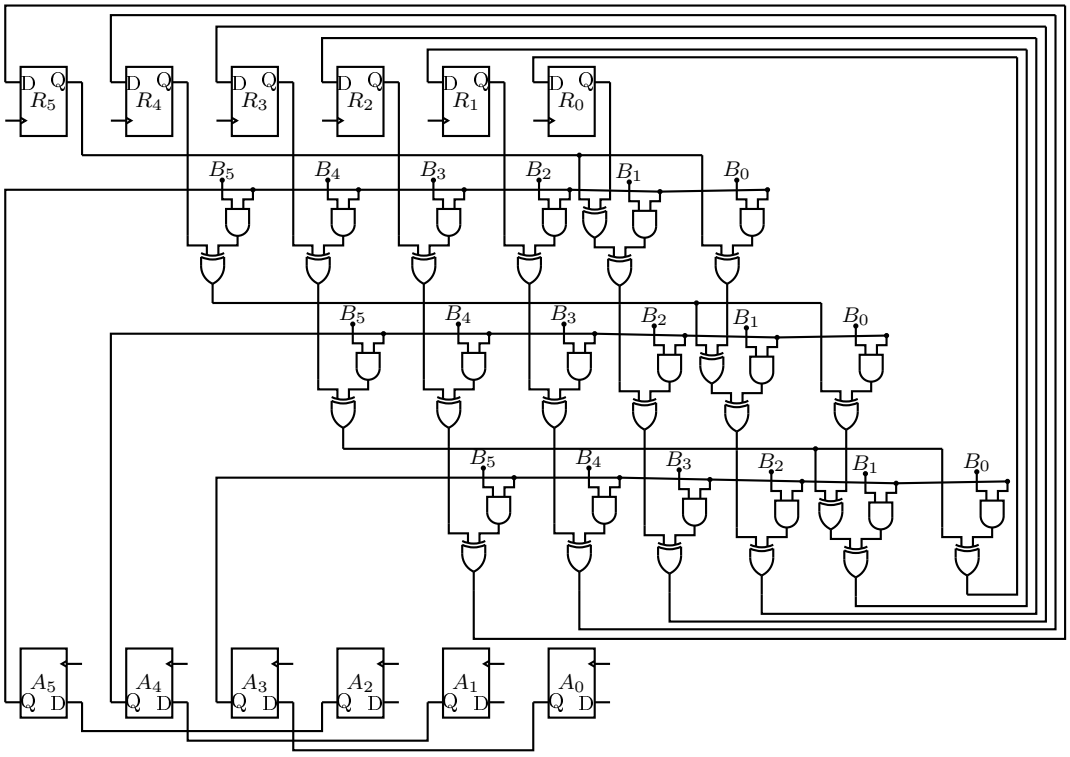

FIG. 4.6 – Multiplicateur polynomial à trois étages

Pour en arriver à mettre les unités en cascade, le registre A doit modifier son pas de décalage pour se conformer au nombre d'étages. Un problème peut se poser si N n'est pas un multiple du nombre d'étages. Il y aura alors un certain nombre d'itérations en trop. Pour contourner le problème, un tampon de bits fixés à 0 doit être concaténé à la gauche du registre A, le cas échéant.

Résultats d'implémentation

Deux des paramètres du circuit ont été étudiés dans cette section. Les résultats ont été obtenus en visant le FPGA Altera Statix EP1S80B956C6. Pour l'extension visée du corps de Galois ($GF(2^{163})$), le nombre d'étages (voir section 4.1.2) mis en cascade a été augmenté avec pour objectif d'obtenir un chemin critique pour l'unité se rapprochant

du chemin critique de l'unité d'inversion polynomiale.

Parmi tous les éléments du processeur étudiés dans ce mémoire, l'unité d'inversion polynomiale décrite à la section 4.3.2 est celle opérant le plus lentement. Pour l'extension visée du corps de Galois ($GF(2^{163})$), le tableau 4.4 donne une fréquence d'opération maximale de 99.73 MHz.

Au tableau 4.1, le nombre d'étages en cascade de l'unité de multiplication polynomiale a été augmenté jusqu'à 55, sans que la fréquence d'opération maximale ne tombe sous les 99.73 MHz. Le pas d'augmentation du nombre d'étages a été choisi de façon à minimiser ce dernier paramètre pour une latence donnée.

Le circuit a également été synthétisé pour toutes les extensions de corps de Galois données par le NIST [34] toujours en visant le FPGA Altera Statix EP1S80B956C6. Le nombre d'étages de l'unité de multiplication polynomiale a été choisi, pour une extension du corps de Galois $GF(2^N)$ donnée, de façon à ce que la fréquence d'opération de l'unité soit supérieur à la fréquence d'opération donnée au tableau 4.2.

Les résultats obtenus se comparent difficilement avec d'autres résultats présentés dans la littérature. Les unités traditionnelles n'utilisent pas le polynôme irréductible $P(x)$ comme paramètre du circuit mais plutôt comme une valeur pouvant être modifiée après synthèse, au même titre qu'une coordonnée [9] [11] [33]. Conséquemment, l'unité de multiplication polynomiale présentée dans ce mémoire se révèle plus performante que les implantations proposées dans la littérature, puisque $P(x)$ est fixé avant la synthèse.

Une autre approche à la multiplication est présentée par [33]. La multiplication y est faite de droite à gauche et la réduction polynomiale est faite, non pas sur le résultat mais sur le paramètre B. Conséquemment, la réduction polynomiale et la multiplication peuvent être effectués en parallèle.

Bien qu'en terme de portes logiques le délai de propagation soit équivalent, cette approche peut se révéler plus performante en CMOS puisque le délai de propagation à travers une porte XOR est généralement plus long que celui à travers une porte ET. Toutefois, la structure générée par cette approche n'est pas paramétrable pour n'importe quel polynôme irréductible et n'a donc pas été retenue. De plus, le design visant une implantation FPGA où les fonctions logiques sont implémentées à partir de tables de correspondances, cette approche n'aurait pas été nécessairement plus performante.

TAB. 4.1 – Fréquence d'opération maximum de l'unité de multiplication pour différentes latences

H	**Latence**	**Fréquence d'opération maximale**	**Éléments logiques**
	Coups d'horloge	**MHz**	
1	163	265.67	510
2	82	257.67	510
3	55	214.27	674
4	41	177.68	684
5	33	213.72	851
6	28	216.64	1012
7	24	191.79	1021
8	21	160.49	1179
9	19	163.37	1189
10	17	173.67	1345
11	15	177.81	1509
12	14	158.98	1677
13	13	151.63	1683
14	12	154.89	1838
15	11	167.31	1881
17	10	151.45	2206
19	9	148.65	2402
21	8	144.32	2535
24	7	130.34	2851
28	6	129.80	3348
33	5	135.34	3860
41	4	115.46	4847
55	3	109.53	6395

TAB. 4.2 – Fréquence d'opération maximum et ressources logiques utilisées pour l'unité de multiplication polynomiale sur différentes extensions de $GF(2^N)$ implémentées avec le polynôme irréductible donné par le NIST

N	$P(x)$	H	**Fréquence d'opération maximale**	**Éléments logiques**
			MHz	
163	$x^{163} + x^7 + x^6 + x^3 + 1$	55	109.53	6395
233	$x^{233} + x^{74} + 1$	47	114.16	7879
283	$x^{283} + x^{12} + x^7 + x^5 + 1$	48	110.32	9725
409	$x^{409} + x^{87} + 1$	59	102.27	17271
571	$x^{571} + x^{10} + x^5 + x^2 + 1$	72	95.11	28716

4.2 Unité de mise au carré polynomiale sur GF(2^N)

La mise au carré d'un polynôme est un cas particulier de la multiplication. Il est possible d'accélérer l'opération de multiplication d'un facteur proche de deux pour un polynôme irréductible judicieusement choisi.

4.2.1 Algorithme de mise au carré polynomiale

La mise au carré polynomiale est un opération particulièrement simple sur un corps non fini tel que le démontre ce qui suit :

$$\begin{aligned} R(x) &= (A(x))^2 \\ &= \left(\sum_{i=0}^{N-1} A_i x^i\right) \times \left(\sum_{j=0}^{N-1} A_j x^j\right) \\ &= \left(\sum_{i=0}^{N-1} A_i x^i \times \left(\sum_{j=0}^{N-1} A_j x^j\right)\right) \\ &= \left(\sum_{i=0}^{N-1} \left(\sum_{j=0}^{N-1} A_i A_j x^{i+j}\right)\right). \end{aligned} \tag{4.1}$$

Si $i \neq j$, on obtient un terme $A_i A_j x^{i+j}$. Or, il existe forcément un autre terme $A_j A_i x^{j+i}$ identique venant s'ajouter au résultat. Sur une extension du corps de Galois $GF(2^N)$, l'addition de deux éléments identiques donne zéro. Seuls les termes où $i = j$ subsistent. (4.1) devient :

$$\begin{aligned} R(x) &= \sum_{i=0}^{N-1} A_i A_i x^{i+i}, \\ &= \sum_{i=0}^{N-1} A_i x^{2i}. \end{aligned} \tag{4.2}$$

L'étape de mise au carré devient ainsi très simple. La position de chaque coefficient différent de zéro est simplement multipliée par 2. Autrement dit, un zéro est inséré entre chaque terme du polynôme initial. Par exemple :

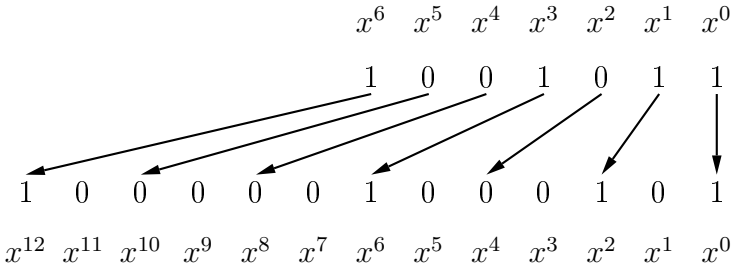

FIG. 4.7 – Exemple de mise au carré polynomiale

De façon matérielle, la mise au carré peut ainsi se faire directement lors de la saisie du polynôme par l'unité. Il suffit de mettre en place un registre de taille $2N - 1$ et de connecter un registre sur deux au bus de données, tout en initialisant les registres intercalés à zéro.

Il existe toutefois une complication. Le produit $K(x)P(x)$ doit être soustrait au résultat obtenu afin de ramener ce dernier dans $GF(2^N)$. L'algorithme 4.5 présente une façon simplifiée de réaliser l'opération de réduction.

Jusqu'ici, l'opération n'est pas plus efficace qu'une multiplication classique. Mais le résultat à réduire possède une caractéristique à exploiter. Seules les coefficients correspondant aux puissances paires sont différents de zéro. Il serait naturel de penser qu'au lieu de multiplier $P(x)$ par x^{-1}, il serait possible de le multiplier par x^{-2} puisque les coefficients entrelacés n'ont pas besoin d'être réduits.

