République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

ÉCOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

DÉPARTEMENT D'ÉLECTRONIQUE

Projet de fin d'études
En vue de l'obtention du diplôme de Master en Électronique

Thème :

Étude et implémentation sur circuit reconfigurable de l'algorithme d'Euclid pour le décodage de Reed-Solomon

Réalisé par :

Mr **KHEMIS Redouane**

Soutenu publiquement le **25/06/2013** devant le jury composé de :

R. SADOUN	Docteur	ENP	**Président**
M. TAGHI	Chargé de Cours	ENP	**Promoteur**
L. ABDELOUEL	Chargé de Cours	ENP	**Co-Promoteur**
M. MAMERI	Chargé de Cours	ENP	**Examinateur**

Promotion : Juin 2013

Résumé

Ce travail s'inscrit dans le cadre de l'étude des codes correcteurs d'erreurs. Je m'intéresse tout particulièrement au bloc d'Euclid utilisé dans le décodage de Reed-Solomon.

Après une étude théorique de ces codes, je propose une architectures pour le décodeurs de Reed-Solomon $(15, 9)$ et $(255, 239)$ basée sur la théorie d'Euclid. Les codes de description sont écrits en VHDL, la synthèse est l'implémentation de ce bloc est réalisé à l'aide de l'outil ISE de Xilinx sur carte FPGA.
Mots clés : Euclid, codes de Reed-Solomon, FPGA, code correcteurs d'erreurs.

Abstract

This work lies within the scope of the studies of the errors correction codes. We are interested particularly to studies the Euclid block used in the decoders of Reed-Solomon. After theoretical study of these codes, I propose a design for the Reed-Solomon encoders and decoders $(15, 9)$ and $(255, 239)$ based on the Euclid theory. The codes of description are written in VHDL, the synthesis and the implementation of this block is carried out using the tool ISE of Xilinx on FPGA board.
Key words :Euclid algorithm , Reed-Solomon codes, Channel coding, FPGA, errors correction codes.

ملخص:

يندرج هذا العمل في إطار دراسة الشفرات المصححة للأخطاء بالخصوص شفرات ريد سولومون بعد القام بدراسة نظرية حول هذه الشفرات أقترح تصميم للشفرات ومفككات التشفير ريد سولومون 15,9 و293,255 تعمد على نظرية إقليدس.

تمت كتابة البرنامج باللغة VHDL فيما تم إنجاز الحوصلة والتثبت لهذه الشفرة بواسطة الأداة ise لـ Xilinx على FPGA

الكلمات الافتتاحية : نظرية إقليدس شفرة ريد سولومون FPGA الشفرات المصححة للأخطاء.

Remerciements

Je remercie mes parents qui m'ont beaucoup encouragés le long de ce projet. Et à l'issu de mes études faites au sein de l'École Nationale Polytechnique je voudrais rendre un hommage tout particulier à : mes promoteurs Mr. **M.Taghi** et Mr. **L.Abdelouel** qui m'ont honoré en acceptant de m'encadrer. Tous **les enseignants** du département **d'électronique** de l'Ecole Nationale Polytechnique.

Mes remerciements à Mr. **R.SADOUN** et **M.MAMERI** d'avoir accepté de juger mon modeste travail.

Je remercie tous ceux qui ont participé de prés ou de loin à l'aboutissement de nos études.

Dédicace

À la mémoire de ma grande mère.

À mes parents,

À mes frères,

À mes sœurs,

À toutes ma famille,

À mes amis,

À toute l'équipe $POPAS$,

À toute l'équipe MW,

Je dédie ce travail.

KHEMIS Redouane

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Introduction

L'histoire des transmissions a été marquée par plusieurs évolutions, des pigeons voyageurs aux communications sans fils. Actuellement, nous assistons à une forte augmentation des besoins en termes de communication numérique. La communication à distance (téléphone, internet, satellite, $\cdots$) entre machine et usagers nécessite les canaux de transmission pour l'acheminement des informations. Les canaux utilisés sans en général loin d'être parfaits, cela peut entraîner une modification du message émis. L'imprévisibilité du message émis par la source impose alors au récepteur l'utilisation de techniques lui permettant de vérifier à la fois l'exactitude et la certitude de l'information reçue.

Problématique

Sans un souci de fiabilités, tous les efforts d'amélioration dés débits de transmission seraient vains car cela impliquerait que certaines données doivent être retransmises. C'est dans la course au débit que les codes correcteurs d'erreurs entrent en jeu.

Un code correcteur d'erreurs est un code qui permet de corriger une ou plusieurs erreurs en ajoutant au mot d'information des symboles de contrôle.Plusieurs codes existent mais dans ce projet de fin d'étude nous intéressons aux codes de Reed-Solomon utilisés dans le domaine des communications sans fils pour le meilleur compromis efficacité et complexité qu'ils présentent.

Objectifs du projet

Ce travail s'inscrit dans le cadre des activités du Laboratoire des Dispositifs de Communications et de Conversions Photovoltaïques (LDCCP) de l'ENP. Il vise à explorer les possibilités qu'offrent les circuits programmables (FPGA) pour l'implémentation du codeur et décodeur de Reed-Solomon.

Plus spécifiquement, les objectifs poursuivis pour ce projet de fin d'étude sont :

- Etude des principes des codes de Reed-Solomon,
- Proposition une architecture pour l'implémentation de bloc d'Euclid utilisé pour la résolution de l'equation clé de décodage de Reed-Solomon,
- Simulation de bloc d'Euclid utilisé dans le décodeur de Reed-Solomon RS(15,9), RS(255,239) sous ISE de Xilinx,
- Implémentation de codeur et décodeur de Reed-Solomon RS$(15, 9)$sur circuit reconfigurable $\ll$ FPGA$\gg$.

Organisation du mémoire

Ce mémoire est organisé en deux chapitres :

- Dans le premier chapitre, j'explique la théorie de codage et de décodage des codes de Reed-Solomon en éclaircissant l'algorithme d'Euclid et ces versions modifiées utilisés.
- Au chapitre 2, nous présentons les résultats de l'implémentation de bloc d'Euclid sur les circuits reconfigurables FPGA en donnant les ressources hardware utilisées ainsi que les performances temporelles de ce bloc.
- En conclusion, nous résumons les objectifs réalisés et ensuite les éventuelles perspectives.

Chapitre 1

Codes de Reed-Solomon

1.1 Introduction

Les codes de Reed Solomon(RS) constituent une famille de codes d'une importance exceptionnelle, tant du point de vue de la théorie que des applications. Récemment, ces codes ont été largement utilisés dans de nombreuses applications telles que la communication par satellite, la diffusion de vidéo numérique (digital video broadcast) et les communications mobiles sans fils. Ces codes, qui fonctionnent en bloc, sont basés mathématiquement sur les champs finis de Galois. Les codes RS permettent de corriger les erreurs et les effacements grâce à des symboles de contrôle ajoutés après l'information.

Je commence ce chapitre par la présentation des corps de Galois, puis nous détaillons les codes de Reed-Solomon et leurs théorie de codage et décodage.

1.2 Corps de Galois

Les « champs de Galois » font partie d'une branche particulière des mathématiques qui modélise les fonctions du monde numérique. Ils sont très utilisés dans la cryptographie ainsi que pour la reconstruction des données.

Il y a deux types de corps, les corps finis et les champs infinis. Les « corps de Galois » finis sont des ensembles d'éléments fermés sur eux-mêmes. L'addition et la multiplication de deux éléments du corps donnent toujours un élément du corps fini.

