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 d'Ingénieur d'État en Électronique

Thème :

Implémentation sur circuit reconfigurable d'un décodeur de Reed-Solomon pour les communications sans fils

Réalisé par :

Mr **KHEMIS Redouane**
Mr **KACED Karim**

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. Nous nous intéressons tout particulièrement aux codes de Reed-Solomon utilisés dans les communications sans fils.

Après une étude théorique de ces codes, nous proposons deux architectures pour les codeurs et décodeurs de Reed-Solomon $(15, 9)$ et $(255, 239)$, l'une basé sur la théorie d'Euclid et l'autre sur les registres à décalage de Berlekamp-Massey. Les codes de description sont écrits en VHDL, la synthèse est l'implémentation aussi bien du codeur et des décodeurs est réalisé à l'aide de l'outil ISE de Xilinx sur carte FPGA.

Mots clés : codes de Reed-Solomon, codage canal, 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 Reed-Solomon codes used in the wireless communication. After theoretical study of these codes, we propose two designs for the Reed-Solomon encoders and decoders $(15, 9)$ and $(255, 239)$, the first is based on the Euclid theory and the second on the shift registers of Berlekamp-Massey. The codes of description are written in VHDL, the synthesis and the implementation of both encoder and decoders is carried out using the tool ISE of Xilinx on FPGA board.

Key words : Reed-Solomon codes, Channel coding, FPGA, errors correction codes.

ملخص

يندرج هذا العمل في إطار دراسة الشفرات المصححة للأخطاء. اهتممنا بالخصوص بالشفرات ريد سولومون المستخدمة في الاتصالات اللاسلكية.

بعد القيام بدراسة نظرية حول هذه الشفرات، نقترح تصميمين للمشفرات و مفككات التشفير ريد سولومون (15،9) و (255،239)، الأولى تعتمد على نظرية إقليدس و الثانية على سجلات بيرلكامب ماسي. تمت كتابة البرامج باللغة VHDL فيما تم انجاز الحوصلة و التثبيت لكل المشفر و مفككات التشفير بواسطة الأداة ISE ل Xilinx على FPGA.

الكلمات المفتاحية : تشفرة ريد سولومون ، تشفير القناة، FPGA، الشفرات المصححة للأخطاء.

Remerciements

Nous remercions nos parents qui nous ont beaucoup encouragés le long de ce projet. Et à l'issu de nos études faites au sein de l'École Nationale Polytechnique nous voudrions rendre un hommage tout particulier à : Nos promoteurs Mr. **M.Taghi** et Mr. **L.Abdelouel** qui nous ont honorés en acceptant de nous encadrer. Tous **les enseignants** du département **d'électronique** de l'Ecole Nationale Polytechnique.

Nos remerciements à Mr. R.SADOUN et MAMERI d'avoir accepté de juger notre modeste travail et de nous avoir honoré par leurs présences.

Nous remercions 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 MW,

Je dédie ce travail.

KHEMIS Redouane

Dédicace

À mes très chers parents,
À mes sœurs et frères,
À mes grands parents,
À mes tantes et oncles,
À tous qui m'ont soutenu le long de mon parcours,
À mes amis,

Je dédie ce travail.

Karim

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, ...) 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 des architectures optimisées pour l'implémentation de codeur et décodeur de Reed-Solomon,
- Simulation de codeur et décodeur de Reed-Solomon RS(15,9), RS(255,239) sous ISE de Xilinx,
- Implémentation de codeur et décodeur de Reed-Solomon RS$(15, 9)$ et RS$(255, 239)$ sur circuit reconfigurable « FPGA».

Organisation du mémoire

Ce mémoire est organisé quatre chapitres :

- Dans le premier chapitre, nous étudions les codes correcteurs d'erreurs en blocs linéaires après avoir donné un rappel sur les communications numérique et la théorie des codes en général.
- Au chapitre 2, nous expliquons la théorie de codage et de décodage des codes de Reed-Solomon et nous donnons leurs applications dans le domaine des communications sans fils notamment le WiMax.
- Dans le troisième chapitre, nous exposons l'architecture des différents blocs utilisés pour l'implémentation de codeur et décodeur de Reed-Solomon.
- Au chapitre 4, nous présentons les résultats de l'implémentation de codeur et décodeur de Reed-Solomon sur les circuits reconfigurables FPGA en donnant les ressources hardware utilisées ainsi que les performances temporelles de chaque bloc.
- En conclusion, nous résumons les objectifs réalisés et ensuite les éventuelles perspectives.

Chapitre 1

Introduction au codes correcteurs d'erreurs

1.1 Introduction

L'objectif fondamental d'un système de communication, est de reproduire en un point de la chaîne de communication un message transmis à partir d'un autre point, mais le problème qui se pose, est que le canal est généralement soumis à des perturbations, telles que le bruit et les interférences. Alors, comment faire pour établir une communication fiable dans ces conditions ?

Les premières tentatives d'évaluations de performance datent des contributions de Nyquist en 1924 [1] et de l'ingénieur Américain R.Hartetly en 1928 [2], mais l'étape décisive fut franchie en 1948 par Claude E. Shannon, lorsque parut son article fondateur de la théorie de l'information, intitulé « A Mathematical Theory of Communication » [3].

À la croisée de la théorie de l'information, des mathématiques et de l'électronique, le codage correcteur d'erreurs (codage de canal) a connu de nombreux développements depuis les travaux fondateurs de Shannon. Du simple code de Hamming (1950) aux récents turbo-codes (1993) en passant par les codes de Reed-Solomon (1960), le codage de canal a considérablement évolué et a intégré des concepts de plus en plus sophistiqués, en particulier le traitement probabiliste de l'information.

Dans ce chapitre, nous donnons, en premier temps, un rappel sur la communication numérique et la théorie des codes. Notre étude se focalisera par la suit sur les codes correcteurs d'erreur en blocs linéaires où nous présentons à la fin une liste non exhaustive de ces codes.

1.2 La chaine de transmission numérique sans fils

On désigne par le paradigme de Shannon, le schéma fondamental d'une communication numérique. Une source engendre un message à l'intention d'un destinataire. La source et le destinataire sont deux entités séparées, éventuellement distantes, qui sont reliées par un canal qui est le support de communication d'une part, mais qui d'autre part est le siège des perturbations.

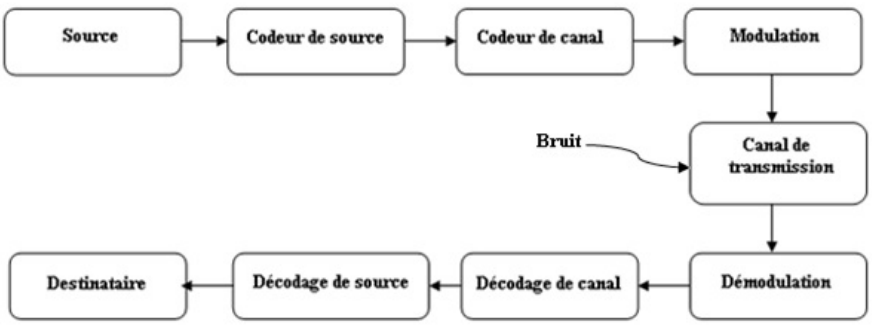

Figure 1.1 – Schéma fondamental d'une communication numérique : le paradigme de Shannon.

1.2.1 Le codage de source

« Le message codé est dépourvu de redondance, bien que le message initial provienne d'une source redondante ... » Gérard Bettail [4].

Le codage de source vise la concision maximale. L'usage d'un canal coûte d'autant plus cher que le message est long, le verbe "coûter" devant s'étendre ici en un sens très général, celui d'exiger l'emploi de ressources limitées telles que le temps, la puissance et la bande passante. Pour diminuer ce coût, on cherche à représenter le message avec le moins de bits possibles, c'est- à-dire le compresser. Pour ce faire, on essaye d'éliminer la redondance contenue dans le message transmis par la source.

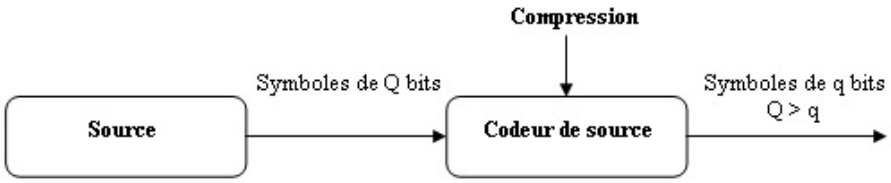

Figure 1.2 – Schéma simplifié d'un codeur de source.

Un point essentiel dans le codage de source est le critère de fidélité. Ce critère varie selon l'application. Les applications où la compression de données doit se faire sans perte, utilisent un codage de source appelé réversible, tandis que, les applications où les pertes sont tolérables, utilisent un codage dit non-réversible

1.2.2 Le codage de canal

« Le message est restitué sans erreurs après le décodage, bien que le message codé soit reçu à travers un canal bruité ... »Gérard Bettail [4].

La finalité du codage de canal est de protéger le message contre les perturbations du canal, et cela, en introduisant une redondance à l'information utile dans le message à transmettre. La redondance et l'information utile sont liées par une loi donnée. A la réception, le décodeur de canal exploite la redondance produite par le codeur dans le but de détecter, puis de corriger si c'est possible les erreurs introduites lors de la transmission [4]. Ce point sera détaillé encore plus dans les sections suivantes.