Malheureusement, cette approche est trop simpliste. Une fois l'opération $R(x) \leftarrow$

Algorithme 4.5 Mise au carrée d'un polynôme défini sur $GF(2^N)$

entrée : $A(x)$ un polynôme défini sur $GF(2^{2N-1})$
$P(x)$ un polynôme irréductible défini sur $GF(2^{N+1})$ de degré N
sortie : $R(x)$ un polynôme défini sur $GF(2^N)$ $(A(x) \bmod P(x))$

$P(x) \leftarrow x^{N-2}P(x)$
tant que $\deg(R(x)) \geq N$ **faire**
  si $R_{\deg(P(x))} = 1$ **alors**
    $R(x) \leftarrow R(x) + P(x)$
  fin si
  $P(x) \leftarrow x^{-1}P(x)$
fin tant que

$R(x) + P(x)$ effectuée, les zéros entrelacés risquent de changer. Mais il est possible de retarder le problème en choisissant judicieusement le polynôme irréductible $P(x)$.

$P(x)$ doit être choisi de façon à minimiser $\deg(P(x) + x^N)$. Autrement dit, $P(x)$ doit être choisi de façon à maximiser la séquence de coefficients à zéro devant x^N. Par exemple, entre les deux polynômes irréductibles de degré dix présentés ci-dessous, (4.3) serait le plus approprié.

$$x^{10} + x^3 + 1 = 10000001001 \tag{4.3}$$
$$x^{10} + x^8 + x^5 + x^4 + 1 = 10100110001 \tag{4.4}$$

Cette séquence de zéros garantit qu'une plage plus ou moins importante de zéros entrelacés dans $R(x)$ ne sera pas affectée par $P(x)$ pendant l'opération de réduction. Un exemple de réduction en choisissant (4.3) comme polynôme irréductible sur $GF(2^N)$ est détaillé à la figure 4.8.

On note qu'il est possible de sauter une étape après chacune des trois premières étapes de réduction. Plus le degré du polynôme $(P(x)+x^N)$ est petit pour une extension de corps de Galois $GF(2^N)$ donnée, plus il est possible de sauter des étapes.

L'algorithme 4.6 intègre cette technique de réduction à l'algorithme de mise au carré d'un polynôme défini sur $GF(2^N)$.

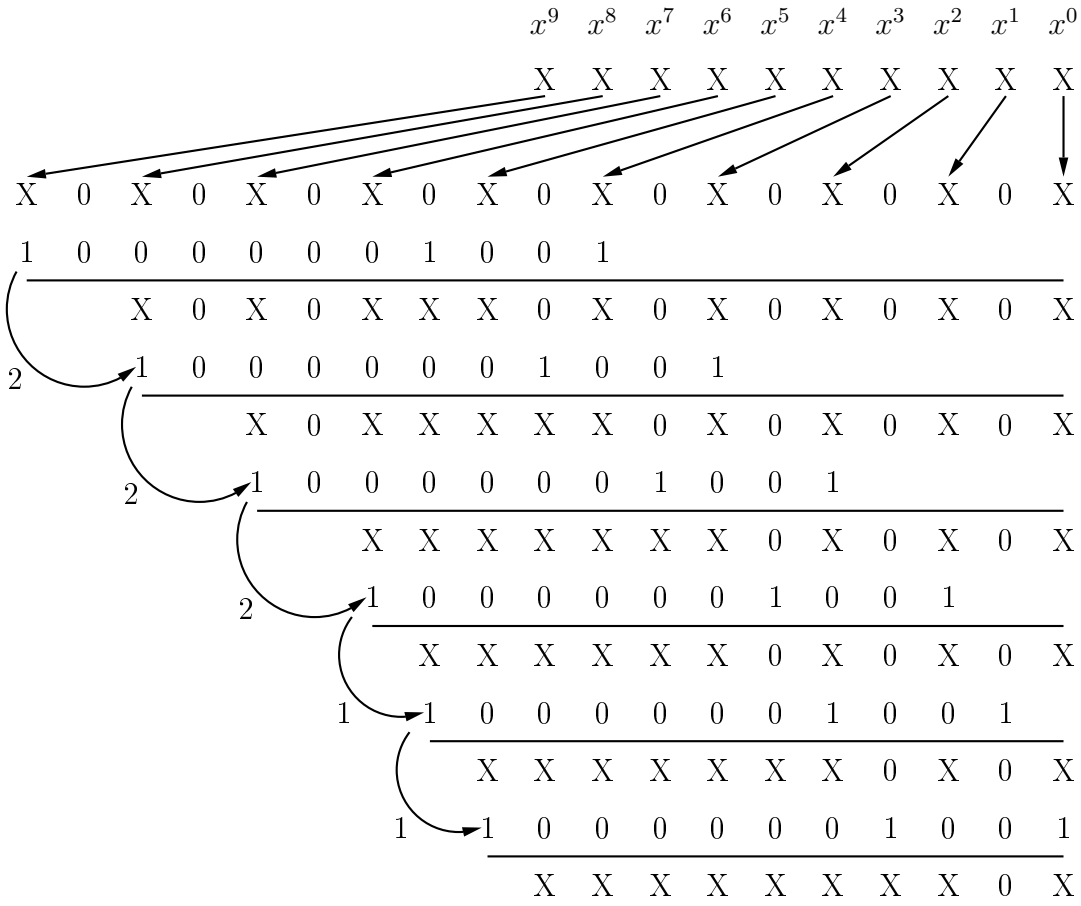

FIG. 4.8 – Exemple de réduction efficace d'un polynôme

4.2.2 Implantation matérielle

Une première version séquentielle du circuit est présentée à la figure 4.9. $x^4 + x + 1$ a été choisi comme polynôme irréductible sur l'extension du corps de Galois $GF(2^4)$.

Le circuit se conforme strictement à l'algorithme 4.6. Une seule itération est effectuée par coup d'horloge. Le circuit se comporte comme un registre à décalage variable. L'ensemble du registre est décalé de un ou deux bits en fonction de la valeur du second bit le plus significatif.

Un multiplexeur connecté à chaque bascule est chargé de choisir entre un décalage de un ou deux bits. À l'avant-dernière itération, une porte ET s'assure de limiter le décalage à un bit. Le circuit évite ainsi l'exécution d'une itération excédentaire.

La latence du circuit est ainsi diminuée en moyenne à près de $\frac{N}{2}$. En effet, les particularités de l'opération de mise au carré garantissent que chaque multiplexeur

Algorithme 4.6 Mise au carré d'un polynôme défini sur $GF(2^N)$

entrée : $A(x)$ un polynôme défini sur $GF(2^N)$
$P(x)$ un polynôme irréductible défini sur $GF(2^{N+1})$ de degré N
sortie : $R(x)$ un polynôme défini sur $GF(2^N)$

$R(x) \leftarrow (A(x))^2$
$P(x) \leftarrow x^{N-2}P(x)$
tant que $R_{\deg(P(x))-1} = 0$ **faire**
  si $R_{\deg(P(x))} = 1$ **alors**
    $R(x) \leftarrow R(x) + P(x)$
  fin si
  $P(x) \leftarrow x^{-2}P(x)$
fin tant que
tant que $\deg(R(x)) \geq N$ **faire**
  $P(x) \leftarrow x^{-1}P(x)$
  si $R_{\deg(P(x))} = 1$ **alors**
    $R(x) \leftarrow R(x) + P(x)$
  fin si
fin tant que
retourne $R(x)$

choisisse presque toujours d'effectuer un décalage de deux bits.

Mise en parallèle

L'algorithme 4.6 se prête assez mal à une implantation série. Le circuit de la figure 4.9 consomme une quantité imposante de ressources logiques pour ne réduire la latence du circuit que d'un facteur deux.

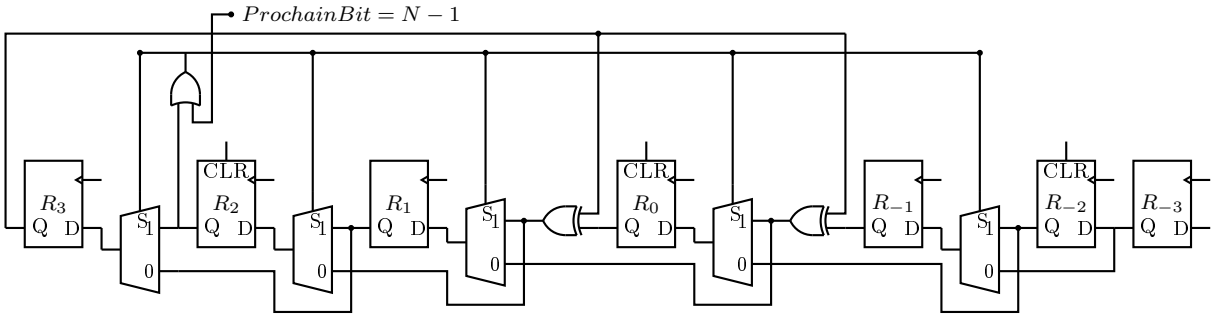

FIG. 4.9 – Unité non cascadée de mise au carré polynomiale

L'unité de mise au carrée série n'a évidemment pas sa place dans le processeur proposé, l'unité de multiplication polynomiale disposant d'une latence beaucoup plus faible grâce à la parallélisation partielle de l'algorithme 4.4.

La prochaine étape dans le processus de conception de l'architecture de l'unité de mise au carré polynomiale serait de tenter de paralléliser partiellement le circuit de la figure 4.9.

Une parallèlisation partielle du circuit se révèle difficile vu le caractère dynamique du décalage entre chaque itération. Une parallèlisation complète du circuit a été choisie. Le fréquence d'opération du circuit, même lorsque la parallèlisation du circuit est complète, se révèle beaucoup plus élevée que la fréquence d'opération associée à l'unité d'inversion polynomiale. La quantité de ressources logiques utilisées demeure également relativement faible.

L'architecture proposée repose sur le circuit de la figure 4.10. Le polynôme x^7+x^3+1 a été choisi comme polynôme irréductible sur $GF(2^7)$. Seuls trois étages de réduction sont nécessaires puisque R_{11}, R_9 et R_7 sont toujours à zéro.

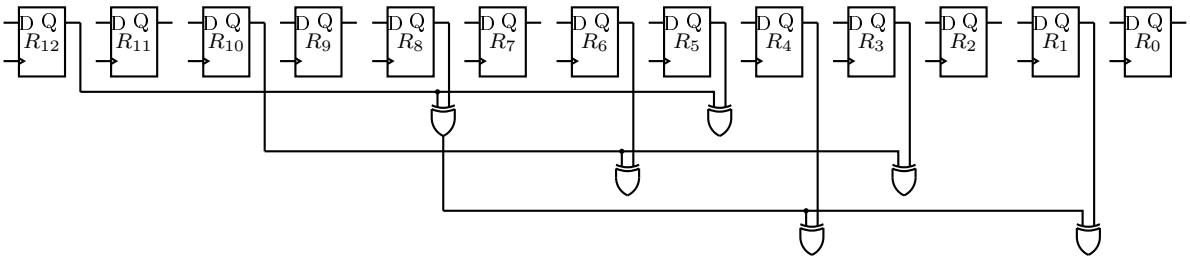

FIG. 4.10 – Unité parallèle de mise au carré polynomiale

Les bascules aux positions impaires étant toujours à zéro, ces dernières peuvent être enlevées, modifiant le circuit de la figure 4.10 pour obtenir le circuit présenté à la figure 4.11.