1.2.1 Éléments des champs de Galois

Un « champ de Galois » consiste en un ensemble de nombres, ces nombres sont constitués à l'aide de l'élément base α comme suit :

$$0, \alpha, \alpha^2, \cdots, \alpha^{(n-1)}$$

En prenant $n = 2^m - 1$, on forme un ensemble de 2^m éléments. Le champ est alors noté $GF(2^m)$.
$GF(2^m)$ est formé à partir du champ de base $GF(2)$ et contiendra des multiples des éléments simples de $GF(2)$.
En additionnant les puissances de α, chaque élément du champ peut être représenté par une expression polynomiale du type :

$$\alpha^{m-1}x^{m-1} + \alpha^{m-2}x^{m-2} + \cdots + \alpha x + \alpha^0$$

Avec : $\alpha^{m-1}, \cdots, \alpha^0$: éléments bases du $GF(2)$ (valeurs : 0,1).

1.2.2 Opérations algébriques

Addition

L'addition dans un champ fini $GF(2)$ correspond à faire une addition modulo 2, donc l'addition de tous les éléments d'un « champ de Galois » dérivés du champ de base sera une addition modulo 2.

Pour effectuer une addition sur le corps $GF(2^m)$, il faut voir que l'on additionne en fait deux vecteurs, et donc que l'on effectue l'addition composante par composante.

Soustraction

Une soustraction dans $GF(2^m)$ correspond à faire une addition dans le même corps.

Multiplication

La multiplication dans le « champ de Galois » peut être réalisé suivant deux techniques la première consiste à utiliser une table de vérité, la deuxième consiste à faire la multiplication entre deux polynômes et ensuite de normaliser le résultat par rapport au polynôme primitif du champ choisi. Cette dernière est utilisée dans ce rapport.

1.2.3 Polynôme irréductible

Soit P un polynôme à coefficients dans K. Si P est irréductible, alors P ne s'annule pas sur K. Par contre on n'a pas la réciproque.

1.2.4 Polynômes primitifs

Ce polynôme permet de construire le « corps de Galois » souhaité. Tous les éléments non nuls du corps peuvent être construits en utilisant l'élément α comme racine du polynôme primitif. Chaque m peut avoir plusieurs polynômes primitifs.
On mentionne dans le tableau1.1 ci-dessous, Les polynômes primitifs pour les principaux « corps de Galois » :

1.2.5 Exemple de construction d'un corps de Galois d'un $GF(2^4)$

On veut construire tous les éléments du $GF(2^4)$ à partir du polynôme primitif :

$$p(x) = x^4 + x + 1$$

On a α est racine du polynôme primitif. Donc :

$$0 = \alpha^4 + \alpha + 1$$
$$\alpha^4 = \alpha + 1$$

m	P(X)	m	P(X)
3	$1+X+X^3$	14	$1+X+X^6+X^{10}+X^{14}$
4	$1+X+X^4$	15	$1+X+X^{15}$
5	$1+X^2+X^5$	16	$1+X+X^6+X^1+X^{16}$
6	$1+X+X^6$	17	$1+X^3+X^{17}$
7	$1+X^3+X^7$	18	$1+X^7+X^{18}$
8	$1+X^2+X^3+X^4+X^8$	19	$1+X+X^2+X^5+X^{19}$
9	$1+X^4+X^9$	20	$1+X^3+X^{20}$
10	$1+X^3+X^{10}$	21	$1+X^2+X^{21}$
11	$1+X^2+X^{11}$	22	$1+X+X^{22}$
12	$1+X+X^4+X^6+X^{12}$	23	$1+X^5+X^{23}$
13	$1+X+X^3+X^4+X^{13}$	24	$1+X+X^2+X^7+X^{24}$

Tableau 1.1 – polynômes primitifs dans $GF(2^m)$.

Maintenant, il suffit de multiplier l'élément α par α à chaque étape et réduire par rapport à $\alpha^4 = \alpha+1$ pour obtenir le champ complet. On aura besoin de 13 multiplications pour compléter ce corps.

Les éléments d'un « champ de Galois » de $GF(2^4)$ sont représentés dans le tableau1.2 :

Éléments	Formes polynômiales	Formes binaires	Formes décimales
0	0	0000	0
1	1	0001	1
α	α	0010	2
α^2	α^2	0100	4
α^3	α^2	1000	8
α^4	$\alpha+1$	0011	3
α^5	$\alpha^3+\alpha^2$	0110	6
α^6	$\alpha^3+\alpha^2$	1100	12
α^7	$\alpha^3+\alpha+1$	1011	11
α^8	α^2+1	0101	5
α^9	$\alpha^3+\alpha$	1010	10
α^{10}	$\alpha^2+\alpha+1$	0111	7
α^{11}	$\alpha^3+\alpha^2+\alpha$	1110	14
α^{12}	$\alpha^3+\alpha^2+\alpha+1$	1111	15
α^{13}	$\alpha^3+\alpha^2+1$	1101	13
α^{14}	α^3+1	1001	9

Tableau 1.2 – éléments de $GF(2^4)$.

1.2.6 Dérivation formelles d'un polynôme dans un corps de Galois

La dérivée formelle du polynôme dans un « champ de Galois » $GF(2^m)$ est définie avec les puissances paires car les puissances impaires sont précédées de coefficients pairs qui

annulent les termes. Un polynôme d'ordre m est défini comme :

$$f(x) = f_m\, x^m \,+\, f_{m-1}\, x^{m-1} \,+\, \cdots \,+\, f_1\, x \,+\, f_0$$

La dérivée formelle de ce polynôme est définie comme :

$$f\prime = f_1 \,+\, 2f_2\; x \,+\, 3\, f_3 x^2 \,+\, \cdots \,+\, m\, f_m\, x^{m-1}$$

$$f\prime = f_1 \,+\, 3f_3 x^2 \,+\, 5f_5 x^4 \,+\, \cdots$$

1.3 Principe des codes de Reed-Solomon

1.3.1 Définition

Les codes de Reed–Solomon[1] sont des codes de détection et de correction des erreurs basés sur les corps de Galois. Ce sont des codes particuliers des codes BCH.

1.3.2 Propriétés des codes Reed-Solomon

Le codeur prend k symboles de donnée (chaque symbole contenant s bits) et calcul les informations de contrôle pour construire n symboles, ce qui donne $n - k$ symboles de contrôle. Le décodeur peut corriger au maximum t symboles, où $2t = n - k$. La longueur maximale d'un code de Reed–Solomon est définie comme :

$$n = k + 2t = 2^m - 1 \tag{1.1}$$

Avec

k : nombre de symboles d'information.

2t : nombre de symboles de contrôle.

m : nombre de bits par symbole.

La distance minimale d'un code Reed–Solomon est :$d_{min} = 2t - 1$.

Ces codes présentent les propriétés suivantes :

- Ils sont linéaires.

- Ils sont cycliques, c'est-à-dire, que chaque mot-code décalé engendre un autre mot-code. Tous les codes cycliques peuvent être réduits en gardant la même capacité d'erreur, mais le nouveau code formé n'est alors pas cyclique.
- Ils sont des codes non-binaires. Les codes sont représentés sur des « champs de Galois » de $GF(2^m)$ et non pas sur des champs de $GF(2)$.
- Les symboles sont définis comme les coefficients du polynôme et le degré de x indique l'ordre. Ainsi, le symbole avec l'ordre le plus élevé est reçu/envoyé en premier et le dernier symbole reçu/envoyé est celui dont l'ordre est moindre.