1.2.3 La modulation

La modulation consiste à effectuer un codage dans l'espace euclidien, espace généralement adapté aux canaux rencontrés en pratique [5]. Pour une modulation M-aire, on associe

à chaque mot de g bits un signal $x_i = 1, 2, \ldots, M$ de durée T choisi parmi les $M = 2^g$ signaux. Quant à la démodulation, son rôle est d'extraire les échantillons et de décider en faveur des symboles les plus probablement émis. Les données fournies par l'unité de démodulation seront traitées par ce que l'on appelle le décodeur. On distingue deux types de décodeurs, le premier est appelé décodeur à décision dure (Hard decision), car il fonctionne à partir des données fermes ($'0'$ ou $'1'$). Le second type est appelé décodeur à décision pondérée (Soft decision), car le démodulateur fournie au décodeur une valeur ferme accompagnée d'une mesure de fiabilité [6].

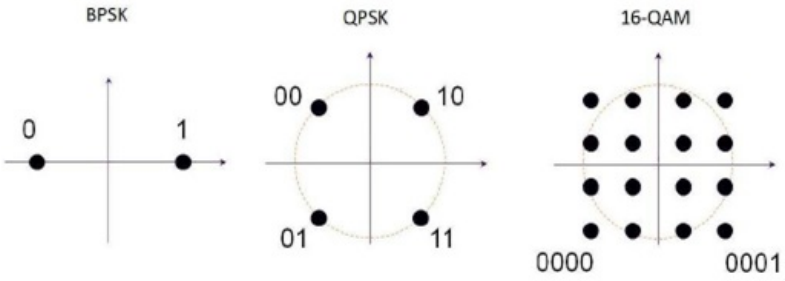

Figure 1.3 – Exemples de modulation numérique [4].

1.2.4 Le canal de communication sans fils

Le canal de communication sans fils est le support physique permettant d'acheminer un message entre une source et un ou plusieurs destinataires. Il existe plusieurs type de canaux, mais en théorie d'information, les canaux les plus utilisés sont appelés canaux discrets.

Le schéma du canal de communication sans fils est représenté sur la figure 1.4

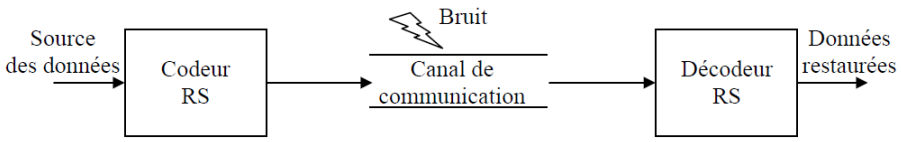

Figure 1.4 – Canal de communications sans fils.

Le canal est dit sans mémoire si le symbole courant de sortie ne dépend que du symbole courant d'entrée et il ne dépend pas des précédents ni des suivants.

1.3 Introduction à la théorie des codes

1.3.1 Définitions et propriétés

Un code sur un alphabet A de longueur n et de dimension k est un sous-ensemble C de A^n.On appelle ce code, un (n, k) code. Les éléments de C sont appelés mots du code (où mot code)

Le codeur de canal permet alors de générer un mot de code c de n bits (où symboles) à partir d'un mot d'information m de k bits (où symboles). Ce code engendre donc $(n - k)$ bits (où symboles) de redondance appelés bits (où symboles) de parité, que nous noterons par le vecteur p (figure1.5).

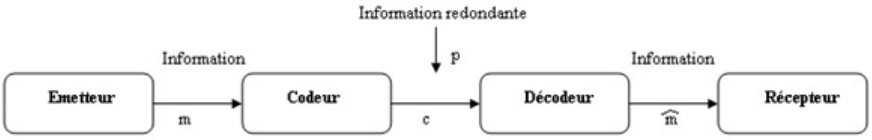

Figure 1.5 – Schéma simplifié d'un codeur/décodeur de canal.

Le rendement d'un code (où le taux d'un code) de dimension k et de longueur n le rapport :

$$R = \frac{k}{n} \tag{1.1}$$

On appelle distance de Hamming entre deux vecteurs x et y, le nombre de positions où ces deux vecteurs sont différents. On note cette distance $d_H(x, y)$

Le poids de Hamming d'un vecteur x, noté $\omega_H(x)$, est égal au nombre de composantes non nulles. On peut donc aussi exprimer le poids de Hamming comme la distance de Hamming entre x et un vecteur nul.

La capacité de correction d'un code C est mesurable si on a connaissance de la distance minimale du code notée d_{min}. La distance minimale d'un code est la plus petite distance de Hamming entre deux mots de codes :

$$d_{min} = \min(d_H(x, y) / x, y \in C, x \neq y) \tag{1.2}$$

Le poids minimum d'un code C est alors :

$$\omega_{min} = \min(\omega_H(x) \,/ x \in C, x \neq 0) \tag{1.3}$$

Comme nous nous intéressons à des codes linéaires (et donc la différence entre deux mots de code est également un mot de code), la distance minimale entre deux mots de code distincts est égale au plus petit poids de Hamming d'un mot de code non nul.

$$d_{min} = \omega_{min} \tag{1.4}$$

1.3.2 Codes correcteurs d'erreurs linéaires

Les codes correcteurs d'erreurs sont un outil visant à améliorer la fiabilité des transmissions sur un canal bruité. La méthode qu'ils utilisent consiste à envoyer sur le canal plus de données que la quantité d'information à transmettre. Une redondance est ainsi introduite. Si cette redondance est structurée de manière exploitable, il est alors possible de corriger d'éventuelles erreurs introduites par le canal. On peut alors, malgré le bruit, retrouver l'intégralité des informations transmises au départ.

Une grande famille de codes correcteurs d'erreurs est constituée des codes en bloc. Pour ces codes l'information est d'abord coupée en blocs de taille constante et chaque bloc est transmis indépendamment des autres, avec une redondance qui lui est propre. La plus grande sous-famille de ces codes rassemble ce que l'on appelle les codes linéaires.

1.4 Codes en bloc linéaires

1.4.1 Définition

Dans un code en bloc, les n symboles des mots code sont calculés uniquement avec les k symboles des mots d'information :

$$m = [m(1)\ m(2)\ \cdots m(k)\ ] \longrightarrow\ c = [c(1)\ c(2)\ \cdots c(n)\ ] \tag{1.5}$$

Au besoin, le message initial est découpé en paquets, ou blocs, de taille k.

La linéarité signifie que les n symboles du mot code sont obtenus par combinaison linéaire des k symboles du mot d'information.

Le mot code c s'obtient à partir du mot d'information m par une expression matricielle de la forme[7] :

$$c = m * G \tag{1.6}$$

où m est un vecteur ligne de taille $(1 * n)$ et G est une matrice de taille $(k * n)$ appelée matrice génératrice du code.

1.4.2 Forme systématique d'un code

Un code est entièrement déterminé par sa matrice génératrice G.

Les vecteurs-lignes de G sont eux-mêmes des mots-code, et sont supposés linéairement indépendants.

Cela revient à imposer que :

$$Si \qquad m_1 \neq m_2 \qquad alors \qquad c_1 = m_1\, G \neq c_2 = m_2\, G$$

Donc, des mots d'information différents ne peuvent pas être identiquement codés (en d'autre terme, la matrice G est de rang k).

On montre de plus que les combinaisons linéaires sur les lignes de G ainsi que les permutations sur les colonnes de G laissent inchangées la probabilité d'erreur par mot code. Cela résulte que la distance du code est inchangée.

D'après ces propriétés, la matrice G peut s'écrire sous la forme[7] :

$$\check{G} = \left[\, I_k \mid P_{n*(n-k)} \,\right] \tag{1.7}$$

La matrice $\check{G}$ est la forme systématique de la matrice G, où I_k est la matrice identité de taille k, et la matrice P est appelée la matrice de parité du code.

Lorsque G est sous sa forme systématique, les mots code commencent par k bits d'information et sont complétés par $(n-k)$ bits de redondance :

$$Si \qquad m = [m_1\ m_2\ \ldots\ m_k] \qquad alors \qquad c = m * G = [m_1\ m_2\ \ldots\ m_k\ p_1\ p_2\ p_{n-k}]$$

Par exemple, le code linéaire (7,4) défini par sa matrice génératrice :

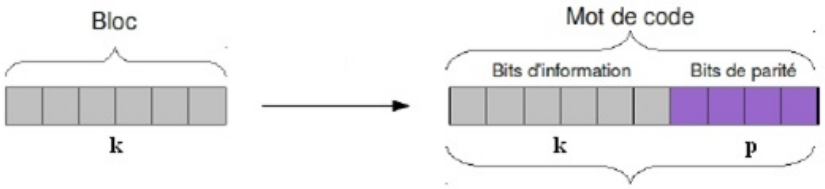

Figure 1.6 – Principe de codage systématique.

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

peut se mettre sous la forme systématique :

$$\check{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Ainsi, si un bloc $m = [m_1\, m_2\, m_3]$ est codé en $c = m * G = [m_1\, m_2\, m_3\, p_1\, p_2\, p_3]$
où

$$\begin{cases} p_1 = c_1 + c_3 + c_4 \\ p_2 = c_1 + c_2 + c_3 \\ p_3 = c_2 + c_3 + c_4 \end{cases}$$

Tout se passe comme si, le codage consistait simplement en l'ajout de$(n-k)$ bits de redondance.