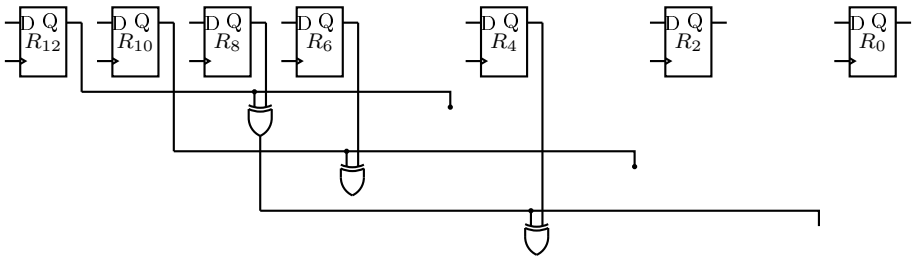

FIG. 4.11 – Unité parallèle simplifiée de mise au carré polynomiale

Dans cet exemple, le délai de propagation n'est que de deux portes XOR. Le délai de

TAB. 4.3 – Fréquence d'opération maximum et ressources logiques utilisées pour l'unité d'inversion polynomiale sur différente extensions de $GF(2^N)$ implémenté avec le polynôme irréductible donné par le NIST

N	$P(x)$	**Fréquence d'opération maximale**	**Éléments logiques**
		MHz	
163	$x^{163} + x^7 + x^6 + x^3 + 1$	351.86	172
233	$x^{233} + x^{74} + 1$	342.11	238
283	$x^{283} + x^{12} + x^7 + x^5 + 1$	283.37	295
409	$x^{409} + x^{87} + 1$	261.64	414
571	$x^{571} + x^{10} + x^5 + x^2 + 1$	247.83	576

propagation est en fait limité au nombre de uns présents dans le polynôme irréductible, exception faite du terme de poids N. Un seul registre de N bits est nécessaire pour sauvegarder l'opérande, puis le résultat au coup d'horloge suivant.

Résultats d'implémentation

L'unité a été synthétisée pour toutes les extensions du corps de Galois $G(2^N)$ suggérées par le NIST [34] en visant le FPGA Altera Statix EP1S80B956C6. Les résultats sont présentés au tableau 4.3.

Peu importe l'extension du corps de Galois visée, l'unité opère à une fréquence d'opération beaucoup plus élevée que l'unité d'inversion polynomiale (voir tableau 4.4). Ainsi, le chemin critique associé processeur ne devrait pas provenir de cette unité, même lorsque l'opération est entièrement parallélisée.

4.3 Unité d'inversion polynomiale sur GF(2^N)

L'inversion polynomiale intervient lors du calcul du paramètre λ aux équation (2.14) et (2.17). Une division entre deux termes appartenant au corps de Galois $GF(2^N)$ y est nécessaire.

Il a été démontré à la section 2.3.3 qu'une division sur un corps de Galois se décompose en une inversion suivie d'une multiplication polynomiale. L'unité doit donc se charger de trouver $R(x)$ tel que :

$$R(x)A(x) \bmod P(x) \quad = \quad 1. \tag{4.5}$$

4.3.1 Algorithme d'inversion polynomiale

L'algorithme AIA (*Almost Inverse Algorithm*) a été proposé par R. Schroeppel et al. [32] pour trouver $R(x)$ à l'équation (4.5). L'algorithme AIA se base sur l'algorithme GCD [14], ainsi que [3] pour trouver $R(x)$ et l tel que :

$$A(x)R'(x) \bmod \quad P(x) \quad = \quad x^l \text{ tel que } l < 2N.$$

L'appellation de l'algorithme tire son origine de l'exposant résiduel l. Afin d'extraire le véritable inverse multiplicatif de $A(x)$, $R'(x)$ doit être multiplié par x^{-l} au moyen d'un simple décalage.

Une version améliorée de l'algorithme a été proposée par D. Hankerson [12]. La version modifiée de l'algorithme MAIA (*Modified Almost Inverse Algorithm*) fournit directement le résultat de l'inversion. L'algorithme 4.7 détaille la méthode proposée. En moyenne, la latence associée à l'unité serait de $3N$ coups d'horloges [25].

4.3.2 Implantation matérielle

Un circuit basé sur une machine à états a été développé afin de se conformer à l'algorithme 4.7. Deux états, *Invert1* et *Invert2* se chargent d'effectuer une seule boucle de l'itération principale tel qu'illustré à la figure 4.12.

Une opération associée à l'état *Invert2* retient l'attention. Le degré de deux polynômes doit être comparé. Une structure récursive [30] a été développée à cette intention. Le problème est ainsi divisé récursivement en deux sous problèmes, comparant les degrés des bits les plus significatifs d'une part et des moins significatifs de l'autre. L'algorithme 4.8 détaille cette approche.

Algorithme 4.7 Inversion d'un polynôme défini sur $GF(2^N)$

entrée : $A(x)$ un polynôme défini sur $GF(2^N)$
$P(x)$ un polynôme irréductible défini sur $GF(2^{N+1})$ de degré N
sortie : $R(x)$ un polynôme défini sur $GF(2^N)$

$R(x) \leftarrow 1$
$I_1(x) \leftarrow 0$
$I_2(x) \leftarrow A(x)$
$I_3(x) \leftarrow P(x)$
boucler
  tant que $I_{2_0} = 0$ **faire**
    $I_2(x) \leftarrow x^{-1} I_2(x)$
    si $R_0 = 1$ **alors**
      $R(x) \leftarrow x^{-1}(R(x) + P(x))$
    sinon
      $R(x) \leftarrow x^{-1} R(x)$
    fin si
  fin tant que
  si $I_2(x) = 1$ **alors**
    retourne $R(x)$
  fin si
  si $\deg(I_2(x)) < \deg(I_3(x))$ **alors**
    $I_2(x) \leftrightarrow I_3(x)$
    $I_1(x) \leftrightarrow R(x)$
  fin si
  $I_2(x) \leftarrow I_2(x) + I_3(x)$
  $R(x) \leftarrow R(x) + I_1(x)$
fin boucle

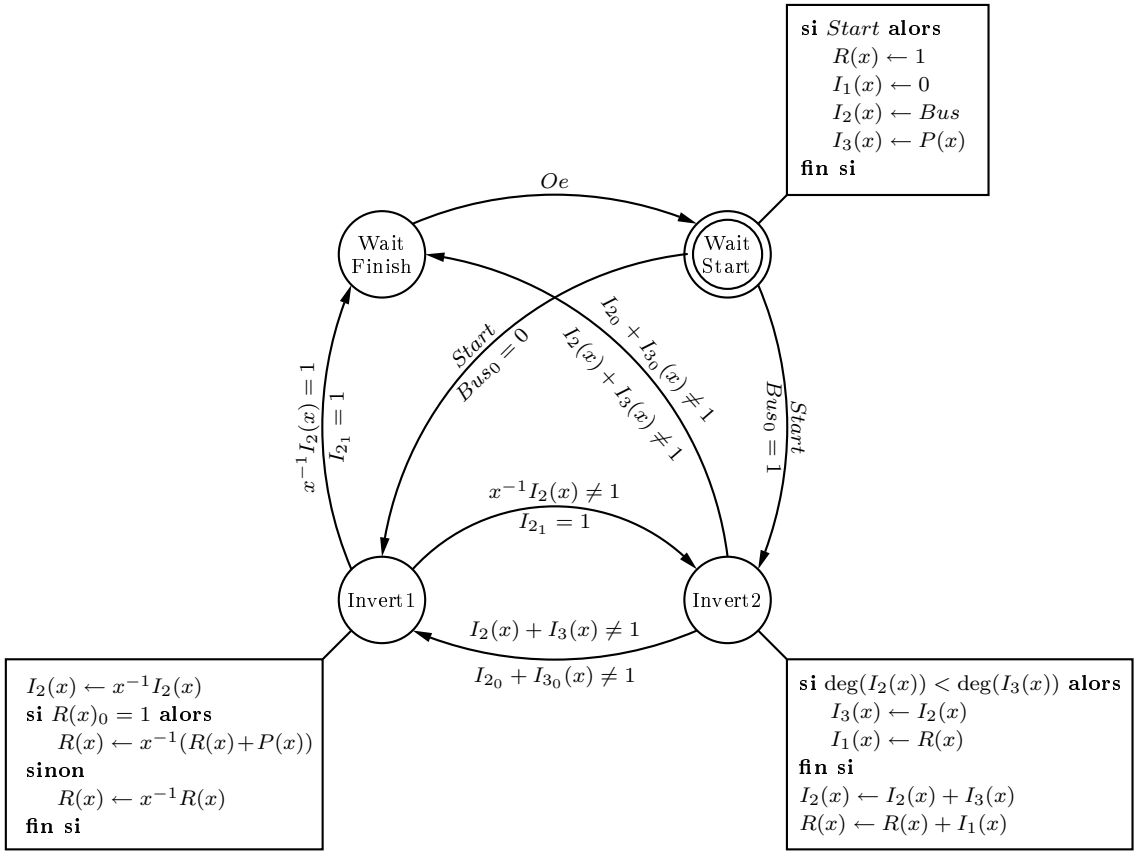

FIG. 4.12 – Machine à états associée à l'unité d'inversion polynomiale

Algorithme 4.8 Algorithme récursif pour comparer le degré de deux polynomes

entrée : $A(x)$ un polynôme défini sur $GF(2^N)$
$B(x)$ un polynôme défini sur $GF(2^N)$
sortie : $R = 1$ si $\deg(A(x)) > \deg(B(x))$, 0 sinon

si $taille(A(x)) = 1$ **alors**
  Cas de base
  si $A(x) = 1 \wedge B(x) = 1$ **alors**
    $R \leftarrow 1$
  sinon
    $R \leftarrow 0$
  fin si
sinon
  Cas récursif
  si $A(x)_{MSB} = 0 \wedge B(x)_{MSB} = 0$ **alors**
    si $\deg(A(x)_{LSB}) > \deg(B(x)_{LSB})$ **alors**
      $R \leftarrow 1$
    sinon
      $R \leftarrow 0$
    fin si
  sinon
    si $\deg(A(x)_{MSB}) > \deg(B(x)_{MSB})$ **alors**
      $R \leftarrow 1$
    sinon
      $R \leftarrow 0$
    fin si
  fin si
fin si

L'algorithme se distingue de la plupart des approches récursives en ce sens que le travail effectué par les deux appels récursifs n'est pas symétrique. Une seule information est extraite de l'appel sur les bits les moins significatifs, à savoir si le degré du second polynôme est plus grand que celui du premier. La même information est extraite des bits les plus significatif seulement cette fois, on doit en plus savoir si un 1 est présent.

Comme la plupart des algorithmes récursifs, le traitement réservé au cas de base est différent de celui associé au cas récursif. Quatre unités ont ainsi été développées, deux pour le cas de base (figures 4.13 et 4.14) et deux pour le cas récursif (figures 4.15 et 4.16).

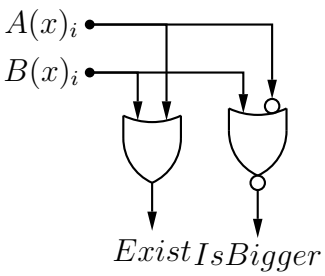

FIG. 4.13 – Unité de base de comparaison des degrés de deux polynômes (bits les plus significatifs)

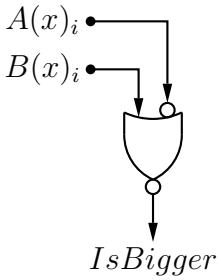

FIG. 4.14 – Unité de base de comparaison des degrés de deux polynômes (bits les moins significatifs)

L'unité développée à la figure 4.14 n'est qu'une simple porte logique. Le signal $IsBigger$ ne passe à un que si $A(x)_i$ est à un et $B(x)_i$ est à zéro, autrement dit, que $A(x)_i > B(x)_i$. À cette unité, se greffe une porte OU à la figure 4.13 signifiant qu'un des bits est à un.