1.3.3 Exemple de nombres de symboles corrigeables

Prenons un code de Reed-Solomon, par exemple $RS(n, k) = RS(15, 9)$. L'objectif est de découvrir combien de bits sont utilisés pour chaque symbole et combien d'erreurs peut-on corriger. n : indique la longueur totale d'un bloc de Reed – Solomon, 15 symboles dans ce cas et k indique la longueur du bloc d'information, 9 symboles dans cet exemple.

La capacité de correction des erreurs du système est :

$$2t = n - k = 15 - 9 = 6$$

Donc

$$t = (n - k)/2 = 3$$

Ce code permettra de corriger 3 symboles. Le nombre de bits s par symbole est :

$$n = 2^m - 1.$$

Donc

$$\begin{aligned} m &= \ln(n+1)/\ln 2 \\ &= \ln(16)/\ln 2 \\ &= 4 \end{aligned}$$

Le nombre de bits utilisés pour coder les symboles est donc de 4. Ce qui nous amène à utiliser un « corps de Galois » de $GF(2^4)$.

1.3.4 Codeur de Reed Solomon

Théories du codage

L'équation clé définissant le codage[1] systématique de Reed–Solomon (n, k) est :

$$c(x) = i(x)x^{n-k} + |i(x)x^{n-k}|mod(g(x)) \tag{1.2}$$

Avec :

$c(x)$: Polynôme du mot-code, degré $n-k$.

i(x) : Polynôme d'information, degré $n-k$.

$|i(x)x^{n-k}|mod(g(x))$: Polynôme de contrôle, degré $n-k-1$.

$g(x)$: Polynôme générateur, degré $n-k$.

Le codage systématique signifie que l'information est codée dans le degré élevé du mot code et que les symboles de contrôle sont introduits après les mots d'information.

Polynôme générateur

Le polynôme générateur sert à générer les symboles de contrôle. Tous les codes Reed-Solomon sont valables si et seulement si ils sont divisibles par leur polynôme générateur, $c(x)$ doit être divisible par $g(x)$.

Pour générer un code correcteur d'erreurs de t symboles, on devrait avoir un polynôme générateur de puissance α^{2t}. La puissance maximale du polynôme est déterminée grâce la distance minimale qui est $d_{min} = 2t + 1$. On devrait avoir $2t + 1$ termes du polynôme générateur. Le polynôme générateur est sous la forme :

$$g(x) = (x - \alpha^1)(x - \alpha^2) \cdots (x - \alpha^{2t})$$

$$g(x) = g_{2t}x^{2t} + g_{2t-1}x^{2t-1} + \cdots + g_1x + g_0$$

Exemple de calcul des coefficients du polynôme générateur pour RS(15,9)

On a :

$$\begin{aligned}
g(x) &= (x-\alpha^1)(x-\alpha^2)\cdots(x-\alpha^{2t}) \\
g(x) &= (x-\alpha^1)(x-\alpha^2)\cdots(x-\alpha^6) \\
&= (x^2+\alpha^5 x+\alpha^3)(x^2+\alpha^7 x+\alpha^7) \\
&= (x^4+\alpha^{13}x^3+\alpha^6 x^2+\alpha^3 x+\alpha^{10})(x^2+\alpha^9 x+\alpha^{11}) \\
&= x^6+x^5(\alpha^{13}+\alpha^9)+x^4(\alpha^6+\alpha^7+\alpha^{11}) \\
&\quad +x^3(\alpha^3+\alpha^9+1)+x^2(\alpha^{10}+\alpha^{12}+\alpha^2) \\
&\quad +x(\alpha^4+\alpha^{14})+\alpha^6 \\
&= x^6+\alpha^{10}x^5+\alpha^{14}x^4+\alpha^4 x^3+\alpha^6 x^2+\alpha^9 x+\alpha^6
\end{aligned}$$

En prenant l'équivalent en décimale, on trouve :

$$g(x) = x^6+7x^5+9x^4+3x^3+12x^2+10x+12$$

1.3.5 Décodeur de Reed-Solomon

Les étapes nécessaires pour le décodage des codes de Reed-Solomon sont :

- Calcul du syndrome.
- Calcul des polynômes d'amplitude et de localisation des erreurs.
- Calcul des racines et évaluation des deux polynômes (de localisation des erreurs et d'amplitude des erreurs).
- Sommation du polynôme constitué et du polynôme reçu pour reconstituer l'information de départ sans erreur.

Calcul du syndrome

Considérons un code de Reed-Solomon $c(x)$ correspondant au code transmis et soit $r(x)$ le code que l'on reçoit. Le polynome d'erreurs introduit par le canal est définit comme :

$$e(x) \;=\; r(x)-c(x) \;=\; r(x)+c(x) \tag{1.3}$$

Le calcul du syndrome est défini comme le reste de la division entre le polynôme reçu r(x) et le polynôme générateur g(x). Le reste indiquera la présence d'erreurs. Il peut être aussi effectué par un processus itératif. On a :

$$r(x) \;=\; c(x) + e(x) \tag{1.4}$$

On obtient :

$$S_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) \tag{1.5}$$

Seuls les premiers $2t$ symboles du syndrome qui sont connus. Ainsi, le syndrome sous forme polynomiale sera :

$$S(x) \;=\; S_{2t}x^{2t-1} + \cdots + S_2x + S_1 \tag{1.6}$$

Si le code reçu $r(x)$ n'est pas affecté par des erreurs alors tous les coefficients du syndrome seront nuls ($r(x) = c(x)$).

Calcul des polynômes localisateur et d'amplitude

Le calcul des polynômes de localisation des erreurs et d'amplitude des erreurs est basé sur versions reformulées de l'algorithme d'Euclide.

Algorithme d'Euclide (EA)

L'algorithme d'Euclide[2] est un algorithme récursif qui permet de trouver le plus grand diviseur commun de deux polynômes $r_0(x)$ et $r_1(x)$ dans le « champ de Galois » $GF(q)$. Il existe deux polynômes $a(x)$ et $b(x)$ en $GF(q)$ tels que :

$$PGCD(r_0(x), r_1(x)) = a(x)r_0(x) + b(x)r_1(x). \tag{1.7}$$

Avec : $a(x)$ et$b(x)$ peuvent être calculés selon l'algorithme d'Euclide.
Le polynôme de localisation des erreurs est défini comme :

$$\sigma(x) = \prod_{k=1}^{v}(1 - \sigma^{i_k}x) = \sigma_v x^v + \sigma_{(v-1)}x^{v-1} + \cdots + \sigma_1 x + 1 \tag{1.8}$$

Le polynôme d'amplitude des erreurs est calculé de la façon suivante :

$$\omega(x) = S(x).\sigma(x) \tag{1.9}$$

tq :

$\sigma(x)$: le polynôme de localisation des erreurs.

$S(x)$: polynôme syndrome.

Comme on connait seulement $2t$ symboles de polynôme syndrome, on devrait limiter le polynôme d'amplitude à $2t$:

$$S(x)\sigma(x) \;=\; \omega(x)\, mod\, (x^{2t}) \tag{1.10}$$

Cette expression est l'équation clé pour les codes de Reed–Solomon. Cette équation clé peut être résolu selon le théorème d'Euclide en posant $r_0(x) = x^{2t}$ et $r_1 = S(x)$. Le calcul du théorème d'Euclide nous donnera comme solution **le polynôme de localisation des erreurs** et **le polynôme d'amplitude des erreurs**.