1.4.3 Majoration de la distance d'un code

Une fois que la matrice génératrice du code est mise sous la forme systématique :

$$\breve{G} = \left[I_k \mid P_{n*(n-k)} \right] = \left[\begin{array}{cccc} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{array} \mid P_{n*(n-k)} \right]$$

On peut dénombrer au plus $(n-k+1)$ zéros sur chaque ligne. Ainsi, $d_{min} = \omega_{min} \leq n-k+1$. Donc, la distance minimale d'un code (n, k, d) est majorée par n-k+1.

1.4.4 Caractérisation d'un code

On appelle matrice de contrôle [7]de parité d'un code linéaire C, une matrice H de dimension$(n-k)*n$ et de rang $(n-k)$ qui vérifie :

$$HG^T = 0 \tag{1.8}$$

Si m est un bloc d'information, de mot code $c = mG$, alors, on doit avoir :

$$H\,c^T = H(G\,m)^T = HG^T\,m^T = (HG^T)\,m^T = 0 \tag{1.9}$$

On a ainsi la propriété : le bloc c reçu est un mot code (donc jugé non erroné) si et seulement si $H\,c^T = 0$

La matrice de contrôle de parité a pour expression :

$$H = \left[(P^T)_{(n-k)*k} \,| I_{n-k} \right] \tag{1.10}$$

1.4.5 Décodage par syndrome

Soit un code linéaire de matrice de contrôle de parité H, c un mot envoyé et $y = c + b$ le mot reçu.

Alors, on a[8] :

$$Hy^T = H\,(c+b)^T = H\,c^T + H\,b^T = H\,b^T \tag{1.11}$$

Ainsi, le vecteur $s = Hy^T$ ne dépend que du bruit et non pas du mot envoyé. On l'appelle le syndrome associé à y.

À partir de la connaissance de H et de mot reçu, on peut donc espérer déterminer le bruit b, car : $s = Hy^T = Hb^T$

Une fois le bruit connu, le mot envoyé c se déduit simplement par l'addition de mot reçu et le bruit calculé : $c = y \bigoplus b$

1.5 Différents codes en bloc

Nous donnons ci-dessous une liste non exhaustive de familles de code en bloc linéaire. Les codes que nous considérons sont binaires ou sont définis sur une extension du corps à deux éléments. Nous noterons (n, k, d) un code linéaire défini sur un alphabet à q éléments de longueur n, de dimension k et de distance minimale d. Nous avons choisi de ne lister ici que des familles de codes ayant un intérêt pratique.

1.5.1 Les code à répétition

Le code à répétition consiste à répéter n fois le bit d'information. C'est un code $(n, 1, n)$, de dimension 1, de longueur n et de distance n. Pour n impair, ce code est parfait et optimal lorsque l'on protège un seul bit de donnée.

En théorie de l'information pour un modèle de bruit donné (canaux sans mémoire) et à rendement constant on peut avoir une probabilité d'erreur qui tend vers 0. Pour un code à répétition, le rendement décroit pour une probabilité d'erreur qui tend vers 0. Ce code ne tire malheureusement pas parti du faite qu'il est plus efficace de protéger simultanément plusieurs bits d'information.

Le décodage s'effectue par vote majoritaire et la capacité de correction est donnée par $\frac{n-1}{2}$.

Ce code est surtout utilisé en télécommunication pour reconnaitre une modulation de signal.

1.5.2 les codes de parité

Un code de parité prend k bits de l'information et ajoute un bit de parité. C'est un code $(k+1, k)$. Deux types de code de parité sont utilisés : parité pair et parité impair. Dans un code parité pair, le bit de parité est choisi de sorte que le nombre de 1 soit pair, dans un code parité impair, le bit de parité est choisi de sorte que le nombre de 1 soit impair. Si une erreur se produit dans la transmission, alors le nombre de 1 va changer. L'algorithme de décodage consiste à compter le nombre de 1. Si il est pair (pour un code de parité pair) , alors aucune erreur n'est détectée. Cependant, si le nombre de 1 est impair, alors une erreur est détectée.

Ce code peut avoir un taux d'information aussi proche de 1 que l'on souhaite, mais a toujours une distance minimale de 2, et donc, ne permet pas de correction, sauf d'un effacement dont on connait la position. Il permet cependant la détection d'une erreur par bloc. Si un bloc est erroné, on peut alors demander un nouvel envoi. Cette méthode peut donc être utilisée avec des canaux introduisant peu d'erreurs et dans lesquelles une demande de répétition est possible.

1.5.3 Les codes de Hamming

Les premiers codes pratiques de correction d'erreurs sont inventés par Hamming en 1948, et il est publié en 1950. Ils sont une classe des codes blocs linéaires capable de corriger une seule erreur. Les paramètres de ces codes sont :

Longueur du code :$n = 2^m - 1$

Nombre de bits d'information :$k = 2^m - m - 1$

Nombre de bits de parité :$n - k = m$

Capacité de correction d'erreur : $t = 1$

Le code de Hamming se définit simplement au moyen d'une matrice de parité H de dimension $(k * n)$ avec $n = 2^m - 1$. La matrice de parité possède toutes ses colonnes distinctes non nulles. Par exemple, pour le code de Hamming $(7, 4)$, une des matrices de parité s'écrie :

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Les colonnes de la matrice de parité étant toutes distinctes, la distance minimale du code est 3. Plus généralement, les codes de Hamming sont des codes$(2^m-1, 2^m-m-1, 3)$. La matrice génératrice correspondante est :

$$G = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Le décodage de ce code s'effectue par syndrome en utilisant la matrice de parité. À chaque erreur de poids 1 correspond une valeur unique de syndrome comme il est montré sur le tableau1.1 :

Modèle d'erreur	**Valeur de syndrome**
0000000	000
0000001	111
0000010	110
0000100	101
0001000	011
0010000	001
0100000	010
1000000	100

Tableau 1.1 – Exemple de décodage par syndrome pour les codes BCH.

La correction se fait alors par l'addition de la valeur de l'erreur avec le message reçu (addition bit par bit).

Cette méthode permet de corriger une unique erreur, si deux erreurs se produisent, ce code est incapable de les corriger et de surplus en ajoute une troisième.

Le code de Hamming généralisé est un code $((2^m, 2^m - m\,,\, 3)$ où le 2^m bit est un bit de parité contrôlant les $2^m - 1$ bits de précédent. Ce nouveau code permet de détecter une deuxième erreur mais sans pouvoir la corriger.

1.5.4 Les codes cycliques

Les codes cycliques ont été étudiés pour la première fois par Prange en 1957. Il s'agit d'une sous-classe des codes blocs. Ils possèdent donc toutes les caractéristiques de ces codes, auxquelles viennent se rajouter d'autres spécifications.

Ces codes possèdent des propriétés intéressantes. Ils ont une forte structure mathématique qui simplifie le codage et le décodage, rendant ainsi une implémentation relativement simple.

Un code linéaire C est dit cyclique[5] s'il est stable par permutation circulaire des coordonnées, i.e

$$(c_0,\, c_1,\, ,\ldots\, ,c_{n-1}) \in C \quad \Longrightarrow \quad (c_{n-1}\, ,c_0\, ,\ldots\, ,c_{n-2}) \tag{1.12}$$

Si on représente le vecteur (mot code) ci-dessus par un polynôme $c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$, alors la permutation circulaire d'une unité à droite correspond à la multiplication par x modulo $x^n - 1$. De plus, si on représente le mot message $m = (m_0,\, m_1,\, \ldots,\, m_{k-1})$ par un polynome $m(x)$ de degré $(k-1)$, alors on écrire la relation suivante :

$$c(x) = m(x) * g(x) \tag{1.13}$$

Où $g(x)$ est un polynôme de degré $(n-k)$.

Lorsqu'il est de degré minimal et unitaire,le polynôme $g(x)$ s'appelle polynôme générateur du code. Ce polynôme est unique et il est diviseur de $x^n - 1$. Le polynôme de parité h(x) peut être déterminé par l'équation :

$$g(x) * h(x) = x^{n-1} \tag{1.14}$$

Le polynôme de syndrome, s(x), est trouvé par la relation :

$$s(x) = r(x) * h(x)\ modulo\ (x^{n-1}) \tag{1.15}$$

Où $r(x)$ est la représentation polynômiale du message reçu.

Si $s(x)$ est égal au polynôme nul, on peut conclure qu'il n'ya pas erreur et le message est validé.

Il existe deux types important des codes cycliques : les code BCH et les codes de Reed-Solomon.

Les codes BCH

Les codes BCH (Bose-Chaudhuri-Hocquenghem) sont des codes cycliques particuliers. Ils ont les paramètres suivants :

Longueur du code : $n = 2^m - 1(bits)$

Capacité de correction : t

Nombre de bits de parité : $n - k \leq m * t$

La théorie des codes BCH se base sur les mathématiques des corps de Galois. Un corps[7] est un ensemble K muni de deux lois internes notées en général + et * vérifiant

- $(K, +)$ forme un groupe commutatif dont l'élément neutre est noté 0.
- $(K \setminus 0, *)$ forme un groupe multiplicatif.
- La multiplication est distributive pour l'addition (à gauche comme à droite).

On parle alors de corps $(K, +, *)$.

Le nombre d'éléments de champ est appelé l'ordre du champ. Il est désigné par la lettre q. Si le corps contient un nombre fini d'éléments, on dit que le corps est un corps fini. L'ordre d'un élément α dans le champ F est le plus petit entier positif n tel que $\alpha^n = 1$. Un élément de F est dit élément primitif si son ordre est égal à $(q-1)$. Les puissances des éléments primitifs génèrent tout les éléments non nul de F. Le polynôme minimale d'un élément α est le polynôme de degré le plus petit dans$GF(2)$ dont α est une racine.