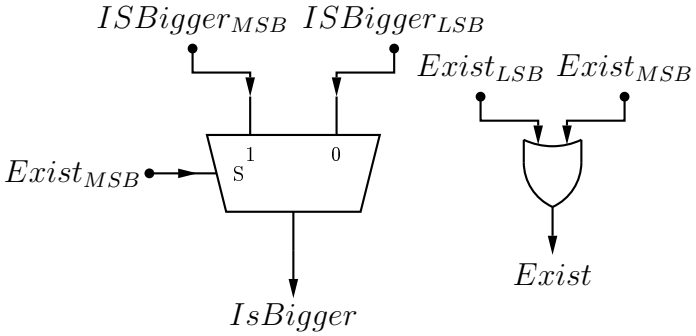

FIG. 4.15 – Unité récursive de comparaison des degrés de deux polynômes (bits les plus significatifs)

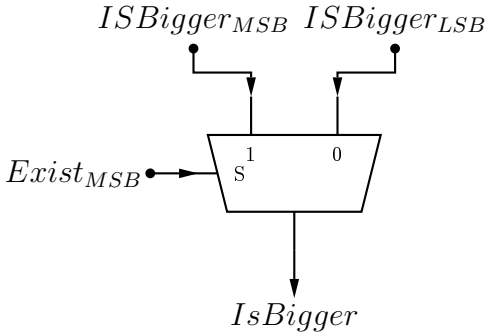

FIG. 4.16 – Unité récursive de comparaison des degrés de deux polynômes (bits les moins significatifs)

Les unités développées pour le cas récursifs (figures 4.15 et 4.16) utilisent un multiplexeur pour laisser passer $IsBigger_{MSB}$ ou $IsBigger_{LSB}$ en fonction de l'existence d'un 1 dans les bits les plus significatifs.

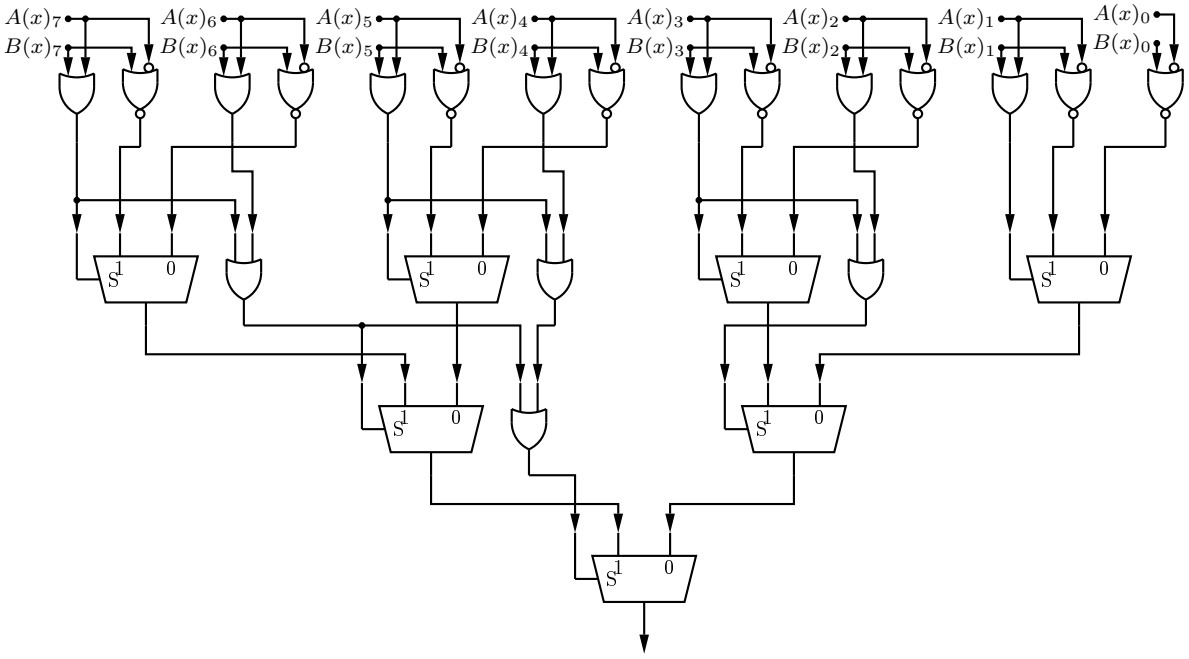

FIG. 4.17 – Architecture récursive de comparaison du degré de deux polynômes

Les différentes unités sont connectées récursivement pour obtenir le circuit présenté à la figure 4.17, comparant le degré de deux polynômes définis sur l'extension du corps de Galois $GF(2^8)$.

La structure ainsi obtenue est facilement paramétrable. Le nombre d'étages présents dans l'arbre récursif est de $\lceil \log_2(N) \rceil$. Parmi toutes les unités paramétrables, l'unité d'inversion polynomiale est la plus lente. Sa fréquence d'opération est limitée par le délai de propagation des signaux à travers la structure présentée à la figure 4.17. Si N devient relativement élevé, le chemin critique associé au processeur peut venir de cette structure. Dans pareil cas, le chemin critique croît avec le logarithme de N, comme la plupart des compteurs présents à travers le processeur.

Résultats d'implémentation

L'unité a été synthétisée pour toutes les extensions du corps de Galois $G(2^N)$ suggérées par le NIST [34] en visant le FPGA Altera Statix EP1S80B956C6. Les résultats sont présentés au tableau 4.4.

L'unité opère bel et bien à une fréquence d'opération moins élevée que les deux autres unités d'arithmétiques polynomiales présentées aux sections 4.1 et 4.2. Ces résultats ont donc servi d'étalon pour les mesures de fréquence d'opération, les résultats présentés

TAB. 4.4 – Fréquence d'opération maximum et ressources logiques utilisées pour l'unité d'inversion polynomiale sur différente extensions de $GF(2^N)$ implémenté avec le polynôme irréductible donné par le NIST

N	$P(x)$	**Fréquence d'opération maximale**	**Éléments logiques**
		MHz	
163	$x^{163} + x^7 + x^6 + x^3 + 1$	99.73	1463
233	$x^{233} + x^{74} + 1$	113.35	2122
283	$x^{283} + x^{12} + x^7 + x^5 + 1$	107.18	2541
409	$x^{409} + x^{87} + 1$	98.78	3746
571	$x^{571} + x^{10} + x^5 + x^2 + 1$	81.69	5171

au tableau 4.4 étant par ailleurs très proches de ceux obtenus pour le processeur en entier.

4.4 Conclusion

L'unité de multiplication polynomiale décrite à la section 4.1 se conforme à l'algorithme de multiplication polynomiale avec réduction entrelacée (algorithme 4.4). Vu la grande fréquence d'opération du circuit dans sa version série, le travail à été partiellement parallélisé afin d'effectuer plusieurs itérations de l'algorithme par coup d'horloge.

Cette parallélisation s'est faite par l'intermédiaire de la mise en cascade des unités décrites aux figures 4.2, 4.3 ainsi que 4.4 sur un nombre d'étages paramétrable. En faisant varier le nombre d'étages, il a été possible de diminuer la latence de l'unité à trois coups d'horloge sans que la fréquence d'opération de cette dernière ne descende en dessous de la fréquence d'opération du processeur.

La mise en cascade de l'unité de multiplication polynomiale a permis d'améliorer de façon significative le temps de traitement de l'opération. Le gain de performance, par rapport à une implantation logicielle s'est révélé beaucoup plus grand que pour l'unité d'inversion polynomiale.

Ce gain de performance combiné à la latence inexistante du module de mise au

carré polynomiale ouvre la porte à l'utilisation de coordonnées projectives (voir annexe A), même en représentant les coordonnées des points sous forme polynomiale. Le gain de performance de ces unités ayant été atteint une fois l'architecture du processeur entièrement élaborée, les coordonnées affines ont été retenues.

Une unité spécialisée de multiplication a été développée indépendamment pour traiter le cas où les termes à multiplier sont identiques. Il a été démontré à la section 4.2.1 que la mise au carré en soi peut se faire directement lors de la saisie du polynôme. Le nombre d'étapes associées à la réduction du polynôme résultant de l'opération de mise au carré peut également être diminué d'un facteur proche de deux en choisissant judicieusement le polynôme irréductible associé à $GF(2^N)$.

Deux architectures matérielles sont proposées à la section 4.2.2. L'architecture série se prête plutôt mal à la mise au carré polynomiale. La latence du circuit proposé étant de beaucoup supérieure à l'unité de multiplication polynomiale, le circuit a été parallélisé. Les résultats de synthèse ont démontré que l'unité ne risque pas d'affecter la fréquence d'opération du processeur.

L'unité opère en effet à une fréquence beaucoup plus élevée que l'unité d'inversion polynomiale décrite à la section 4.3. L'unité se base sur l'algorithme *Modified Almost Inverse Algorithm (MAIA)* nécessitant de comparer le degré de deux polynômes définis sur l'extension du corps de Galois $GF(2^N)$. À cette fin, une structure récursive a été développée, générant un chemin critique proportionnel à $\lceil \log_2(N) \rceil$.

Chapitre 5

Conclusion

Un processeur cryptographique capable de se conformer au protocole d'échange de clef de Diffie-Hellman a été développé. Ce cryptosystème utilise le groupe elliptique comme substitut au groupe multiplicatif.

Bien que toutes les unités soient paramétrables pour n'importe quelle extension du corps de Galois $GF(2^N)$, le cryptosystème vise l'extension du corps de Galois $GF(2^{163})$. La courbe elliptique choisie est celle recommandée par le NIST [34] pour cette extension soit :

$$y^2 + xy \quad = \quad x^3 + x^2 + 1. \tag{5.1}$$

Le NIST recommande une courbe légèrement différente pour les autres extensions du corps de Galois $GF(2^N)$, courbe incompatible avec le microcode tel que présenté à l'annexe B. Les mesures des performances du circuit se sont donc limitées à l'extension du corps de Galois $GF(2^{163})$.

Un bus principal relie les différentes unités constituant le processeur. Ce bus est arbitré par un séquenceur, également chargé de contrôler les différentes unités et de réagir aux signaux d'interface externes. Le séquenceur est de type microprogrammé. Deux architectures ont été développées en fonction du FPGA visé.

Une première version de l'architecture utilise une micromémoire dont le bus d'adresses n'est pas isolé par un point de synchronisation, lorsque pareille mémoire est disponible sur le FPGA visé.

La seconde version de l'architecture utilise une micromémoire dont le bus d'adresses est isolé du reste du circuit par un point de synchronisation. Bien que diminuant le délai de propagation, cette architecture peut se révéler plus lente puisque les branchements doivent être calculés une microinstruction à l'avance, introduisant des microinstructions excédentaires.

Afin d'accélérer le protocole d'échange, la clef privée a été représentée sous sa forme w-NAF. La clef privée étant toujours générée aléatoirement, une transformation permettant de générer un nombre aléatoire directement en représentation w-NAF a été proposée à la section 3.3. Une architecture matérielle associée à la transformation a également été proposée.