Si le nombre d'erreurs v dans le mot-code transmis $c(x)$ est plus petit ou égal à t, l'équation clé a une seule paire de solutions $\sigma(x)$ et $\omega(x)$. Les deux degrés des polynômes doivent respecter la contrainte qui suit :

$$\deg(\omega(x)) \;<\; \deg(\sigma(x)) \;\leq\; t. \tag{1.11}$$

L'organigramme de l'algorithme d'Euclide est donné dans la figure1.1 ci-dessous :

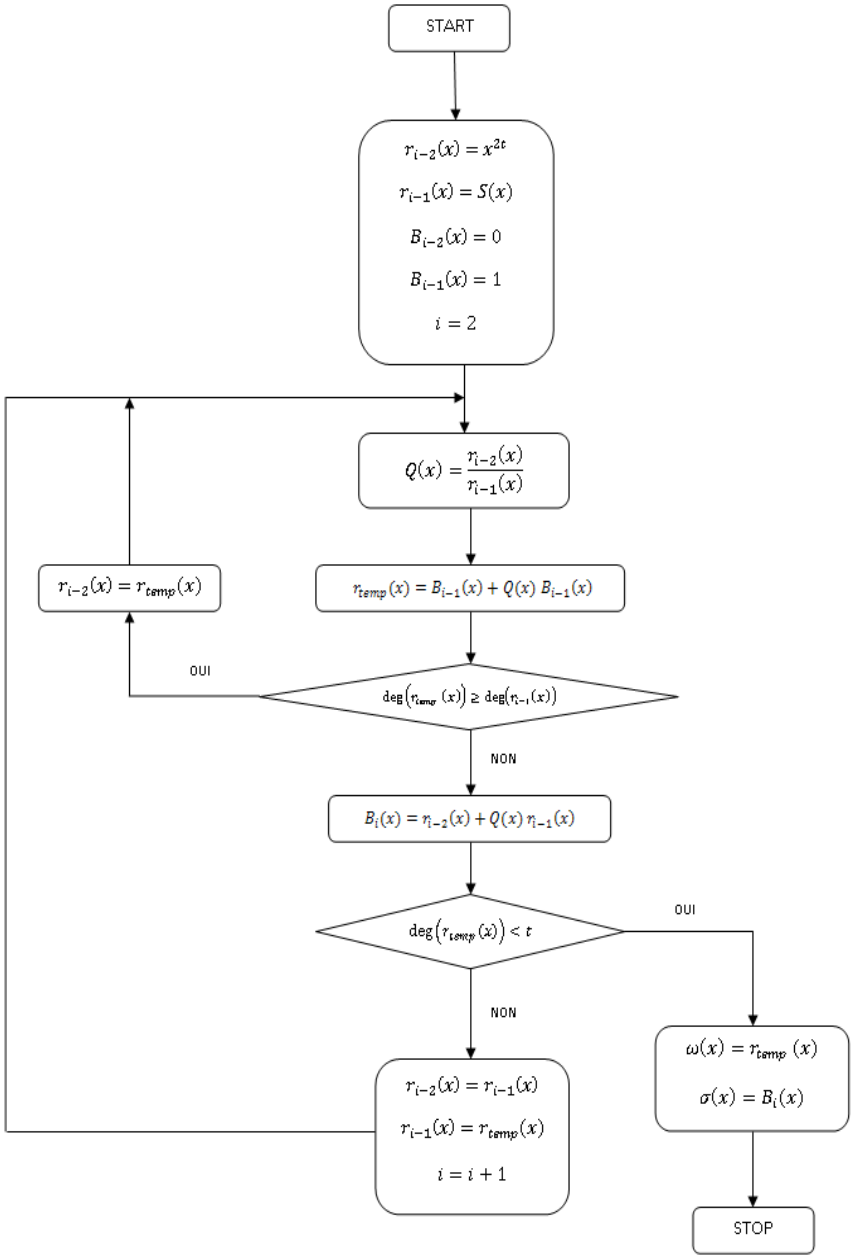

Figure 1.1 – L'organigramme de l'algorithme d'Euclide pour le calcul du polynôme de localisation et le polynôme d'amplitude.

Le dernier reste de la division nous donnera le polynôme d'amplitude. Le polynôme de localisation des erreurs est donné selon la relation :

$$\sigma_i(x) = \sigma_{i-2}(x) + \sigma_{i-1}(x)Q_i(x) \tag{1.12}$$

Avec :

$$\sigma_i(x) = B_i(x) \tag{1.13}$$

Algorithme d'Euclid étendu (EEA)

l'algorithme d'Euclid (EA)[3] calcul le polynôme d'amplitude, ensuite le polynôme de localisation des erreurs sera calculé par l'équation 1.12 ce qui augmente la latence. L'algorithme d'Euclid étendu calcul les deux polynômes d'amplitude et de localisation des erreurs simultanément, mais nécessite de faire la division dans les champs de Galois

Algorithme d'Euclide modifié (MEA)

L'algorithme d'Euclid et d'Euclid modifié sont des algorithmes récursifs qui nécessites des opérations de multiplication et de division polynomiales. La condition d'arrêt de l'algorithme d'Euclid est (deg(ω) ¡ t). Cela fait que la latence de cet algorithme change suivant le degré de polynôme du syndrome.

Des nouvelles modifications de l'algorithme d'Euclid sont proposées dans [4], [5]. Sugiyama et al. [6] a souligné que l'Algorithme d'Euclide Etendu (EEA) pour le calcul du plus grand diviseur commun de deux polynômes (PGCD) peut être utilisé pour résoudre l'équation clé.

Une version modifiée de l'algorithme EEA est présentée dans [7]. Cette version produit les polynômes :

$X(x) = \alpha x^{2t-v}\sigma(x)$: Polynôme de localisation des erreurs calculé avec l'algorithme MEA,

$V(x) = \alpha x^{2t-v}\omega(x)$: Polynôme d'amplitude des erreurs calculé avec l'algorithme MEA.

Avec

$\sigma(x)$: Polynôme de localisation des erreurs calculé avec l'algorithme d'Euclid,

$\omega(x)$: Polynôme d'amplitude des erreurs calculé avec l'algorithme d'Euclid,

v : Nombre d'erreur.

t : Capacité de correction.

Ces polynômes sont calculés de façon itérative sans la nécessité d'effectuer la division polynomiale. La latence de cet algorithme est égale à $2t$.

Les étapes de cet algorithme sont illustrées sur L'organigramme de la figure1.2.