Soit α un élément primitif de $GF(2^m)$. Alors, le polynôme générateur $g(x)$ d'un code BCH est le polynôme de degré minimal de $GF(2)$, qui possède $2t$ racines qui sont des puissances consécutives de α. Si on désigne par $\phi_i(x)$ le polynôme minimal de α^i, alors $g(x)$ est donné par[9] :

$$g(x) = PPCM\left\{\Phi_1(x), \Phi_2(x), \ldots, \Phi_{2t}(x)\right\} \tag{1.16}$$

où PPCM est le plus petit commun multiple. Il est clair que c'est un t-correcteur code.

Cependant, on ne peut pas augmenter t indéfiniment. Le tableau1.2 illustre quelques codes BCH et leurs capacités de correction.

Le polynôme de syndrome peut être calculé par multiplication polynomiale. Pour des valeurs larges de t (la capacité de correction d'erreur d'un code), le degré de polynôme de syndrome devient proportionnellement large. Comme conséquence, l'approche de la table de syndrome pour la correction d'erreur ne devient pas pratique.

Le polynôme de localisation des erreurs, $\sigma(x)$, est le polynôme dont les racines sont les emplacement des erreurs. Des procédures pour le calcul de ce polynôme seront présentées dans le chapitre 2.

n	k	t	n	k	t
7	4	1	127	120	1
15	11	1		113	2
	7	2		106	3
	5	3		99	4
31	26	1		92	5
	21	2		85	6
	16	3		78	7
	11	5		71	9
	6	7		64	10
63	57	1		57	11
	51	2		50	13
	45	3		43	14
	39	4		36	15
	36	5		29	21
	30	6		22	23
	24	7		15	27
	18	10		8	31
	16	11			
	10	13			
	7	15			

Tableau 1.2 – Codes BCH générés par des éléments primitifs d'ordre inférieur à 127 [9].

Les codes de Reed-Solomon

Les codes BCH sont des codes binaires. Partant d'un alphabet binaire, il peut s'avérer plus pratique de représenter les données issues d'une source binaire par des symboles q-aires, surtout lorsque q est une puissance de 2. Un symbole q-aire correspondra à un mot constitué de q éléments binaires.

Les codes de Reed-Solomon sont des codes cycliques q-aires. Ils sont introduits par Irvine Reed et Gustave Solomon en 1960 [10]. Les paramètres de ces codes sont :

Longueur du code : $n = 2^m - 1$ (symboles)

Longueur du symbole : m

Capacité de correction d'erreur : t (symboles)

Nombre de symboles de parité : $n - k = 2t$ (symboles)

Travaillant sur des corps finis, ces codes sont basés sur le sur-échantillonnage des polynômes et sur l fait qu'un polynôme de degré t est déterminé par (t+1) valeurs. Ils

sont efficaces dans la correction d'erreurs par paquet du fait de leur structure. Ils seront détaillés dans le chapitre 2.

1.5.5 produits,codes concaténés

1.5.6 Les codes produits

Le produit de deux codes linéaires en bloc sur un même alphabet q-aire est définit comme l'ensemble des matrices génératrices dont toutes les lignes sont dans un code et toutes les colonnes dans un autre code. Il s'agit de l'une des premières constructions de codes disposant d'un algorithme de décodage itératif.

1.5.7 Les codes concaténés

Les codes concaténés ont constitué, et constituent toujours, l'une des constructions les plus utilisées pour obtenir une protection contre des niveaux de bruit importants.

Dans un schéma concaténé, l'information est codée deux fois séquentiellement. Tels qu'ils ont été décrits initialement par Forney son travail de thèse en 1965 [11], ils utilisent deux codes en bloc. Le premier, le code externe, est défini sur un alphabet de grande taille q et le second, le code interne, binaire en général, codera chaque symbole q-aire afin de fournir une protection supplémentaire. Par la suite, le code interne a été remplacé par un code convolutif [7]. Le schéma concaténé avec un code convolutif interne et un code de Reed-Solomon externe a été standardisé pour beaucoup types de communications.

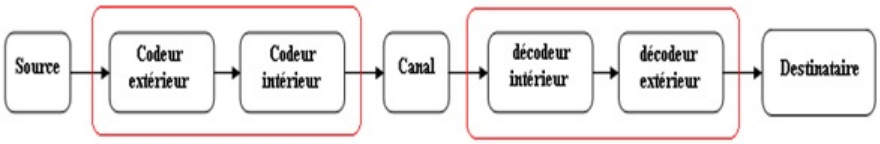

Figure 1.7 – Concaténation de deux codes correcteurs d'erreurs.

1.6 Conclusion

Dans ce chapitre,aprés avoir expliqué brièvement chaque partie dans le domaine des communications sans fils, nous avons présenté un panorama sur les codes correcteurs d'erreurs.Vu la diversité de ces codes, nous avons mis l'accent sur les codes en bloc linéaire où nous avons exposé quelques variantes accompagnées de leurs algorithmes de décodage.

Chapitre 2

Codes de Reed-Solomon

2.1 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[12]. 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.

Nous commençons ce chapitre par la présentation des corps de Galois, puis nous détaillons les codes de Reed-Solomon et leurs théorie de codage et décodage. Nous finissons par l'exposé de l'application des codes RS dans le domaine des communications sans fils notamment la norme WiMax.

2.2 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.

2.2.1 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).

2.2.2 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.

2.2.3 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.

2.2.4 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 tableau2.1 ci-dessous, Les polynômes primitifs pour les principaux « corps de Galois » :

2.2.5 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 2.1 – 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 tableau2.2 :

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 2.2 – éléments de $GF(2^4)$.

2.2.6 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 :

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

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

2.3 Principe des codes de Reed-Solomon

2.3.1 Définition

Les codes de Reed–Solomon[12] sont des codes de détection et de correction des erreurs basés sur les corps de Galois. Ce sont des codes particuliers des codes BCH.

2.3.2 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 :

$$n = k + 2t = 2^m - 1 \tag{2.1}$$

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.

2.3.3 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 $GF(2^4)$.

2.3.4 Codeur de Reed Solomon

Théories du codage

L'équation clé définissant le codage[13] systématique de Reed–Solomon (n, k) est :

$$c(x) = i(x)x^{n-k} + \left[i(x)x^{n-k}\right] mod(g(x)) \tag{2.2}$$

Avec :

$c(x)$: Polynôme du mot-code, degré $n-1$.

$i(x)$: Polynôme d'information, degré $k-1$.

$\left[i(x)x^{n-k}\right] mod(g(x))$: 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 :

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

2.3.5 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 :

$$e(x) \;=\; r(x)-c(x) \;=\; r(x)+c(x) \tag{2.3}$$

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 :

$$r(x) \;=\; c(x) + e(x) \tag{2.4}$$

On obtient :

$$S_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) \tag{2.5}$$

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

$$S(x) \;=\; S_{2t}x^{2t-1} + \cdots + S_2x + S_1 \tag{2.6}$$

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, soit de l'algorithme du Berlekamp-Massey, ou l'algorithme d'Euclide.

1. **Algorithme d'Euclide (EA)**

   L'algorithme d'Euclide[12] 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 :

$$PGCD(r_0(x), r_1(x)) = a(x)r_0(x) + b(x)r_1(x). \tag{2.7}$$

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

$$\sigma(x) = \prod_{k=1}^{v}(1 - \sigma^{i_k}x) = \sigma_v x^v + \sigma_{(v-1)}x^{v-1} + \cdots + \sigma_1 x + 1 \tag{2.8}$$

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

$$\omega(x) = S(x).\sigma(x) \tag{2.9}$$

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 $2t$:

$$S(x)\sigma(x) \;=\; \omega(x)\, mod\,(x^{2t}) \tag{2.10}$$

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 :

$$\deg(\omega(x)) \;<\; \deg(\sigma(x)) \;\leq\; t. \tag{2.11}$$

L'organigramme de l'algorithme d'Euclide est donné dans la figure2.1 ci-dessous :