Trois unités d'arithmétique polynomiale ont été greffées au processeur. Elles ont été décrites au chapitre 4. Le gain important de performances associé aux unités de multiplication ainsi que de mise au carré par rapport à une implantation logicielle ouvre la porte à l'utilisation de coordonnées projectives (voir annexe A).

5.1 Protocole de vérification de l'échange de clefs

Le processeur avec séquenceur pipeliné a été implanté sur un FPGA Altera Statix EP1S80B956C6. L'interface *Signal Tap* a été exploité afin de recueillir les informatisations échangées durant l'échange de clef.

L'interface *Signal Tap* permet d'observer facilement les signaux issus du traitement du processeur, mais ne permet pas d'interagir avec ce dernier. Une coquille a donc été développée afin d'interagir avec le processeur. La coquille ne fournit strictement que des sorties permettant de vérifier que le processeur se conforme adéquatement au protocole d'échange de clefs Diffie-Hellman, sans requérir la moindre entrée.

La coquille procède en émulant un interlocuteur générant toujours la même clef privée et, par conséquent, la même clef publique. Cette même clef publique est ensuite transmise au processeur en échange de la clef publique de dernier. Finalement, le processeur fournit sa clef partagée à la coquille.

Un script Matlab émulant le même interlocuteur que la coquille se charge ensuite de calculer une clef partagée à partir de la clef publique du processeur. Finalement, une vérification est faite pour vérifier que les deux clefs partagées sont identiques.

TAB. 5.1 – Latence moyenne de l'opération de multiplication d'un point sur la courbe elliptique et fréquences d'opérations du processeur pour les architectures pipelinée et non-pipelinée sur l'extension du corps de Galois $GF(2^N)$

w	H	**FPGA**	**Éléments logiques**	**Latence**	**Fréquence d'opération maximale**	**Temps de traitement**	**Architecture**
				Coups d'horloge	**MHz**	**ms**	
4	55	EP1M350F780C5	10662	84795.7	82.57	1.0270	N-P
4	55	EP1M350F780C5	10635	87226.9	93.11	0.9368	P
4	55	EP1S80B956C6	10174	87226.9	94.19	0.9279	P

5.2 Résultats

Les deux architectures ont été synthétisées en visant un FPGA de la compagnie Altera approprié :

- EP1S80B956C6 (Latence de 87226.9 coups d'horloge) pour la version pipelinée,
- EP1M350F780C5 (Latence de 84795.7 coups d'horloge) pour la vesion non-pipelinée.

L'architecture pipelinée a également été synthétisée visant le FPGA EP1M350F780C5 pour fins de comparaison, ce dernier disposant de beaucoup moins de ressources logiques, limitant les possibilités d'optimisation de l'outil de synthèse. Les résultats sont fournis au tableau 5.1.

L'architecture pipelinée s'est révélée légèrement plus rapide en synthèse. La différence de latence entre les deux architectures est relativement faible, expliquant en bonne partie les résultats obtenus sur le temps de traitement. C'est cet élément qui a été retenu pour comparer les résultats obtenus à ceux trouvés dans la littérature.

Le tableau 5.2 compile différents temps de traitement trouvés dans la littérature récente pour des circuits comparables.

TAB. 5.2 – Temps de traitement trouvés dans la littérature

Publication	N	**Temps de traitement**
		ms
Ernst et al., 2001 [7]	155	1.29
Leung et al., 2000 [18]	155	6.8
Mentens et al., 2004 [22]	160	3.801

Le circuit développé dans le cadre de ce mémoire se compare avantageusement aux différentes implantations trouvées dans la littérature. La représentation w-NAF de la clef privée ainsi que les performances des unités arithmétiques sur l'extension du cops de Galois $GF(2^N)$ expliquent cet avantage.

Ce mémoire ouvre la porte à des recherches plus approfondies. Il est de l'avis de l'auteur que l'utilisation de coordonnées projectives serait avantageuse, même en utilisant la forme polynomiale pour représenter les éléments de l'extension du corps de Galois $GF(2^N)$.

Bibliographie

[1] Essame Al-Daoud, Ramlan Mahmod, Mohammad Rushdan et Adem Kilicman : A New Addition Formula for Elliptic Curves over $GF(2^n)$. *IEEE Trans. Comput.*, 51(8):972–975, 2002.

[2] Paul H. Bardell, William H. McAnney et Jacob Savir : *Built-in test for VLSI : pseudorandom techniques*. Wiley-Interscience, New York, NY, USA, 1987.

[3] Elwyn R. Berlekamp : *Algebraic coding theory*. McGraw-Hill, New York, 1968.

[4] Ian F. Blake, Gadiel Seroussi et Nigel P. Smart : *Advances in elliptic curve cryptography*. Cambridge University Press, New York, 2004.

[5] B. W. Bomar : Implementation of Microprogrammed Control in FPGAs. *IEEE Trans. Ind. Electron.*, 29(2):415–422, 2002.

[6] Diffie, Whitfield et Martin E. Hellman : New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, novembre 1976.

[7] M. Ernst, S. Klupsch, O. Hauck et S. A. Huss : Rapid Prototyping for Hardware Accelerated Elliptic Curve Public-Key Cryptosystems. *In RSP '01 : Proceedings of the 12th International Workshop on Rapid System Prototyping*, page 24, Washington, DC, USA, 2001. IEEE Computer Society.

[8] Adrian E. Escott, John C. Sager, Alexander P. L. Selkirk et Dimitrios Tsapakidis : Attacking Elliptic Curve Cryptosystems Using the Parallel Pollard rho Method. *RSA's Laboratories CryptoBytes (Technical Newsletter)*, 4(2):15–19, Winter 1999. ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto4n2.pdf.

[9] C. Grabbe, M. Bednara, J. Teich, J. von zur Gathen et J. Shokrollahi : FPGA Designs of Parallel High Performance $GF(2^{233})$ Multipliers. *Proceedings of The IEEE International Symposium on Circuits and Systems (ISCAS-2003).*, 2:268—271, 2003.

[10] Vipul Gupta, Sumit Gupta, Sheueling Chang et Douglas Stebila : Performance analysis of elliptic curve cryptography for SSL. *In WiSE '02 : Proceedings of the ACM workshop on Wireless security*, pages 87–94, New York, NY, USA, 2002. ACM Press.

[11] Nils Gura, Sheueling Chang Shantz, Hans Eberle, Sumit Gupta, Vipul Gupta, Daniel Finchelstein, Edouard Goupy et Douglas Stebila : An End-to-End Systems Approach to Elliptic Curve Cryptography. *In CHES '02 : Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 349–365, London, UK, 2003. Springer-Verlag.

[12] Darrel Hankerson, Julio López Hernandez et Alfred Menezes : Software Implementation of Elliptic Curve Cryptography over Binary Fields. *In CHES '00 : Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–24, London, UK, 2000. Springer-Verlag.

[13] Marc Joye et Christophe Tymen : Compact Encoding of Non-adjacent Forms with Applications to Elliptic Curve Cryptography. *In PKC '01 : Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 353–364, London, UK, 2001. Springer-Verlag.

[14] Donald E. Knuth : *Seminumerical Algorithms*, volume 2 de *The Art of Computer Programming*. Reading, Mass., 1969.

[15] Donald E. Knuth : *The art of computer programming, volume 2 (3rd ed.) : seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[16] N. Koblitz : Elliptic curve cryptosystems. *Mathematics of Computation*, Vol. 48, No. 177, pp. 203-209, 1987.

[17] Paul Kohlbrenner et Kris Gaj : An embedded true random number generator for FPGAs. *In FPGA '04 : Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 71–78, New York, NY, USA, 2004. ACM Press.

[18] K. H. Leung, K. W. Ma, W. K. Wong et P. H. W. Leong : FPGA implementation of a Microcoded Elliptic Curve Cryptographic Processor. *In FCCM '00 : Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, page 68, Washington, DC, USA, 2000. IEEE Computer Society.

[19] Julio López et Ricardo Dahab : Improved Algorithms for Elliptic curve arithmetic in $GF(2^n)$. *In SAC '98 : Proceedings of the Selected Areas in Cryptography*, pages 201–212, London, UK, 1999. Springer-Verlag.

[20] Alfred Menezes, Scott Vanstone et Tatsuaki Okamoto : Reducing elliptic curve logarithms to logarithms in a finite field. *In STOC '91 : Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 80–89, New York, NY, USA, 1991. ACM Press.

[21] Alfred J. Menezes, Scott A. Vanstone et Paul C. Van Oorschot : *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[22] Nele Mentens, Siddika Berna Ors et Bart Preneel : An FPGA implementation of an elliptic curve processor $GF(2^m)$. *In GLSVLSI '04 : Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 454–457, New York, NY, USA, 2004. ACM Press.

[23] Victor S. Miller : Use of Elliptic Curves in Cryptography. *In CRYPTO '85 : Advances in Cryptology*, pages 417–426, London, UK, 1986. Springer-Verlag.

[24] J. Muir et D. Stinson : Minimality and Other Properties of the Width-w Nonadjacent Form. *Math. Comput.*, 75(253):369––384, 2005.

[25] Nghi Nguyen, Kris Gaj, David Caliga et Tarek El-Ghazawi : Implementation of Elliptic Curve Cryptosystems on a reconfigurable computer. *Proc. IEEE International Conference on Field-Programmable Technology, FPT 2003*, pages 60–67, décembre 2003.

[26] K. Okeya, K. Schmidt-Samoa, C. Spahn et T. Takagi : Signed binary representations revisited, 2004.

[27] Prakash Panjwani et Yuri Poeluev : *Additional ECC Groups For IKE*, juin 2000. Work in Progress.

[28] G. W. Reitwiesner : Binary arithmetic. *Advances in Computers*, Vol. 1, pp. 231-308, 1960.

[29] R. L. Rivest, A. Shamir et L. Adleman : A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[30] Sébastien Roy : Notes de cours : Systèmes vlsi GIF-19264 (automne 2005). Rapport technique.

[31] Takakazu Satoh et Kiyomichi Araki : Fermat Quotients and the Polynomial Time Discrete Log Algorithm for Anomalous Elliptic Curves.

[32] Richard Schroeppel, Hilarie Orman, Sean W. O'Malley et Oliver Spatscheck : Fast Key Exchange with Elliptic Curve Systems. *In CRYPTO '95 : Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 43–56, London, UK, 1995. Springer-Verlag.

[33] Leilei Song et Keshab K. Parhi : Low-Energy Digit-Serial/Parallel Finite Field Multipliers. *J. VLSI Signal Process. Syst.*, 19(2):149–166, 1998.

[34] U.S. National Institute of Standards and Technology (NIST) : Recommended elliptic curves for federal government use, 1999. http://csrc.nist.gov/encryption.

[35] U.S. National Institute of Standards and Technology (NIST) : Specification for the Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication 197 (FIPS PUB 197)*, novembre 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[36] U.S. National Institute of Standards and Technology (NIST) : A statistical test suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications, 2001. http://csrc.nist.gov/rng.

[37] Scott A. Vanstone, Ronald C. Mullin et Gordon B. Agnew : Elliptic curve encryption systems, octobre 2000.

[38] S. Wolfram : *A New Kind of Science*. Wolfram Media, 2002.