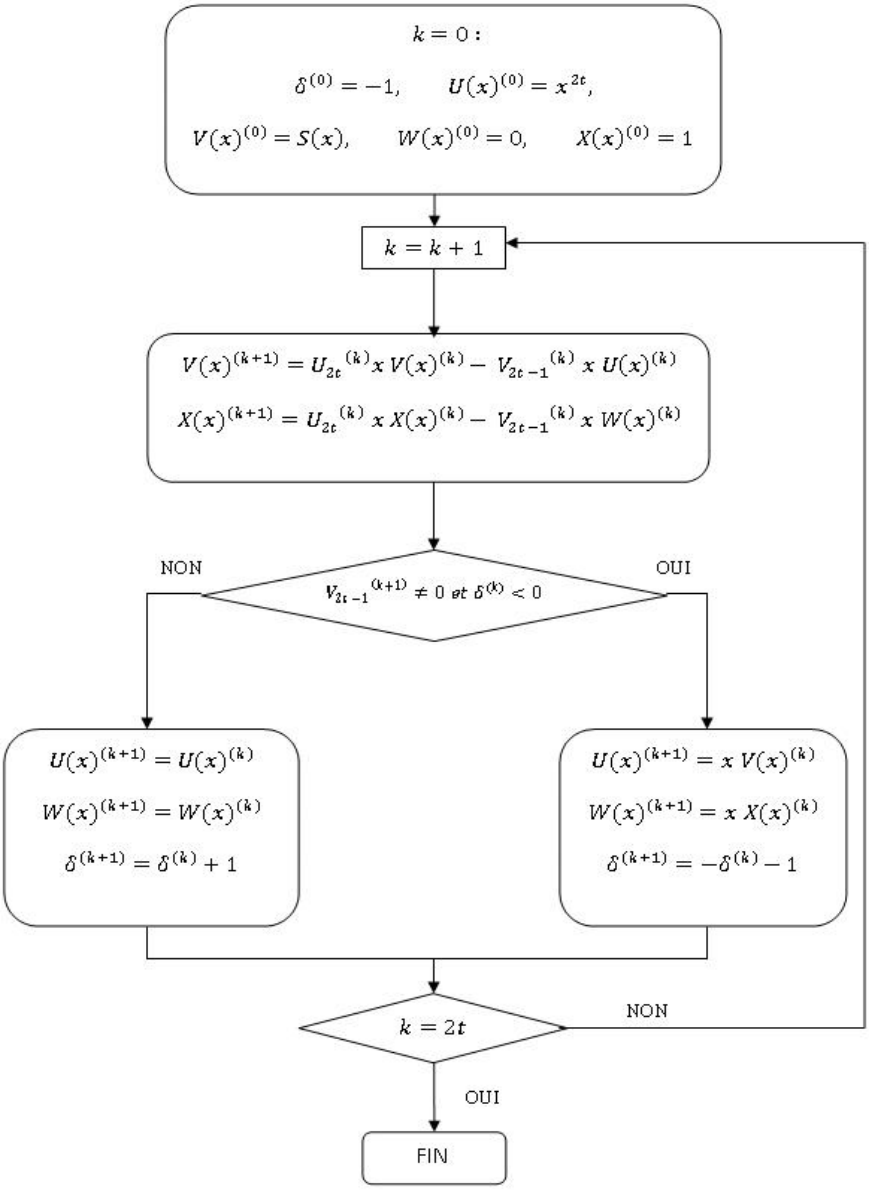

Figure 1.2 – L'organigramme de l'algorithme MEA pour le calcul du polynôme de localisation et le polynôme d'amplitude.

Quand l'algorithme se termine, si $v \leq t$ erreurs sont survenues, alors :

$$\delta = 2v - 2t - 1 < 0 \tag{1.14}$$

$$
\begin{aligned}
(X_{2t},\, X_{2t-1},\, \cdots, X_{2t-v}, X_{2t-v-1}, \cdots, X_0) &= (\beta\sigma_v, \beta\sigma_{v-1}, \cdots, \beta\sigma_0, 0, \cdots, 0) \\
(V_{2t}, V_{2t-1}, \cdots, V_{2t-v}, V_{2t-v-1}, \cdots, V_0) &= (0,\ \beta\omega_{v-1},\ \cdots, \beta\omega_0, 0, \cdots, 0)
\end{aligned}
$$

β est un élément de champ de Galois non nul. Si l'algorithme termine avec $\delta \geq 0$, alors le nombre d'erreur excède t et les polynômes calculés sont erronés.

Chien Search

Une fois le polynôme de localisation des erreurs calculé, on doit évaluer ses racines et sa dérivée.

L'évaluation des racines s'effectue avec l'algorithme appelé « Chien Search » qui évalue toutes les possibilités. Par exemple pour un $RS(15, 9)$, on évalue le polynôme de localisation des erreurs et sa dérivée pour tous les éléments du « corps de Galois » $GF(2^4)$, sauf pour l'élément nul.
Cet algorithme permet ainsi l'évaluation du polynôme d'amplitude.

Algorithme de Forney

Cet algorithme permet de construire le polynôme d'erreurs $e(x)$ à additionner avec le polynôme reçu $r(x)$ pour reconstituer le polynôme $c(x)$. Pour le calcul du polynôme d'erreurs $e(x)$, les polynomes $\sigma(\alpha^i)$, $\sigma\prime(\alpha^i)$ et $\omega(\alpha^i)$ sont nécessaires.
L'algorithme de Forney est défini comme :

$$e_i = \frac{\omega(\alpha^i)}{\sigma\prime(\alpha^i)} \tag{1.15}$$

Avec :

$\omega(\alpha^i)$: Polynôme d'amplitude évalué pour les valeurs de $GF(2^m)$.

$\sigma\prime(\alpha^i)$: Dérivée du polynôme de localisation des erreurs pour les valeurs de $GF(2^m)$.

1.4 Conclusion

J'ai exposé les bases mathématiques de la théorie de codage et décodage de Reed-Solomon, j'ai ensuite présenté l'algorithmes de décodage RS basé sur le théorème d'Euclid. L'implémentation de ce dernier sera développée dans le chapitre 2.

Chapitre 2

Implémentation de l'algorithme d'Euclid

Introduction

Dans ce chapitre, j'expose tout d'abord l'architecture de bloc d'Euclid utilisé pour l'implémentation et j'illustre ce bloc par un exemple résumant les résultats de chaque étape du calcul.

à la fin de ce chapitre,je donne les ressources hardware et les performances temporelle de bloc d'Euclid,ainsi qu'un exemple de simulation.

2.1 Architecture de bloc d'Euclid

L'algorithme d'Euclid modifié est présenté dans le chapitre 1. La cellule élémentaire qui réalise une itération selon cet algorithme est montrée sur la figure2.1.

Cette cellule contient divers blocs permettant à chaque itération de déterminer le