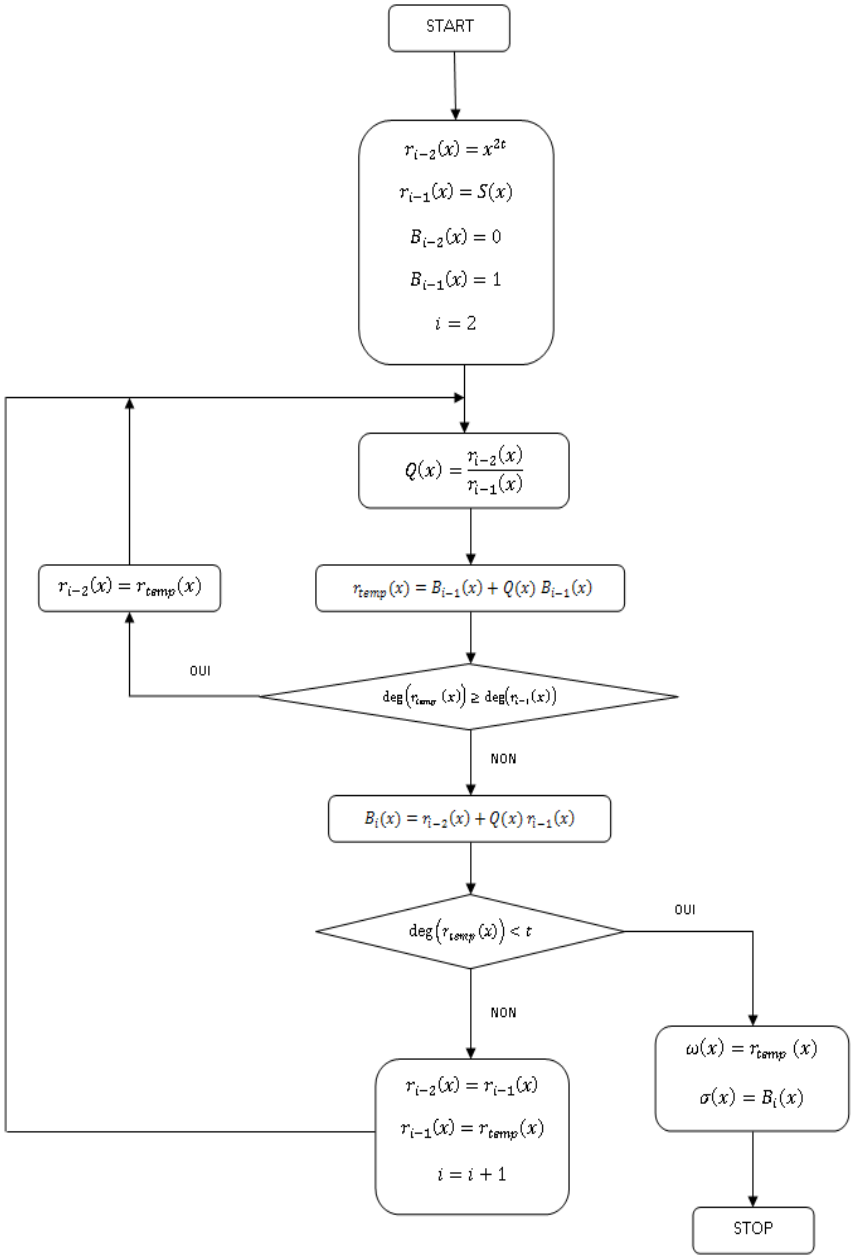

Figure 2.1 – L'organigramme de l'algorithme d'Euclide pour le calcul du polynôme de localisation et le polynôme d'amplitude.

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

$$\sigma_i(x) = \sigma_{i-2}(x) + \sigma_{i-1}(x)Q_i(x) \tag{2.12}$$

Avec :

$$\sigma_i(x) = B_i(x) \tag{2.13}$$

2. **Algorithme d'Euclide modifié (MEA)**

L'algorithme d'Euclid est un algorithme récursif qui nécessite des opérations de multiplication et de division polynomiales. La condition d'arrêt de cet algorithme est ($\deg(\omega) < t$). 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 [13], [14]. Sugiyama et al. [15] 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 [16]. 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 figure2.2.

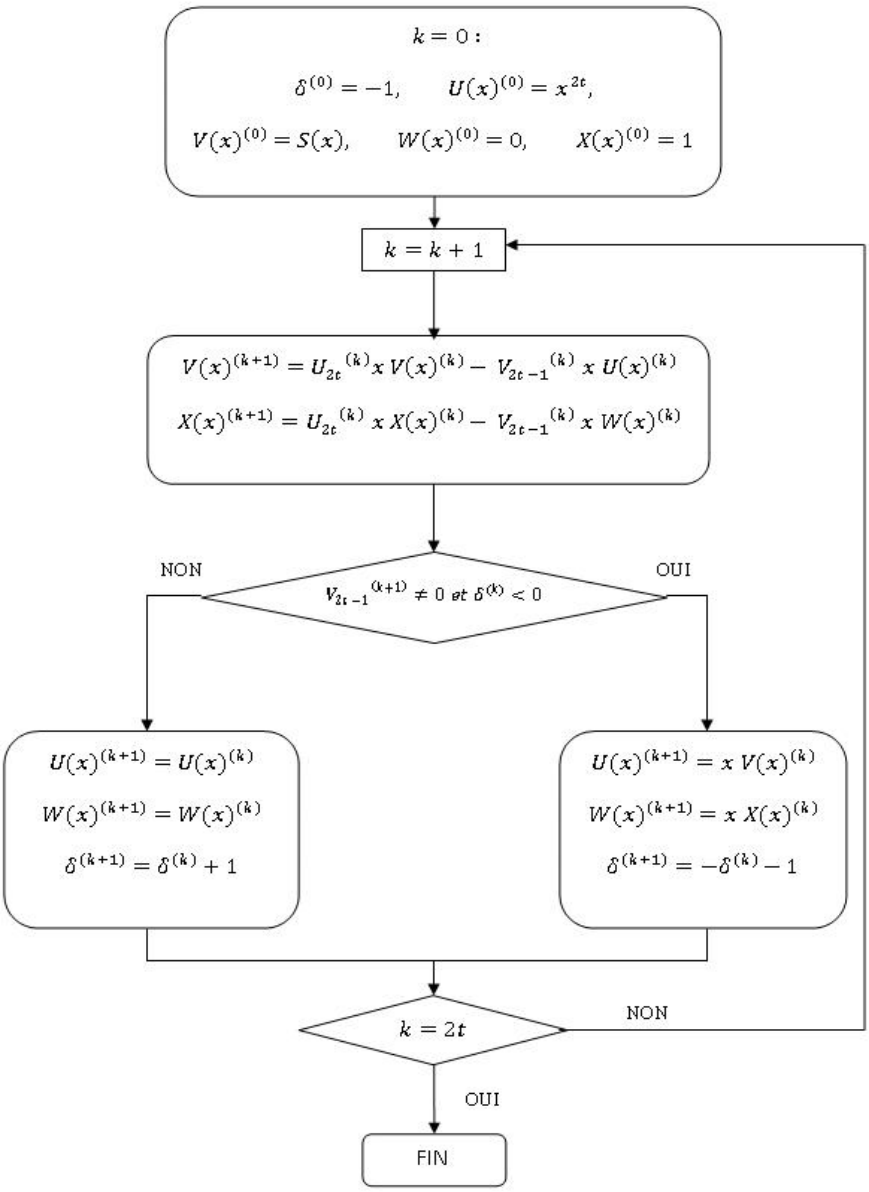

Figure 2.2 – L'organigramme de l'algorithme MEA pour le calcul du polynôme de localisation et le polynôme d'amplitude.

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

$$\delta = 2v - 2t - 1 < 0 \tag{2.14}$$

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

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.

3. **Algorithme de Berlekamp-Massey (BM)**

L'algorithme de Berlekamp–Massey[12] est basé sur la synthèse du polynôme de localisation des erreurs en utilisant des registres à décalage pour la résolution des identités des Newton.

En 1969, Massey a développé un algorithme itératif [17] basé sur le principe des registres à décalage de Berlekamp. Ce nouvel algorithme est appelé algorithme de Berlekamp –Massey. L'identité de Newton à résoudre pour les codes de Reed-Solomon est :

$$\begin{aligned} S_{v+1} + \sigma_1 S_v + \sigma_2 S_{v-1} + \cdots + \sigma_v S_1 &= 0. \\ S_{v+2} + \sigma_1 S_{v+1} + \sigma_2 S_v + \cdots + \sigma_v S_2 &= 0. \\ \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots & \\ S_{2t} + \sigma_1 S_{2t-1} + \sigma_2 S_{2t-2} + \cdots + \sigma_{2t} S_{2t-v} &= 0. \end{aligned}$$

Le but de l'algorithme de Berlekamp-Massey est de trouver le polynôme $\sigma(x)$ de degré le plus petit possible et satisfaisant l'identité de Newton. Le calcul de $\sigma(x)$ est effectué itérativement avec $2t$ étapes selon L'organigramme de la figure2.3.