Annexe A

Coordonnées projectives

Bien que simplifiées, les formules d'addition démontrées à la section 2.3 présentent un obstacle important à l'implantation : l'opération d'inversion est en effet très ardue. Ce problème peut être contourné en effectuant un changement de coordonnées. López et Dahab [19] ont proposé une transformation évitant toute inversion lors de l'addition. La seule inversion nécessaire intervient lors du retour en coordonnées affine. La transformation est donnée par :

$$x = \frac{X}{Z}, \tag{A.1}$$

$$y = \frac{Y}{Z^2}. \tag{A.2}$$

A.1 Addition de deux points différents

Essame Al-Daoud et al. [1] ont proposé des équations pour l'addition de deux points différents exploitant la transformation donnée précédemment. Cette transformation contraint toutefois la coordonnée en Z de la première opérande à 1. Cette contrainte est respectée en posant $Z = 1$ lors du passage en coordonnées projectives. Il suffit ensuite de s'assurer que $P_1 = (X_1, Y_1, Z_1)$, la première opérande, ne soit pas le résultat d'une addition précédente. Aucune contrainte n'est imposée sur P_2. L'algorithme pour respecter ces contraintes est donné à la section A.3. L'addition est ainsi définie par :

$$
\begin{aligned}
U &= Z_2^2 Y_1 + Y_2, && \text{(A.3)}\\
S &= Z_2 X_1 + X_2, && \text{(A.4)}\\
T &= Z_2 S, && \text{(A.5)}\\
Z_3 &= T^2, && \text{(A.6)}\\
V &= Z_3 X_1, && \text{(A.7)}\\
X_3 &= U^2 + T(U + S^2 + Ta), && \text{(A.8)}\\
Y_3 &= (V + X_3)(TU + Z_3) + Z_3^2(Y_1 + X_1). && \text{(A.9)}
\end{aligned}
$$

Il peut être démontré que l'addition ainsi définie correspond bien aux résultats introduits à la section 2.3 :

$$
\begin{aligned}
\frac{X_3}{Z_3} &= \frac{U^2 + T(U + S^2 + Ta)}{T^2},\\
&= \frac{U^2}{T^2} + \frac{U + S^2 + Ta}{T},\\
&= \frac{(Z_2^2 Y_1 + Y_2)^2}{(Z_2 S)^2} + \frac{(Z_2^2 Y_1 + Y_2)}{Z_2 S} + \frac{S^2}{Z_2 S} + a,\\
&= \frac{(Z_2^2 Y_1 + Y_2)^2}{(Z_2(Z_2 X_1 + X_2))^2} + \frac{(Z_2^2 Y_1 + Y_2)}{Z_2(Z_2 X_1 + X_2)} + \frac{Z_2 X_1 + X_2}{Z_2} + a,\\
&= \frac{(Y_1 + \frac{Y_2}{Z_2^2})^2}{(X_1 + \frac{X_2}{Z_2})^2} + \frac{Y_1 + \frac{Y_2}{Z_2^2}}{X_1 + \frac{X_2}{Z_2}} + X_1 + \frac{X_2}{Z_2} + a. && \text{(A.10)}
\end{aligned}
$$

Comme on a posé $Z_1 = 1$, on peut réécrire :

$$
\begin{aligned}
\frac{X_3}{Z_3} &= \frac{(\frac{Y_1}{Z_1^2} + \frac{Y_2}{Z_2^2})^2}{(\frac{X_1}{Z_1} + \frac{X_2}{Z_2})^2} + \frac{\frac{Y_1}{Z_1^2} + \frac{Y_2}{Z_2^2}}{\frac{X_1}{Z_1} + \frac{X_2}{Z_2}} + \frac{X_1}{Z_1} + \frac{X_2}{Z_2} + a,\\
&= \frac{(y_1 + y_2)^2}{(x_1 + x_2)^2} + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a,\\
&= \lambda^2 + \lambda + x_1 + x_2 + a, && \text{(A.11)}
\end{aligned}
$$

ce qui correspond bien à (2.15). En répétant la même mécanique avec y, on trouve :

$$
\begin{aligned}
\frac{Y_3}{Z_3^2} &= \frac{(V+X_3)(TU+Z_3)+Z_3^2(Y_1+X_1)}{T^{2^2}}, \\
&= \frac{(Z_3X_1+X_3)(TU+Z_3)+Z_3^2(Y_1+X_1)}{T^4}, \\
&= \frac{TU(Z_3X_1+X_3)+Z_3(Z_3X_1+X_3)+Z_3(Z_3Y_1+Z_3X_1)}{T^4}, \\
&= \frac{TU(Z_3X_1+X_3)+Z_3(Z_3Y_1+X_3)}{T^4}, \\
&= \frac{TU(Z_3X_1+X_3)+Z_3(Z_3Y_1+X_3)}{T^4}, \\
&= \frac{U(Z_3X_1+X_3)}{T^3}+\frac{Z_3^2Y_1}{T^4}+\frac{Z_3X_3}{T^4}, \\
&= \frac{(Z_2^2Y_1+Y_2)(T^2X_1+X_3)}{T^3}+\frac{T^4Y_1}{T^4}+\frac{T^2X_3}{T^4} \\
&= \frac{(Z_2^2Y_1+Y_2)}{T}\frac{(T^2X_1+X_3)}{T^2}+Y_1+\frac{X_3}{T^2}, \\
&= \frac{(Z_2^2Y_1+Y_2)}{T}\frac{(T^2X_1+X_3)}{T^2}+Y_1+\frac{X_3}{Z_3}, \\
&= \frac{(Z_2^2Y_1+Y_2)}{Z_2(Z_2X_1+X_2)}(X_1+\frac{X_3}{Z_3})+Y_1+\frac{X_3}{Z_3}, \\
&= (\frac{Y_1+\frac{Y_2}{Z_2^2}}{X_1+\frac{X_2}{X_2}})(X_1+\frac{X_3}{Z_3})+Y_1+\frac{Z_3}{X_3}, \\
&= (\frac{y_1+y_2}{x_1+x_2})(x_1+x_3)+y_1+x_3, \qquad \text{(A.12)}
\end{aligned}
$$

ce qui correspond bien à (2.16). Ainsi, les formules introduites pour l'addition de points en coordonnées affines peuvent être transformées pour additionner les points en coordonnées projectives sans qu'une seule inversion n'ait lieu.

A.2 Addition d'un point avec lui-même

López et Dahab [19] ont introduit une formule exploitant le même changement de coordonnées mais pour le calcul de l'addition d'un point avec lui-même. La démonstration requiert toutefois une réécriture des formules proposées à la section 2.3.

En associant les coefficients des termes en x de (2.8) et (2.9), sachant que $x_1 = x_2$, puis en effectuant les simplifications sur le corps $GF(2^N)$, introduites à la section 2.3, on obtient :

$$\begin{aligned} x_1x_2 + x_1x_3 + x_2x_3 &= -y_0 - 2\lambda y_0, \\ x_1x_1 + (x_1x_3 + x_1x_3) &= y_0 + (\lambda y_0 + \lambda y_0), \\ x_1^2 &= y_0. \end{aligned} \tag{A.13}$$

En respectant le même principe mais cette fois en associant les termes en x^3, on obtient :

$$\begin{aligned} -x_1x_2x_3 &= b - y_0^2, \\ x_1^2x_3 &= b + y_0^2, \\ x_3 &= \frac{b + y_0^2}{x_1^2}. \end{aligned} \tag{A.14}$$

En substituant (A.13) dans (A.14), on obtient finalement :

$$\begin{aligned} x_3 &= \frac{b + (x_1^2)^2}{x_1^2}, \\ x_3 &= \frac{b}{x_1^2} + x_1^2. \end{aligned} \tag{A.15}$$

L'équation donnant y_3 doit également être modifiée. En récrivant (2.16,) on trouve :

$$\begin{aligned} y_3 &= \lambda(x_3 + x_1) + y_1 + x_3, \\ y_3 &= \lambda x_3 + \lambda x_1 + y_1 + x_3, \\ y_3 &= \lambda x_3 + (\frac{y_1}{x_1} + x_1)x_1 + y_1 + x_3, \\ y_3 &= \lambda x_3 + y_1 + x_1^2 + y_1 + x_3, \\ y_3 &= \lambda x_3 + x_1^2 + x_3. \end{aligned} \tag{A.16}$$

On note ensuite que l'équation de la tangente λ au point peut être modifiée en associant les équations de x_3, donnée en (2.18) et (A.15) :

$$
\begin{aligned}
x_3 = \lambda^2 + \lambda + a &= \frac{b}{x_1^2} + x_1^2, \\
(x_1 + \frac{y_1}{x_1})^2 + (x_1 + \frac{y_1}{x_1}) + a &= \frac{b}{x_1^2} + x_1^2, \\
x_1^2 + x_1\frac{y_1}{x_1} + x_1\frac{y_1}{x_1} + \frac{y_1^2}{x_1^2} + x_1 + \frac{y_1}{x_1} + a &= \frac{b}{x_1^2} + x_1^2, \\
\frac{y_1 2}{x_1^2} + x_1 + \frac{y_1}{x_1} + a &= \frac{b}{x_1^2}, \\
\lambda = x_1 + \frac{y_1}{x_1} &= \frac{b + y_1^2 + a x_1^2}{x_1^2}. \qquad \text{(A.17)}
\end{aligned}
$$

Finalement, en substituant (A.17) et (A.15) dans (A.16), on trouve :

$$
\begin{aligned}
y_3 &= x_1^2 + (x_1 + \frac{y_1}{x_1})x_3 + x_3, \\
&= \frac{b}{x_1^2} + (\frac{y_1^2 + b + a x_1^2}{x_1^2})x_3, \\
&= \frac{b}{x_1^2} + (\frac{y_1^2 + b}{x_1^2})x_3 + (\frac{a x_1^2}{x_1^2})x_3, \\
&= \frac{b}{x_1^2} + (\frac{y_1^2 + b}{x_1^2})(\frac{b}{x_1^2} + x_1^2) + a x_3, \\
&= \frac{b}{x_1^2} + (\frac{y_1^2 + b}{x_1^2})(\frac{b + x_1^4}{x_1^2}) + a x_3, \\
&= \frac{b}{x_1^2} + (y_1^2 + b)(\frac{b}{x_1^4} + 1) + a x_3. \qquad \text{(A.18)}
\end{aligned}
$$

Les formules d'addition d'un point en coordonnées projectives avec lui-même sont données par :

$$
\begin{aligned}
Z_3 &= Z_1^2 X_1^2, \qquad \text{(A.19)} \\
X_3 &= X_1^4 + b Z_1^4, \qquad \text{(A.20)} \\
Y_3 &= b Z_1^4 Z_3 + X_3(a Z_3 + Y_1^2 + b Z_1^4). \qquad \text{(A.21)}
\end{aligned}
$$