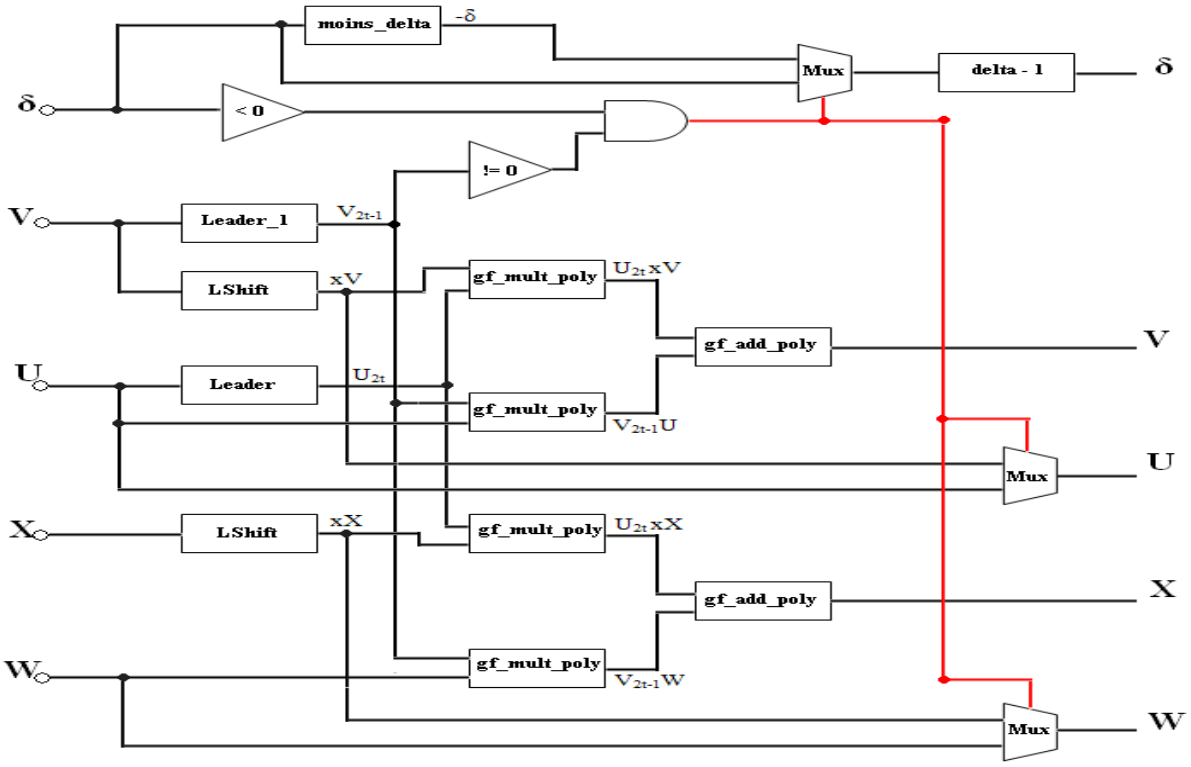

Figure 2.1 – shéma de la cellule d'Euclid

polynôme de localisation des erreurs $\sigma(x)$ et le polynôme d'amplitude des erreurs $\omega(x)$. Les signaux utilisés sont δ, U, V, X et Ω. Le signal δ est un entier, il a initialement (-1) comme valeur. Les signaux U, V, X et W sont des signaux sur (2t+1)*m bits. En effet, ces signaux sont une concaténation de (2t+1) symboles de m bits chacun. Les valeurs initiales de ces signaux sont montrées sur la figure2.2.

	Symbole 2t			Symbole t			Symbole 0 (m bits)
X	0	0	...	0	...	0	1
U	1	0	...	0	...	0	0
V	0	S_{2t-1}	...	S_t	...	S_1	S_0
W	0	0	...	0	...	0	0

Figure 2.2 – Valeurs initiales des signaux X, U, V et W.

Le bloc « Leader_1 » sert à extraire le symboleV_{2t-1} du signal V tandis que le bloc « Leader » permet d'extraire le symbole de poids le plus fort du signal U (U_{2t}). Le bloc

« LShift » sert à réaliser un décalage de m bits à gauche du signal U ou X. Le bloc « gf_mult_poly » réalise une multiplication d'un signal sur $((2t+1)*m)$ bits avec un signal sur (m) bits. Le premier signal est la représentation d'un polynôme de degré $(2t)$ $((2t+1)$ symboles) et le deuxième est un symbole (il peut être V_{2t-1} ou U_{2t}. Le bloc « gf_add_poly » effectue une addition dans le champ de Galois symbole par symbole entre deux polynômes. Comme l'addition dans le champ de Galois $(GF(2))$ est réalisée avec une porte XOR, l'addition effectuée par le bloc « gf_add_poly » peut être effectuée bit par bit.

Le bloc « moins_delta » calcul la valeur $(-\delta)$. Le test $(V_{2t-1} < 0)$ est effectué par le bloc « is_not_zero » (sur le schéma ce bloc est nommé «! = 0»). La sortie de ce bloc est un signal de niveau logique haut quand la condition est vérifiée. Le test $(\delta < 0)$ est réalisé avec le bloc « inferieur_zero » (sur le schéma ce bloc est nommé «! = 0 »). La sortie du bloc « inferieur_zero » est mis à 1 quand la valeur de δ est inférieur à zéro. Les sorites du bloc « is_not_zero » et « inferieur_zero » sont les entrées d'une porte AND. La sortie cette porte sert comme une entrée de sélection pour les multiplexeurs « Mux ». Ainsi, ces multiplexeurs sélectionne les bonnes valeurs des signaux U, Ω et la valeur de l'entrée du bloc « delta-1 » (cette valeur peut être δou $-\delta$) pour trouver la nouvelle valeur de δ.

Un registre est placé à la fin de la cellule pour la sauvegarde des différents résultats et assurer le calcul de l'itération suivante. Quand le nombre d'itération est égal à 2t, l'algorithme s'arrête. Le polynôme de localisation des erreurs est alors obtenu sur le signal X en prenant les (t+1) symboles du poids forts tandis que le polynôme d'amplitude des erreurs est obtenu sur le signal V en prenant les (t) symboles du poids forts à partir du symbole du poids le plus fort (voir Figure2.3).

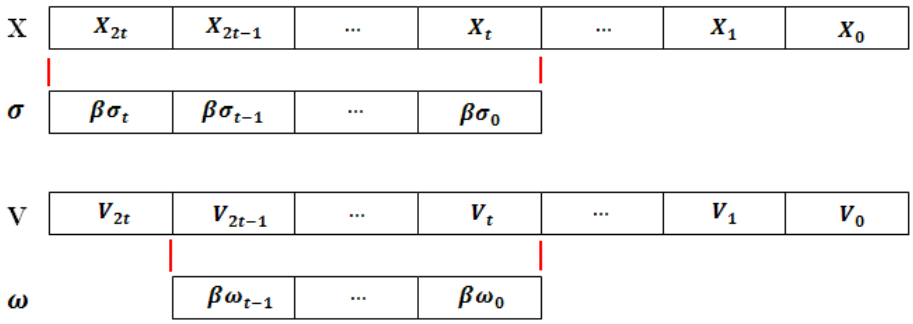

Figure 2.3 – Extraction des polynômes de localisation des erreurs et d'amplitude des erreurs à partir du X et V.

2.2 Implémentation de bloc d'Euclid

2.2.1 Implémentation des opérations dans les corps de Galois

Les opérations de bases dans les corps de Galois utilisées soit dans le bloc d'Euclid du décodeur de Reed-Solomon sont : l'addition, la multiplication dans les $GF(2^4)$ et $GF(2^8)$.

Implémentation de l'addition dans les $GF(2^m)$

Pour additionner deux éléments, on prendra la notation binaire de chaque élément et on les additionnera modulo 2. L'addition modulo 2 est une opération logique définie par l'opérateur logique « XOR » bit à bit.

la cellule élémentaire pour le bloc de l'addition dans les gf(2^4) est représentée sur la figure2.4

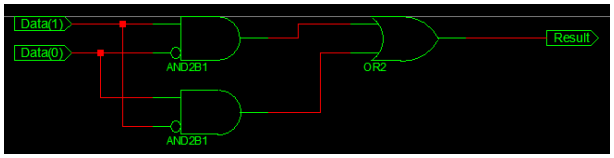

Figure 2.4 – cellule élémentaire pour le bloc de l'addition dans les gf(2^4)

Implémentation de la multiplication dans $GF(2^m)$

La multiplication dans le « champ de Galois » est une opération modulaire, c'est-à-dire que la multiplication entre deux éléments d'un champ fini donne toujours un élément dans le même champ. Elle est réalisée par la multiplication entre deux polynômes et ensuite de normaliser le résultat par rapport au polynôme primitif du champ choisi.

la cellule élémentaire pour le bloc de la multiplication dans les gf(2^4) est représentée sur la figure 2.5

2.2.2 Implémentation de bloc d'Euclid utilisé dans le décodeur de RS$(15, 9)$

Structure de bloc d'Euclid

Le circuit de bloc d' Euclid est composé d'un seul fichier qui contient les connexions entre les différents blocs déclarés comme composant. Ces derniers sont décrits dans des