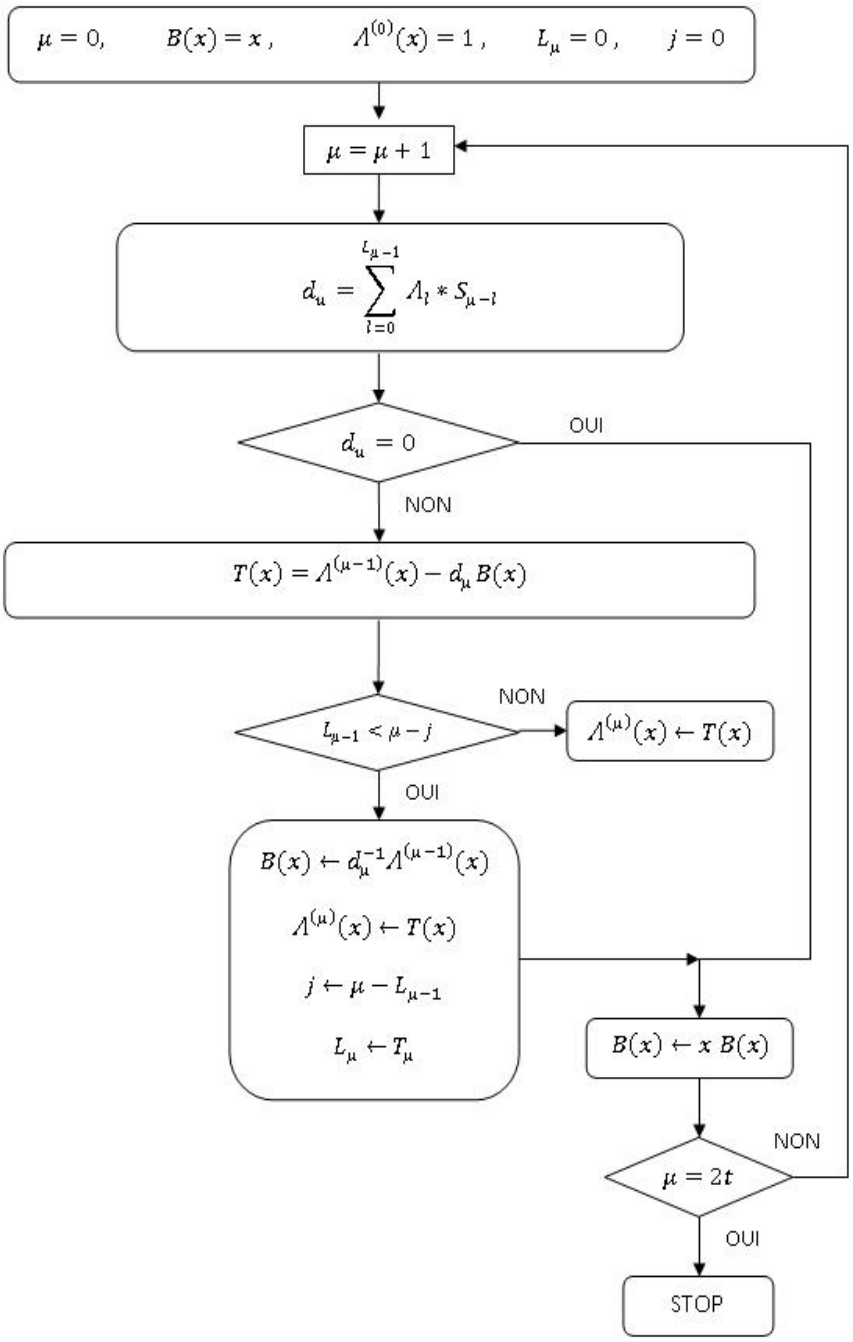

Figure 2.3 – L'organigramme de l'algorithme de Berlekamp-Massey.

Avec :

$\Lambda^{(\mu)} = \sigma^{(\mu)}(x)$: polynôme de localisation des erreurs après μ étapes.

L_μ : degré du polynôme de localisation des erreurs après μ étapes.

d_μ : incohérence après μ étapes.

$T(x)$: polynôme de connexion annulant l'incohérence.

Le polynôme calculé sera sous la forme :

$$\sigma(x) = \sigma_0 + \sigma_1 x + \cdots + \sigma_t x^t \tag{2.15}$$

Le polynôme d'amplitude sera de degré t-1 :

$$\omega(x) = [S(x)\sigma(x)] \mod x^t. \tag{2.16}$$

$$\begin{aligned} \omega(x) &= [(S_{2t}x^{2t-1} + \cdots + S_2 x + S_1)(\sigma_0 + \sigma_1 x + \cdots + \sigma_t x^t)] \; mod \; x^t. \\ &= (S_t + \sigma_1 S_{t-1} + \cdots)x^{t-1} + \cdots + (S_2 + \sigma_1 S_1)x + S_1. \end{aligned}$$

4. **Algorithme de Berlekamp-Massey sans Inversion Etendu (EIBM)**

L'algorithme original de Berlekamp-Massey se déroule en six itérations pour calculer le polynôme de localisation des erreurs. Une fois ce polynôme est trouvé, le polynôme d'amplitude des erreurs se calcul suivant l'équation 2.16 . Cette opération fait augmenter la latence de l'algorithme, ce qui diminue la vitesse de décodage.En plus, des opérations d'inversion dans le champ de Galois sont nécessaires.

L'algorithme EIBM (Extended Inversionless Berlekamp-Massey) permet de calculer en même temps les deux polynômes (de localisation et d'amplitude des erreurs) de façon itérative sans l'utilisation d'inversion dans le champ de Galois. Cet algorithme est présenté dans [9].
Les étapes de cet algorithme sont illustrées sur L'organigramme de la figure2.4.

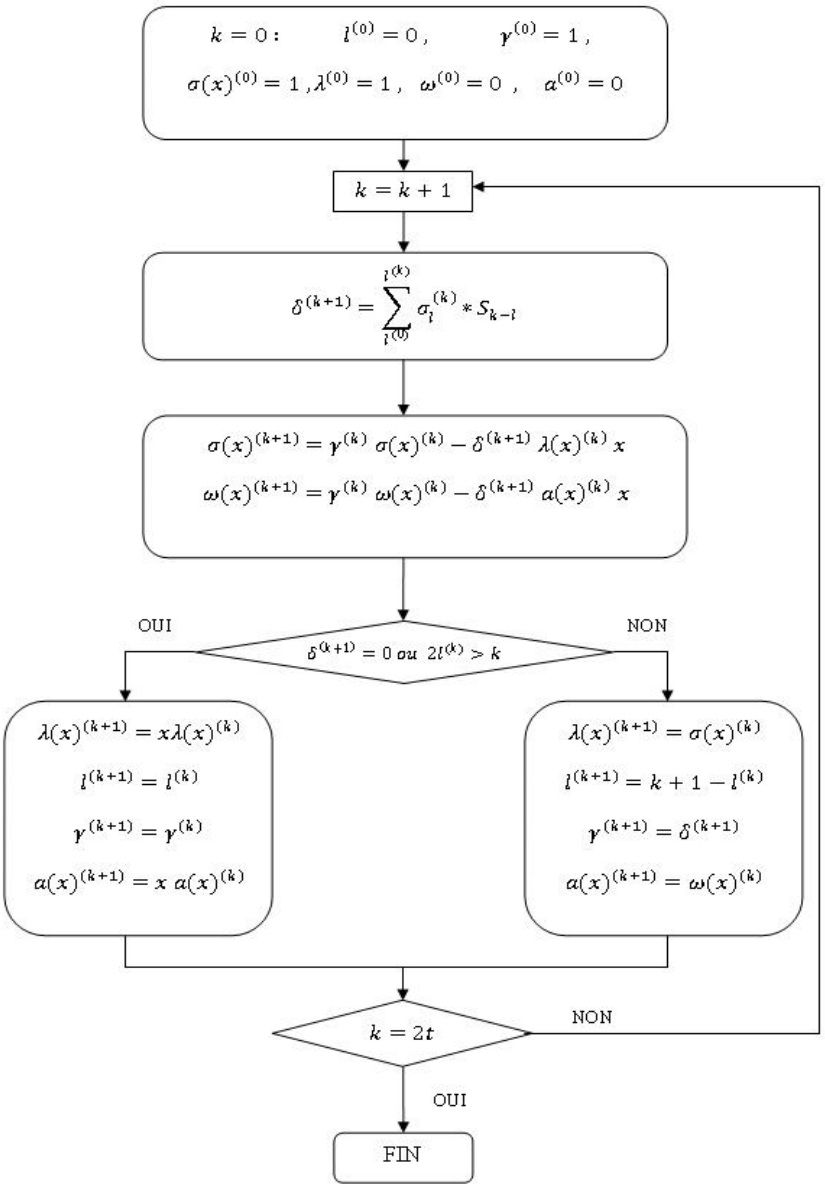

Figure 2.4 – Organigramme de l'algorithme de EIBM.

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 :

$$e_i = \frac{\omega(\alpha^i)}{\sigma\prime(\alpha^i)} \tag{2.17}$$

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 $GF(2^m)$.

2.4 Application des codes de Reed-Solomon dans les systèmes de communication sans fils

les codes de reed solomon sont largement utilisés dans les systèmes de communications sans fil comme le GSM, IS-95 et WLANs (IEEE 802.11), et le Wimax pour le contrôle des erreurs. À titre d'illustration nous présentons dans ce qui suit l'application de ces techniques de codage dans le systme Wimax.

WiMax (Worldwide Interoperability for Microwave Access) est le nom commercial de la technologie sans fil 802.16 pour l'accès au réseau de l'opérateur.

La norme 802.16 a connu de nombreuses évolutions au fur et à mesure qu'elle gagne en popularité. Destinées originellement à desservir les zones les plus éloignées en haut débit en tant que réseau d'accès, cette norme s'oriente de plus en plus vers la mobilité notamment dans la version 802.16e.

2.4.1 La technique FEC dans le WiMax

La technologie de communication numérique sans fil WiMax présente plusieurs avantages par rapport aux systèmes de communication filaires et sans fils traditionnels. Parmi ces avantages nous identifions : une réduction du coût d'installation et du service, une plus grande couverture et débit, une meilleure qualité de service et efficacité spectrale. Ces

avantages exceptionnels sont le résultat de l'intégration de plusieurs techniques avancées à la norme de WiMax. Une de ces techniques est la correction d'erreur directe (forward error correction, FEC). En effet, FEC est l'une des fonctions, les plus sophistiquées et les plus exigeantes en termes de ressources et de calcul, spécifiées au niveau de la couche physique de WiMax.

La technique FEC permet au récepteur de détecter et de corriger les erreurs de transmission et ainsi, cette dernière augmente la capacité du canal de communication. Comme spécifié dans le standard de WiMax, le processus de correction d'erreur directe est réalisé en concaténant l'encodeur Reed-Solomon avec l'encodeur convolutionnel, à l'émetteur. Par conséquent, le décodeur de Viterbi qui effectue le décodage d'une séquence codée avec un encodeur convolutionnel ainsi que le décodeur Reed-Solomon doivent être implémentés au récepteur.