Il reste à démontrer que ces formules sont équivalentes à celles introduites précédemment. Pour la coordonnée en x on trouve :

$$\begin{aligned}
\frac{X_3}{Z_3} &= \frac{X_1^4 + bZ_1^4}{Z_1^2 X_1^2}, \\
&= \frac{X_1^4}{Z_1^2 X_1^2} + \frac{bZ_1^4}{Z_1^2 X_1^2}, \\
&= \frac{X_1^2}{Z_1^2} + \frac{bZ_1^2}{X_1^2}, \\
&= x_1^2 + \frac{b}{x_1^2}, \qquad \text{(A.22)}
\end{aligned}$$

ce qui correspond bien à (A.15). En poursuivant avec y on trouve :

$$\begin{aligned}
\frac{Y_3}{Z_3^2} &= \frac{bZ_1^4 Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4)}{(Z_1^2 X_1^2)^2}, \\
&= \frac{bZ_1^4 Z_3}{Z_1^4 X_1^4} + \frac{aZ_3 X_3}{Z_1^4 X_1^4} + \frac{Y_1^2 X_3}{Z_1^4 X_1^4} + \frac{bZ_1^4 X_3}{Z_1^4 X_1^4}, \\
&= \frac{bZ_3}{X_1^4} + \frac{aZ_3 X_3}{Z_1^4 X_1^4} + \frac{Y_1^2 X_3}{Z_1^4 X_1^4} + \frac{bX_3}{X_1^4}, \\
&= \frac{b(Z_1^2 X_1^2)}{X_1^4} + \frac{a(Z_1^2 X_1^2)(X_1^4 + bZ_1^4)}{Z_1^4 X_1^4} + \frac{Y_1^2(X_1^4 + bZ_1^4)}{Z_1^4 X_1^4} + \frac{b(X_1^4 + bZ_1^4)}{X_1^4}, \\
&= \frac{bZ_1^2}{X_1^2} + \frac{a(X_1^4 + bZ_1^4)}{Z_1^2 X_1^2} + \frac{Y_1^2(X_1^4 + bZ_1^4)}{Z_1^4 X_1^4} + \frac{b(X_1^4 + bZ_1^4)}{X_1^4}, \\
&= \frac{bZ_1^2}{X_1^2} + \frac{aX_1^4}{Z_1^2 X_1^2} + \frac{abZ_1^4}{Z_1^2 X_1^2} + \frac{Y_1^2 X_1^4}{Z_1^4 X_1^4} + \frac{Y_1^2 bZ_1^4}{Z_1^4 X_1^4} + \frac{bX_1^4}{X_1^4} + \frac{b^2 Z_1^4}{X_1^4}, \\
&= b(\frac{Z_1}{X_1})^2 + a(\frac{X_1}{Z_1})^2 + ab(\frac{Z_1}{X_1})^2 + (\frac{Y_1}{Z_1^2})^2 + b(\frac{Y_1}{Z_1^2})^2(\frac{Z_1}{X_1})^4 + b + b^2(\frac{Z_1}{X_1})^4, \\
&= \frac{b}{x_1^2} + ax_1^2 + \frac{ab}{x_1^2} + y_1^2 + \frac{by_1^2}{x_1^4} + b + \frac{b^2}{x_1^4}, \\
&= \frac{b}{x_1^2} + a(x_1^2 + \frac{b}{x_1^2}) + y_1^2(1 + \frac{b}{x_1^4}) + b(1 + \frac{b}{x_1^4}), \\
&= \frac{b}{x_1^2} + ax_3 + (1 + \frac{b}{x_1^4})(y_1^2 + b), \qquad \text{(A.23)}
\end{aligned}$$

ce qui correspond bien à (A.18).

A.3 Multiplication d'un point par un scalaire en coordonnées projectives

Certaines contraintes sont imposées lors de la multiplication d'un point P, définit en coordonnées projectives sur la courbe elliptique, par un scalaire k.

L'addition de deux points, $P_1 + P_2$, en utilisant (A.3) à (A.9) impose une contrainte particulière sur P_1. Ce dernier doit en effet respecter la forme :

$$P_1 \quad = \quad (X_1, Y_1, 1). \tag{A.24}$$

Les opérations introduites en (A.3) à (A.9) ainsi qu'en (A.19) à (A.21) n'imposent aucune contrainte sur la coordonnée en Z du point résultant. Il est donc essentiel de s'assurer que le point P_1 associé à l'addition ne résulte jamais d'une de ces opérations.

Afin d'y arriver, l'algorithme A.1 doit être respecté. En effectuant ainsi la multiplication à partir du bit le plus significatif, une seule opération d'addition ($R \leftarrow P + R$) intervient. Le premier opérande de la multiplication étant le point d'origine, la multiplication respectant A.1 se prête à l'utilisation de coordonnées projectives.

Algorithme A.1 Calcul de kP lorsque P est représenté en coordonnées projectives

entrée : k : un entier sur N bits
P : un point sur la courbe elliptique
sortie : $R = kP$

$R \leftarrow \mathcal{O}$
pour $i = N - 1$ down to 0 **faire**
  $R \leftarrow R + R$
  $R \leftarrow P + R$
fin pour
retourne R

L'algorithme se prête à priori très bien à l'utilisation de la forme w-NAF (voir section 3.3.1) pour représenter la clef privée puisqu'il est presque identique à l'algorithme 3.1. Par contre, les points extraits de la table de correspondance se doivent de fixer Z à "un" pour respecter les contraintes imposées par les coordonnées projectives. Une inversion serait donc nécessaire lors du calcul de chaque point dans la table de correspondance, augmentant le temps consacré à remplir cette dernière et diminuant du même coup la valeur optimale du paramètre w.

Annexe B

Microcode

Le microcode se substitue à une machine à états pour se conformer au protocole d'échange de clefs Diffie-Hellman sur une courbe elliptique. Le microcode développé vise spécifiquement l'extension du corps de Galois $GF(2^{163})$ et la courbe elliptique qui y est associée [34] :

$$y^2 + xy \quad = \quad x^3 + x^2 + 1. \tag{B.1}$$

Deux versions du microcode ont été développées en fonction des ressources logiques disponibles sur le FPGA visé.

B.1 Séquenceur non-pipeliné

Une première version du microcode est conçue pour un séquenceur non-pipeliné. Ce type de séquenceur devrait être utilisé si le FPGA visé dispose d'une mémoire statique sans point de synchronisation sur le bus d'adresses. 97 microinstructions sont nécessaires, limitant la taille du bus d'adresses à 7 bits.

START :**CONT**

DefinedLUAdress	←0
SelectLUAddress	←1
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif
ChangeLUAddress	←Actif

CALL $MULL_R$

NewRand	←Actif
InitX	←Actif
InitY	←Actif

CBT \$

BranchCondition	←*SetX*
Ready	←Actif

CONT

BusOe	←Actif
RxLoad	←Actif

CBT \$

BranchCondition	←*SetY*

CONT

BusOe	←Actif
RyLoad	←Actif

CALL $MULL_R$

CBT \$

BranchCondition	←*NewExchange*
Ready	←Actif

BR *START*

MUL_R :**CALL** $FILL_{LU}$

DefinedLUAdress	←0
SelectLUAddress	←2
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif
ChangeLUAddress	←Actif

CONT

SetR	←Actif
RValue	←$\mathcal{O}$
SelectLUAddress	←1
ChangeLUAddress	←Actif

CBT \$

BranchCondition	←*DigitIsReady*

CONT

GetNextDigit	←Actif

CONT

CBF \$ + 2

BranchCondition	←*DigitIsZero*

CALL *ADD*

SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif

CBF \$ + 3

BranchCondition	←*IsLastDigit*

CALL *DOUBLE*

SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif

BR \$ − 7

RET

$FILL_{LU}$:**CONT**

DefinedLUAdress	←0
SelectLUAddress	←2
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif
ChangeLUAddress	←Actif
RxOe	←Actif
LUWr	←Actif

CONT

SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif
RyOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif

CALL *DOUBLE*

AccOe	$\leftarrow$Actif
AccLoad	$\leftarrow$Actif
DefinedLUAdress	$\leftarrow 2^{w-2}-1$
SelectLUAddress	$\leftarrow 2$
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
ChangeLUAddress	$\leftarrow$Actif

PSHLDCT $2^{w-2}-3$

RxOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

RyOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif
IncLUCounter	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif

CALL *ADD*

DefinedLUAdress	$\leftarrow 0$
SelectLUAddress	$\leftarrow 2$
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
ChangeLUAddress	$\leftarrow$Actif

CONT

SelectLUAddress	$\leftarrow 3$
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
ChangeLUAddress	$\leftarrow$Actif

CONT

SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
RxOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif

CONT

SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif
RyOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif

LP $\$+3$

IncLUCounter	$\leftarrow$Actif
SelectLUAddress	$\leftarrow 3$
ChangeLUAddress	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

POPCT

RET

CALL *ADD*

DefinedLUAdress	$\leftarrow 2^{w-2}-1$
SelectLUAddress	$\leftarrow 2$
ChangeLUAddress	$\leftarrow$Actif

CONT

SelectLUAddress	←3
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif
ChangeLUAddress	←Actif

BR $\$ - 6$

SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif
RxOe	←Actif
LUWr	←Actif

ADD :**CBF** $\$ + 2$

AccOe	←Actif
AccLoad	←Actif
BranchCondition	←*LUIsInf*
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif

RET

CBF $\$ + 4$

BranchCondition	←*RIsInf*

CONT

SelectLUCoordinate	←y
ChangeLUCoordinate	←Actif
LUOe	←Actif
RxLoad	←Actif

CONT

SetR	←Actif
RValue	←0
SelectLUCoordinate	←y
ChangeLUCoordinate	←Actif

RET

LuOe	←Actif
RyLoad	←Actif

CONT

LuOe	←Actif
AccLoad	←Actif

CONT

RxOe	←Actif
AccLoad	←Actif

CONT

CBT $\$ + 7$

BranchCondition	←*AccIsZero*
AccOe	←Actif
AccLoad	←Actif

CONT

RyOe	←Actif
AccLoad	←Actif
SelectLUCoordinate	←y
ChangeLUCoordinate	←Actif

CONT

CONT

LuOe	←Actif
AccLoad	←Actif

CONT

CBF *DOUBLE*

BranchCondition	←*AccIsZero*
AccOe	←Actif
AccLoad	←Actif

RET

SetR	←Actif
RValue	←$\mathcal{O}$

CONT

AccLoad	←Actif
RxOe	←Actif
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif

CONT

LUOe	←Actif
AccLoad	←Actif

CONT

AccOe	←Actif
AccLoad	←Actif
LoadInvPol	←Actif
StartInvPol	←Actif
SelectLUCoordinate	←y
ChangeLUCoordinate	←Actif

CONT

BranchCondition	←0
AccLoad	←Actif
RyOe	←Actif
SelectLUCoordinate	←y
ChangeLUCoordinate	←Actif

CONT

LUOe	←Actif
AccLoad	←Actif
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif

CBF $\$ + 2$

BranchCondition	←*DigitSign*
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif

CONT

LUOe	←Actif
AccLoad	←Actif

CONT

AccOe	←Actif
AccLoad	←Actif
LoadMulPol	←2

CBT $\$$

BranchCondition	←*InvPolReady*

CONT

InvPolOe	←Actif
StartMulPol	←Actif
LoadMulPol	←1

CBT $\$$

BranchCondition	←*MulPolReady*

CONT

MulPolOe	←Actif
LoadSqrtPol	←Actif
StartSqrtPol	←Actif
AccLoad	←Actif

CONT

AccOe	←Actif
LoadMulPol	←2
IncAcc	←Actif
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif