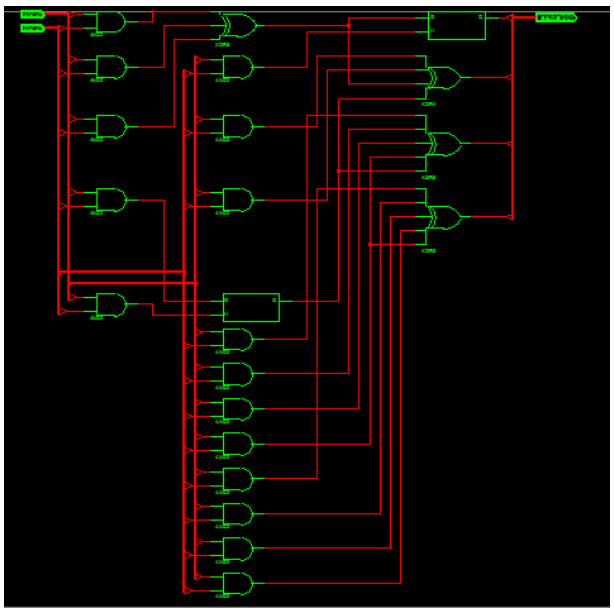

Figure 2.5 – cellule élémentaire pour le bloc de la multiplication dans les GF(2^4)

fichiers indépendants.Ce circuit est commandé par une machine d'état représenté sur la figure 2.6

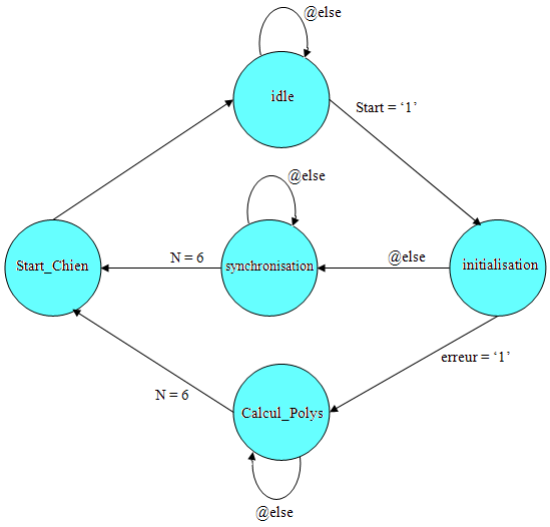

Figure 2.6 – Machine d'état de module d'Euclid.

La structure de bloc d'Euclid présenté dans le chapitre 3 est celle décrite dans la Figure2.7

La cellule élémentaire pour le calcul de bloc d'Euclid "MEA" est représentée sur la figure2.8.

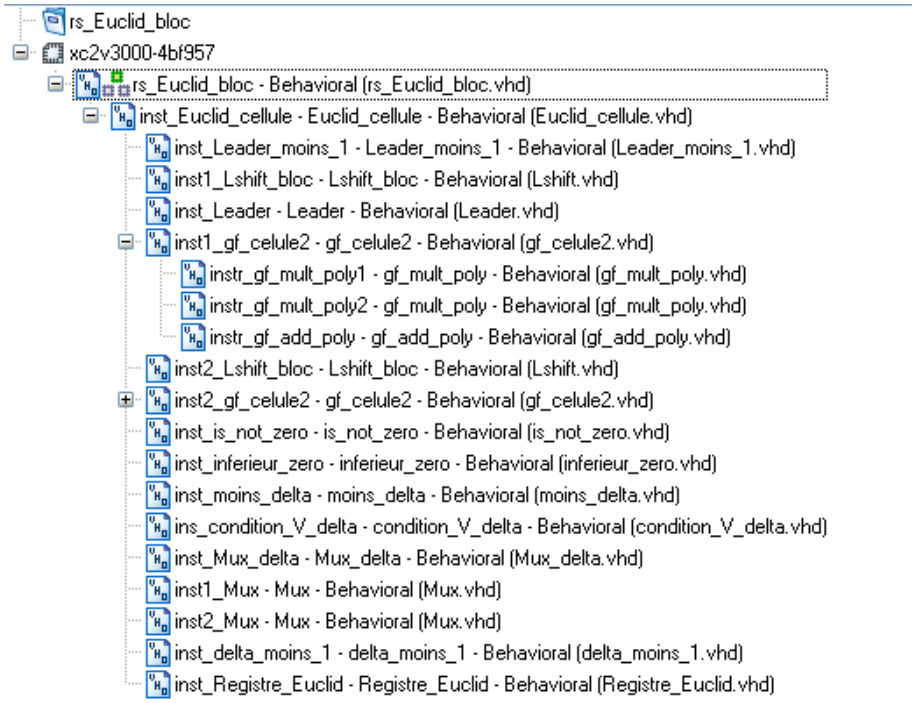

Figure 2.7 – Structure de bloc d'Euclid.

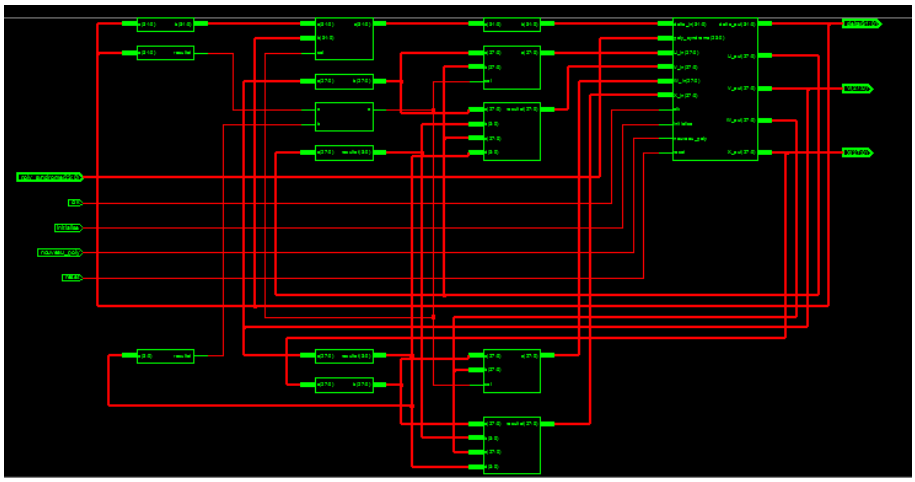

Figure 2.8 – Cellule élémentaire pour le calcul de bloc d'Euclid.

Le bloc qui calcule le polynôme d'amplitude et de localisation des erreurs suivant l'algorithme d'Euclid est :

Signaux de commande et d'entrée/sortie de bloc d'Euclid

les signaux utilisés sont :

- clk : signal d'horloge (std_logic).
- start_euclid : signal qui permet d'activer le bloc d'Euclid (std_logic), c'est une impulsion d'un cycle d'horloge qui précède le polynôme de syndrome.
- Reset : signal permettant la remise à zéro de bloc d'Euclid (std_logic).

Figure 2.9 – Schéma de bloc d'Euclid.

- erreurs : signal qui permet de signaler la présence d'erreur.Il est à 1 lorsque qu'il y a des erreurs (std_logic).
- poly_syndrome : signal d'entrée des symboles de polynôme de syndrome. Cette entrée est sur 24 bits (std_logic_vector(23 downto 0)).
- poly_amplitude : signal de sortie de polynôme d'amplitude des erreurs. Cette sortie est sur 12 bits (std_logic_vector(11 downto 0)).
- poly_localisation : signal de sortie de polynôme de localisation des erreurs. Cette sortie est sur 16 bits (std_logic_vector(15 downto 0)).
- Start_chienSearch : signal de sortie qui permet l'activation du bloc de chiensearch (std_logic).