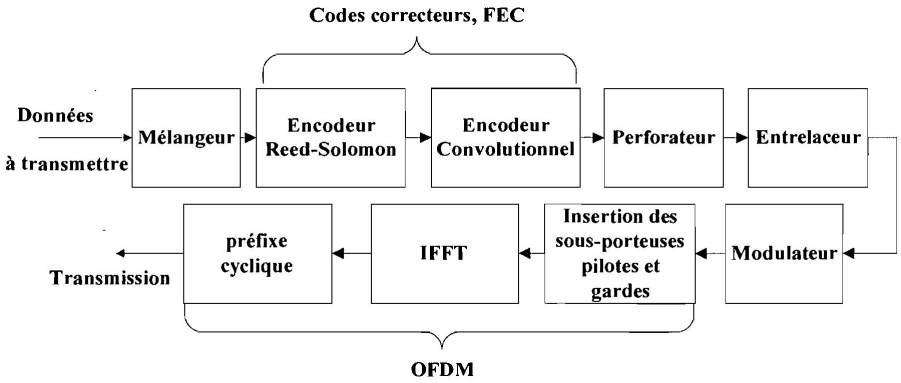

Figure 2.5 – Schéma d'un émetteur WiMax[18].

2.4.2 specification du standard wimax concernant le codes RS

La norme de WiMax stipule que le code RS utilisé doit être dérivé du code $RS(n = 255, k = 239, t = 8)$ utilisant un corps de Galois à 2^8 éléments noté $GF(2^8)$. Pour générer ce code, deux polynômes sont utilisés :
Le polynôme primitif permettant de générer les symboles codés est :

$$p(X) = X^8 + X^4 + X^3 + X^2 + 1 \tag{2.18}$$

Le polynôme générateur de code permettant de calculer les symboles de parité est :

$$\begin{aligned} g(X) &= X^{16} + 118X^{15} + 52X^{14} + 103X^{13} + 31X^{12} + 104X^{11} + 126X^{10} \\ &\quad +187X^9 + 232X^8 + 17X^7 + 56X^6 + 183X^5 + 49X^4 \\ &\quad +100X^3 + 81X^2 + 44X^1 + 79 \end{aligned} \tag{2.19}$$

Pour que ce code puisse traiter des paquets de tailles différentes et avoir des capacités ajustables de correction, il doit être raccourci et poinçonné. Quand un paquet est raccourci à k symboles, un préfixe de 239-k symboles nuls est ajouté au paquet et une fois le processus de codage terminé, ces symboles codés sont supprimés. Quand la fonction de poinçonnage est appliquée sur un mot du code pour corriger t' symboles, seuls les$2t\prime$ symboles de parité parmi les 16 sont utilisés. Ce fonctionnement est illustré sur la figure2.6.

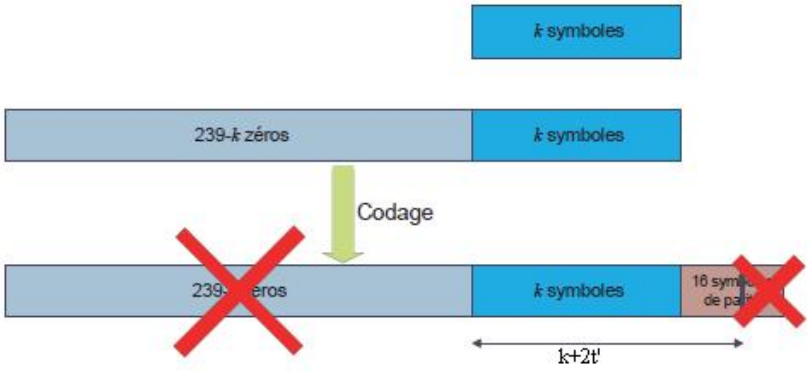

Figure 2.6 – Processus de poinçonnage et de raccourcissement dans un code Reed-Solomon.

Dans ce contexte, k est égale à 24, 36, 48, 72, 96, 108 octets pour les profils QPSK 1/2, QPSK 3/4, 16QAM 1/2 ,16QAM 3/4, 64QAM 2/3, 64QAM 3/4 respectivement. À la sortie des données du codeur RS, la deuxième phase commence. Cette phase est atteinte en deux étapes. La première étape consiste à extraire les premiers 2T' octets de parité à partir des 16 octets de parité générés par le codeur RS. 2 t' est égale à 8, 4, 16, 8, 12,12 pour les profils de modulation QPSK 1/2, QPSK 3/4, 16QAM 1/2 ,16QAM 3/4, 64QAM 2/3, 64QAM 3/4 respectivement. La deuxième étape sert à la réorganisation des octets de façon à ce que les octets de parité ($2t\prime$) soient lus en premier suivis par les octets des données. Les zéros octets sont éliminés. Donc, la taille du bloc valable sera égale à

$k + 2t\prime$.Les profils de modulation et de codage adoptés par le WiMax sont représentés dans le tableau2.3 :

Profil	**Modulation**	**Taille du bloc non codé**	**Code RS (n,k,t)**
0	BPSK	12	(12, 12, 0)
1	QPSK	24	(32, 24, 4)
2	QPSK	36	(40, 36, 2)
3	16QAM	48	(64, 48, 8)
4	16QAM	72	(80, 72, 4)
5	64QAM	96	(108, 96, 8)
6	64QAM	108	(120, 108, 6)

Tableau 2.3 – Les sept profils de modulation et de codage adoptés par le WiMax.

2.5 Conclusion

Nous avons exposé les bases mathématiques de la théorie de codage et décodage de Reed-Solomon, nous avons ensuite présenté les deux algorithmes de décodage RS,l'un est est basé sur le théorème d'Euclid, et l'autre sur les registres à décalage de Berlekamp-Massey.

Nous avons, à la fin, mis en avant l'application de ces types de codage et décodage dans les systèmes de communication sans fils selon la norme WiMax.

Chapitre 3

Architecture de codeur et décodeur de Reed-Solomon

3.1 Introduction

Les principes de codage et décodage de Reed-Solomon ont été décrits dans le chapitre 2. Dans ce chapitre, nous exposerons l'architecture des différents blocs utilisés pour l'implémentation de codeur et décodeur de Reed-Solomon. Chaque bloc sera suivi d'un exemple illustrant les résultats de chaque étape du calcul.

3.2 L'architecture du codeur RS

Selon l'équation clé définissant le codage de Reed-Solomon présenté dans chapitre 2,

$$c(x) = i(x)x^{n-k} + \left[i(x)x^{n-k}\right] mod(g(x))$$

L'implémentation du codeur RS demande deux opérations : un décalage et une division. Ces deux opérations peuvent être effectuées grâce à des registres à décalage et à des multiplexeurs.

La figure3.1 illustre le diagramme en block d'un codeur RS. Ce dernier contient $2t$ éléments mémoires (registre à m bits), où m est le nombre de bits par symbole et t est la capacité de la correction du code RS. Ces registre sont appelés registres de parité où registres de contrôle. Il contient ainsi des additionneurs, des multiplicateurs dans le champ de Galois ($GF(2^m)$) et des multiplexeurs.

Ce circuit effectue la division polynomiale du polynôme d'infirmation $i(x)$ sur le polynôme générateur $g(x)$. Le reste de la division est enregistré dans les 2t registres de parités. Le codage est systématique, on doit effectuer une opération de décalage pour placer les informations dans le degré élevé du mot code de sortie. Le polynôme du mot de code, $c(x)$ est la concaténation de $i(x)$ suivi du reste de la division.

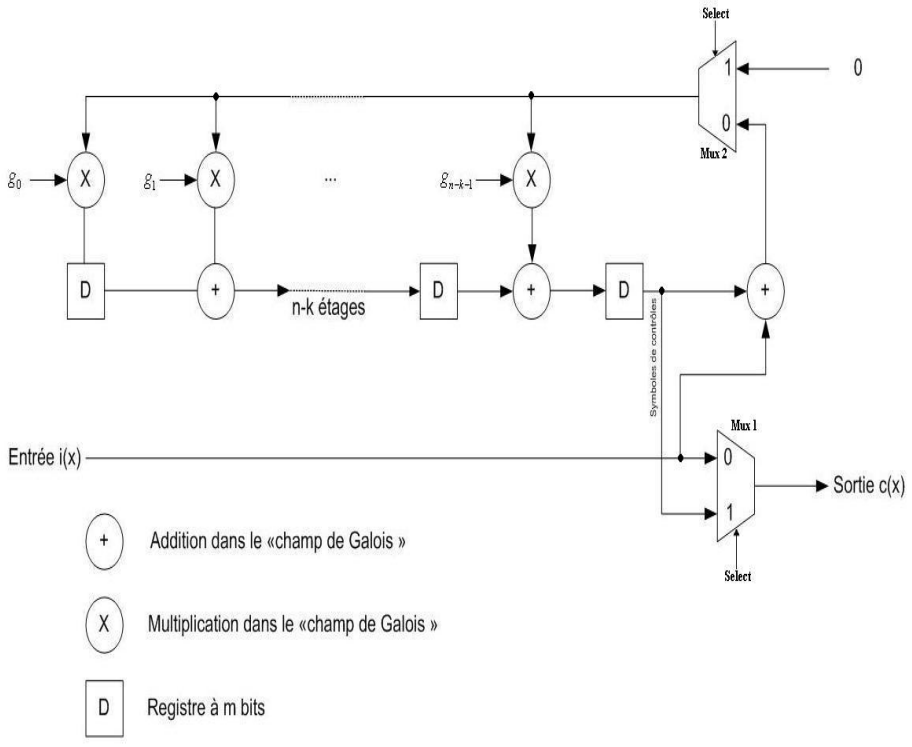

Figure 3.1 – Diagramme en bloc d'un codeur RS.