CONT

RxOe	←Actif
AccLoad	←Actif
SelectLUCoordinate	←x
ChangeLUCoordinate	←Actif

CONT

LUOe	←Actif
AccLoad	←Actif

CBT \$

BranchCondition	←*SqrtPolReady*

CONT

SqrtPolOe	←Actif
AccLoad	←Actif

CONT

RxOe	←Actif
AccLoad	←Actif

CONT

AccOe	←Actif
StartMulPol	←Actif
LoadMulPol	←1

CONT

RxOe	←Actif
AccLoad	←Actif

CONT

AccOe	←Actif
RxLoad	←Actif

CONT

RyOe	←Actif
AccLoad	←Actif

CBT \$

BranchCondition	←*MulPolReady*

CONT

MulPolOe	←Actif
AccLoad	←Actif

RET

AccOe	←Actif
AccLoad	←Actif
RyLoad	←Actif

DOUBLE :**CBF** \$ + 2

BranchCondition	←*RIsInf*

RET

CONT

LoadInvPol	←Actif
StartInvPol	←Actif
RxOe	←Actif

CONT

AccLoad	←Actif
RxOe	←Actif

CONT

RyOe ←Actif
LoadMulPol ←2

CBT $

BranchCondition ←*InvPolReady*

CONT

InvPolOe ←Actif
StartMulPol ←Actif
LoadMulPol ←1

CBT $

BranchCondition ←*MulPolReady*

CONT

MulPolOe ←Actif
AccLoad ←Actif

CONT

LoadSqrtPol ←Actif
StartSqrtPol ←Actif
AccOe ←Actif

CONT

AccOe ←Actif
LoadMulPol ←2
IncAcc ←Actif

CBT $

BranchCondition ←*SqrtPolReady*

CONT

SqrtPolOe ←Actif
AccLoad ←Actif

CONT

RxOe ←Actif
AccLoad ←Actif

CONT

AccOe ←Actif
StartMulPol ←Actif
LoadMulPol ←1

CONT

RxOe ←Actif
AccLoad ←Actif

CONT

AccOe ←Actif
RxLoad ←Actif

CONT

RyOe ←Actif
AccLoad ←Actif

CBT $

BranchCondition ←*MulPolReady*

CONT

MulPolOe ←Actif
AccLoad ←Actif

RET

AccOe ←Actif
AccLoad ←Actif
RyLoad ←Actif

B.2 Séquenceur pipeliné

La seconde version du microcode est conçue pour un séquenceur pipeliné. Ce type de séquenceur devrait être utilisé si toutes les mémoires statiques du FPGA visé possèdent un point de synchronisation sur le bus d'adresses. Plusieurs microinstructions excédentaires ont du être insérées afin de pouvoir prendre un branchement une microinstruction à l'avance. 133 microinstructions sont nécessaires, imposant ainsi l'ajout d'un bit au bus d'adresses par rapport à la version non-pipelinée.

CONT
DefinedLUAdress ←0
SelectLUAddress ←1
SelectLUCoordinate ←x
ChangeLUCoordinate ←Actif
ChangeLUAddress ←Actif

START :**CONT**
NewRand ←Actif
InitX ←Actif
Inity ←Actif

CALL MUL_R

CONT

CBT $
BranchCondition ←*SetX*
Ready ←Actif

CONT
Ready ←Actif

CONT
BusOe ←Actif
RxLoad ←Actif
Ready ←Actif

CBT $
BranchCondition ←*SetY*
Ready ←Actif

CONT
Ready ←Actif

CALL MUL_R
BusOe ←Actif
RyLoad ←Actif

CONT

CBT $
BranchCondition ←*NewExchange*
Ready ←Actif

CONT
Ready ←Actif

BR *START*

CONT

MUL_R :**CONT**

CALL $FILL_{LU}$

DefinedLUAdress	$\leftarrow 0$
SelectLUAddress	$\leftarrow 2$
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
ChangeLUAddress	$\leftarrow$Actif

CONT

CONT

SetR	$\leftarrow$Actif
RValue	$\leftarrow \mathcal{O}$
SelectLUAddress	$\leftarrow 1$
ChangeLUAddress	$\leftarrow$Actif

CBT \$

BranchCondition	$\leftarrow$*DigitIsReady*

CONT

CONT

GetNextDigit	$\leftarrow$Actif

CONT

CBF $\$ + 4$

BranchCondition	$\leftarrow$*DigitIsZero*

CONT

CALL *ADD*

SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

CBF $\$ + 6$

BranchCondition	$\leftarrow$*IsLastDigit*

CONT

CALL *DOUBLE*

SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

BR $\$ - 12$

CONT

RET

CONT

$FILL_{LU}$:**CONT**

DefinedLUAdress	$\leftarrow 0$
SelectLUAddress	$\leftarrow 2$
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
ChangeLUAddress	$\leftarrow$Actif
RxOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif

CALL *DOUBLE*

SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif
RyOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif

CONT

AccOe	$\leftarrow$Actif
AccLoad	$\leftarrow$Actif
DefinedLUAdress	$\leftarrow 2^{w-2}-1$
SelectLUAddress	$\leftarrow 2$
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
ChangeLUAddress	$\leftarrow$Actif

LDCT $2^{w-2}-3$

RxOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

CALL *ADD*

RyOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif
IncLUCounter	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

DefinedLUAdress	$\leftarrow 0$
SelectLUAddress	$\leftarrow 2$
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
ChangeLUAddress	$\leftarrow$Actif

CONT

SelectLUAddress	$\leftarrow 3$
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
ChangeLUAddress	$\leftarrow$Actif

CONT

SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
RxOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif

LP $\$+4$

SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif
RyOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif

CONT

IncLUCounter	$\leftarrow$Actif
SelectLUAddress	$\leftarrow 3$
ChangeLUAddress	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

RET

CONT

CONT

CALL *ADD*

DefinedLUAdress	$\leftarrow 2^{w-2}-1$
SelectLUAddress	$\leftarrow 2$
ChangeLUAddress	$\leftarrow$Actif

CONT

BR $\$-7$

SelectLUAddress	$\leftarrow 3$
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
ChangeLUAddress	$\leftarrow$Actif

CONT

SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif
RxOe	$\leftarrow$Actif
LUWr	$\leftarrow$Actif

ADD :**CBF** $\$ + 4$

AccOe	$\leftarrow$Actif
AccLoad	$\leftarrow$Actif
BranchCondition	$\leftarrow$*LUIsInf*
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

RET

CONT

CBF $\$ + 5$

BranchCondition	$\leftarrow$*RIsInf*

CONT

CONT

SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif
LUOe	$\leftarrow '1'$
RxLoad	$\leftarrow$Actif

RET

SetR	$\leftarrow$Actif
RValue	$\leftarrow 0$
SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

LUOe	$\leftarrow '1'$
RyLoad	$\leftarrow$Actif

CONT

LUOe	$\leftarrow '1'$
AccLoad	$\leftarrow$Actif

CONT

RxOe	$\leftarrow$Actif
AccLoad	$\leftarrow$Actif

CONT

CBT $\$ + 10$

BranchCondition	$\leftarrow$*AccIsZero*
AccOe	$\leftarrow$Actif
AccLoad	$\leftarrow$Actif

CONT

CONT

RyOe	$\leftarrow$Actif
AccLoad	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

CONT

LUOe	$\leftarrow '1'$
AccLoad	$\leftarrow$Actif

CONT

CBF *DOUBLE*

BranchCondition	$\leftarrow$*AccIsZero*
AccOe	$\leftarrow$Actif
AccLoad	$\leftarrow$Actif

CONT

RET

SetR	$\leftarrow$Actif
RValue	$\leftarrow\mathcal{O}$

CONT

CONT

AccLoad	$\leftarrow$Actif
RxOe	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

LUOe	$\leftarrow'1'$
AccLoad	$\leftarrow$Actif

CONT

AccOe	$\leftarrow$Actif
AccLoad	$\leftarrow$Actif
LoadInvPol	$\leftarrow$Actif
StartInvPol	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

BranchCondition	$\leftarrow$0
AccLoad	$\leftarrow$Actif
RyOe	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow y$
ChangeLUCoordinate	$\leftarrow$Actif

CBF $\$+3$

BranchCondition	$\leftarrow$*DigitSign*
LUOe	$\leftarrow'1'$
AccLoad	$\leftarrow$Actif
SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

SelectLUCoordinate	$\leftarrow x$
ChangeLUCoordinate	$\leftarrow$Actif

CONT

LUOe	$\leftarrow'1'$
AccLoad	$\leftarrow$Actif

CONT

AccOe	$\leftarrow$Actif
AccLoad	$\leftarrow$Actif
LoadMulPol	$\leftarrow$2

CBT $

BranchCondition	$\leftarrow$*InvPolReady*

CONT

CONT

InvPolOe	$\leftarrow$Actif
StartMulPol	$\leftarrow$Actif
LoadMulPol	$\leftarrow$1

CBT $

BranchCondition	$\leftarrow$*MulPolReady*

CONT

CONT

MulPolOe ←Actif
LoadSqrtPol ←Actif
StartSqrtPol ←Actif
AccLoad ←Actif

CONT

AccOe ←Actif
LoadMulPol ←2
IncAcc ←Actif
SelectLUCoordinate ←x
ChangeLUCoordinate ←Actif

CONT

RxOe ←Actif
AccLoad ←Actif
SelectLUCoordinate ←x
ChangeLUCoordinate ←Actif

CONT

LUOe ←$'1'$
AccLoad ←Actif

CBT \$

BranchCondition ←*SqrtPolReady*

CONT

CONT

SqrtlPolOe ←Actif
AccLoad ←Actif

CONT

RxOe ←Actif
AccLoad ←Actif

CONT

AccOe ←Actif
StartMulPol ←Actif
LoadMulPol ←1

CONT

RxOe ←Actif
AccLoad ←Actif

CONT

AccOe ←Actif
RxLoad ←Actif

CONT

RyOe ←Actif
AccLoad ←Actif

CBT \$

BranchCondition ←*MulPolReady*

CONT

RET

MulPolOe ←Actif
AccLoad ←Actif

CONT

AccOe ←Actif
AccLoad ←Actif
RyLoad ←Actif

DOUBLE :**CBF** \$ + 4

BranchCondition ←*RIsInf*

CONT

RET

CONT

CONT

LoadInvPol ←Actif
StartInvPol ←Actif
RxOe ←Actif

CONT

AccLoad ←Actif
RxOe ←Actif

CONT

RyOe ←Actif
LoadMulPol ←2

CBT $

BranchCondition ←*InvPolReady*

CONT

CONT

InvPolOe ←Actif
StartMulPol ←Actif
LoadMulPol ←1

CBT $

BranchCondition ←*MulPolReady*

CONT

CONT

MulPolOe ←Actif
AccLoad ←Actif

CONT

LoadSqrtPol ←Actif
StartSqrtPol ←Actif
AccOe ←Actif

CONT

AccOe ←Actif
LoadMulPol ←2
IncAcc ←Actif

CBT $

BranchCondition ←*SqrtPoReady*

CONT

CONT

SqrtlPolOe ←Actif
AccLoad ←Actif

CONT

RxOe ←Actif
AccLoad ←Actif

CONT

AccOe ←Actif
StartMulPol ←Actif
LoadMulPol ←1

CONT

RxOe ←Actif
AccLoad ←Actif

CONT

AccOe	←Actif
RxLoad	←Actif

CONT

RyOe	←Actif
AccLoad	←Actif

CBT $

BranchCondition	←*MulPolReady*

CONT

RET

MulPolOe	←Actif
AccLoad	←Actif

CONT

AccOe	←Actif
AccLoad	←Actif
RyLoad	←Actif

Index