Simulation de bloc d'Euclid

Un exemple de simulation de bloc d'Euclid est représenté sur la figure2.10

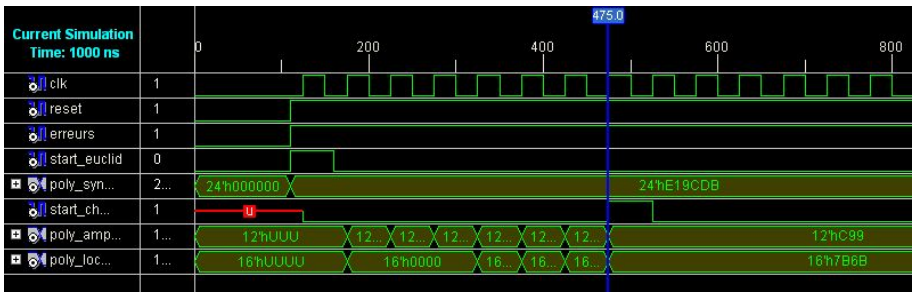

Figure 2.10 – Exemple de simulation de bloc d'Euclid.

Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour le bloc d'Euclid RS(15,9) après placement et routage sous Xilinx Virtex 2xc2v3000-4bf957 sont représentées respectivement dans les tableaux 2.1 et 2.2 :

FPGA Device	Virtex 2 XC2V1000-4FG456	
Type de ressources	Utilisé	Taux d'utilisation
Slice Flip Flop	134 de 10240	1%
4 inputs LUTs	400 de 10240	3%
Bonded IOBs	57 de 324	17%
Total occuped slices	207 de 5120	4%

Tableau 2.1 – Ressource hardware utilisé sous Xilinx Virtex 2 XC2V1000-4FG456 pour le bloc d'Euclid.

FPGA Device	Virtex 2XC2V1000-4FG456
Vitesse	-4
Période minimum	8.035 ns
Fréquence maximale	124.456 MHz

Tableau 2.2 – Performance temporelle pour le bloc d'Euclid.

2.2.3 Implémentation de bloc d'Euclid utilisé dans le décodeur de RS$(255, 239)$

Le bloc qui calcule le polynôme d'amplitude et de localisation des erreurs suivant l'algorithme d'Euclid est :

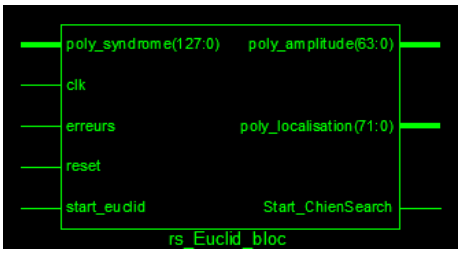

Figure 2.11 – Schéma de bloc d'Euclid.

Signaux de commande et d'entrée/sortie de bloc d'Euclid

les signaux utilisés sont :

- clk : signal d'horloge (std_logic).
- start_euclid : signal qui permet d'activer le bloc d'Euclid (std_logic), c'est une impulsion d'un cycle d'horloge qui précède le polynôme de syndrome.
- Reset : signal permettant la remise à zéro de bloc d'Euclid (std_logic).
- erreurs : signal qui permet de signaler la présence d'erreur.Il est à 1 lorsque qu'il y a des erreurs (std_logic).

- poly_syndrome : signal d'entrée des symboles de polynôme de syndrome. Cette entrée est sur 128 bits (std_logic_vector(127 downto 0)).
- poly_amplitude : signal de sortie de polynôme d'amplitude des erreurs. Cette sortie est sur 64 bits (std_logic_vector(63 downto 0)).
- poly_localisation : signal de sortie de polynôme de localisation des erreurs. Cette sortie est sur 72 bits (std_logic_vector(71 downto 0)).
- Start_chienSearch : signal de sortie qui permet l'activation du bloc de chiensearch (std_logic).

Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour le bloc d'Euclid RS$(255, 239)$ après placement et routage sous Xilinx Virtex XC2V1000-4FG456C sont représentées respectivement dans les tableaux 2.3 et 2.4 :

FPGA Device	**Virtex 2 XC2V1000-4FG456**	
Type de ressources	Utilisé	Taux d'utilisation
Slice Flip Flop	586 de 10240	5%
4 inputs LUTs	3856 de 10240	37%
Bonded IOBs	269 de 324	39%
Total occuped slices	269 de 5120	83%

Tableau 2.3 – Ressource hardware utilisé sous Xilinx Virtex 2 XC2V1000-4FG456 pour le bloc d'Euclid.

FPGA Device	**Virtex 2XC2V1000-4FG456**
Vitesse	-4
Période minimum	8.209 ns
Fréquence maximale	121.818 MHz

Tableau 2.4 – Performance temporelle pour le bloc d'Euclid.

2.3 Conclusion

Dans ce chapitre, j'ai exposé,tout d'abord, l'architecture de bloc d'Euclid que j'ai proposée. Ensuite,j'ai implémenté les opérations de bases dans les corps de Galois $GF(2^4)$ et $GF(2^8)$, ainsi que le celle de bloc résolvant l'équation clé de décodeur de Reed-Solomon RS$(15, 9)$.

A la fin,j'ai implémenté le bloc d'Euclid pour le décodeur de Reed-Solomon RS $(255, 239)$ et j'ai donné les ressources hardware ainsi que les performances temporelles de ce bloc.

Conclusion et Perspectives

L'objectif de ce mémoire est l'implémentation de bloc d'Euclid utilisé dans décodeur de Reed-Solomon pour les communications sans fils « WiMax » sur un circuit reconfigurable « FPGA ». J'ai décrit le principe de fonctionnement de codeur et décodeur RS.

Le bloc qui résout l'équation clé permettant de calculer les polynômes de localisation des erreurs et d'amplitude des erreurs est la partie principale de décodage. Plusieurs algorithmes existent et les plus répondus sont l'algorithme de Berlekamp-Massey et l'algorithme d'Euclid.Ce dernier est implémenté car il présente les meilleurs performances.

J'ai développé les codes en VHDL pour le bloc d'Euclid utilisé dans décodeur de Reed-Solomon RS$(15, 9)$ utilisé dans les communications spatiales ainsi que pour le codeur de Reed-Solomon RS$(255, 239)$ utilisé dans les communications sans fils « WiMax » en suivant l'architecture proposée.Ce bloc est simulé et synthétisé.

A l'avenir il serait pertinent d'implémenter le bloc d'Euclid pour la correction des erreurs et des effacements.

Bibliographie

[1] V. Glavac,"A VHDL code generator for Reed-Solomon encoders and decoders", mémoire pour l'obtention du grade de Master des Sciences Appliquées, Université de Concordia, 2003.

[2] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A methodfor solving key equation for decoding Goppa codes," Information and Control, vol. 27, pp. 87–99, January 1975.

[3] H. H. Lee, M. L. Yu and L. Song, "VLSI design of Reed-Solomon decoder architectures", IEEE Int. Symp. Circuits Syst. (ISCAS' 2000), vol. 5, May 2000, pp. 705-708.

[4] H. M. Shao, T. K. Truong, L. J. Deutsch, J. H. Yuen and I. S. Reed, "A VLSI design of a pipeline Reed-Solomon decoder", IEEE Trans. Comput., vol. C-34, pp. 393-403, May 1985.

[5] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equation for decoding Goppa codes", Information and Control, vol. 27, pp. 87–99, January 1975.

[6] D. V. Sarwate, Z. Yan, "Modified Euclidean Algorithms for Decoding Reed-Solomon Codes", IEEE