Les opérations de ce circuit peuvent êtres décrite comme suit :

1. L'état initial de chaque registre de parité est 0.

2. On considère que le message est divisé uniformément en symboles de m bits, où chaque symbole est considéré comme un élément du champ du Galois. Chaque symbole alors est associé avec une puissance croissante de x, en commençant par $x_0 = 1$ et en terminant à x_{k-1}, Ainsi on forme un polynôme $m(x)$ dans $GF(2^m)$. Le polynôme message est de dégrée $k-1$; il a alors k coefficient. À chaque cycle d'horloge, un coefficient entre dans le circuit. Ces coefficients entrent dans le circuit dans l'ordre décroissant en termes de signification.
3. Pour les premières k cycles d'horloge, les symboles de message entrent dans le circuit et ce dernier calcul le reste de la division. La commande « Select » est à 1 pendant les premiers k cycles d'horloge et les données codées correspondent alors au polynôme message (le multiplexeur 2 sélectionne les symboles d'information).
4. Quand tous les symboles d'information sont entrés dans le codeur, la commande « select » est mise à 0. Le multiplexeur 2 sélectionne le symbole nul du champ de Galois ($GF(2^m)$) et le résultat des multiplications est 0. Les résultats des additions, et par conséquent les entrés des registres sont simplement les valeurs des registres précédents. Le reste de la division est décalé vers la sortie. Le multiplexeur 1 sélectionne alors à chaque fois la sortie du dernier registre.

Exemple Soit le code RS(15,9). On veut coder le mot d'information i(x) suivant :

$$i(x) = \alpha^4 x^8 + \alpha x^7 + \alpha^{10} x^6 + \alpha^{14} x^5 + \alpha^6 x^4 + \alpha^8 x^3 + \alpha^{10} x^2 + \alpha^{11} x + \alpha^5$$

Les étapes du calcul du mot code $c(x)$ ainsi le contenu des registres de parité $(D_i; i = 1, \cdots, 6)$ sont montré sur le tableau3.1.

Cycle d'horloge	$i(x)$	D_1	D_2	D_3	D_4	D_5	D_6	$c(x)$
1	α^4	α^{10}	α^{13}	α^{10}	α^8	α^3	α^{14}	α^4
2	α	α^{13}	α^8	0	α^{14}	α^{14}	α^6	α
3	α	α^{13}	α^{12}	α^3	α^{11}	α^8	α^{13}	α^{10}
4	α^{14}	α^8	α^4	α^9	α^2	α^6	α^9	α^{14}
5	α^6	α^{11}	α^6	α^{13}	0	α^{10}	α^{13}	α^6
6	α^8	α^9	1	α^5	α^5	α^2	α^9	α^8
7	α^{10}	α^4	1	α	α	α^{14}	1	α^{10}
8	α^{11}	α^3	α^{12}	α^{14}	0	α^6	α	α^{11}
9	α^5	α^8	α^5	α^8	α^8	α	α^4	α^5
10		0	α^8	α^5	α^9	α^8	α	α^4
11		0	0	α^8	α^5	α^9	α^8	α
12		0	0	0	α^8	α^5	α^9	α^8
13		0	0	0	0	α^8	α^5	α^9
14		0	0	0	0	0	α^8	α^5
15		0	0	0	0	0	0	α^8

Tableau 3.1 – Calcul du mot code pour un polynôme $i(x)$.

3.3 Architecture du décodeur RS

Le schéma de décodage des codes RS est montré sur la figure3.2 . Il comprend le block du syndrome, le block de la résolution de l'équation clé (le block d'Euclid ou le block de Berlekamp-Massey), le block de Chien Search, le block de Forney, le block du message reçu et le block de correction.

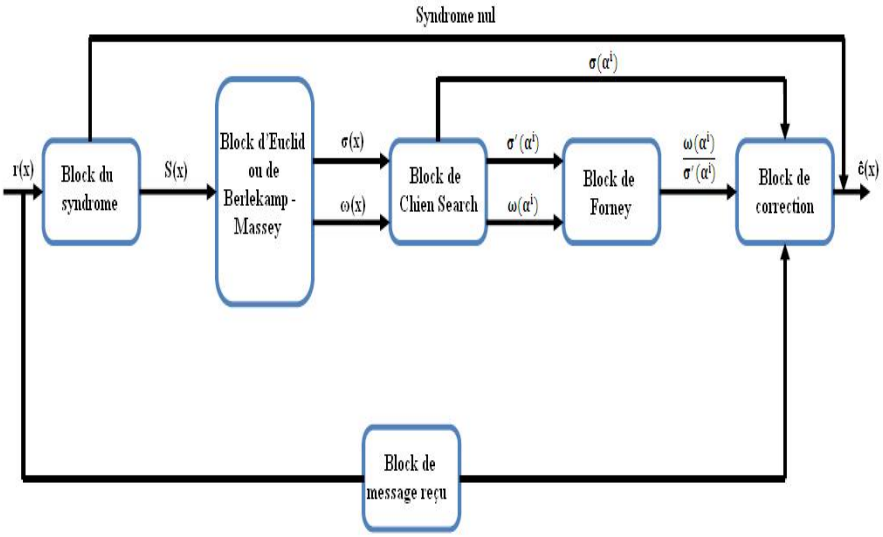

Figure 3.2 – Schéma du décodage.

Avec :

$r(x)$: mot code reçu,

$S(x)$: syndrome calculé,

$\omega(x)$: polynôme d'amplitude des erreurs,

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

$\sigma(\alpha^i)$: polynôme de localisation des erreurs évalué pour tous les éléments compris dans $GF(2^m)$,

$\omega(\alpha^i)$: polynôme d'amplitude des erreurs évalué pour tous les éléments compris dans $GF(2^m)$,

$\frac{\omega(\alpha^i)}{\sigma\prime(\alpha^i)}$: Division entre polynôme d'amplitude évalué et la dérivée du polynôme de localisation des erreurs évaluées,

$\hat{c}(x)$: sortie du décodeur.

3.3.1 Bloc du syndrome

La cellule élémentaire utilisé pour le calcul du syndrome est montré sur la figure3.3 . Ce schéma calcul un symbole du syndrome de façon itérative.

Pour le calcul du syndrome complet, on a besoin de $2t$ cellules comme illustré dans la figure3.4.

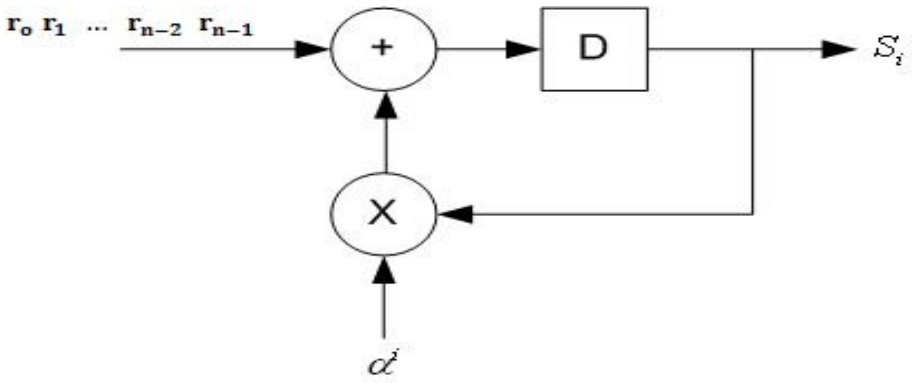

Figure 3.3 – Synoptique du circuit de calcul d'un élément du syndrome.

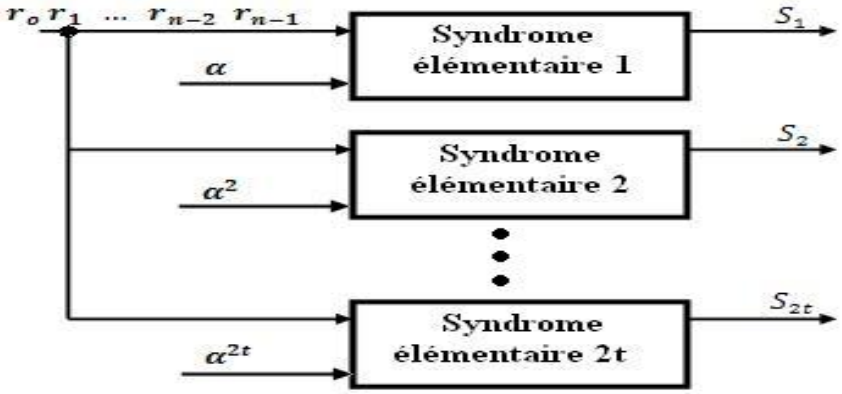

Figure 3.4 – Schéma du circuit global de calcul du syndrome.

Exemple de calcul

Soit le code $RS(15, 9)$.

$$\begin{aligned} r(x) &= \alpha^9x^{14} + \alpha x^{13} + \alpha^{11}x^{12} + \alpha^{14}x^{11} + \alpha^6x^{10} + \alpha^8x^9 + \alpha^{10}x^8 + \alpha^5x^6 \\ &\quad +\alpha^4x^5 + \alpha x^4 + \alpha^8x^3 + \alpha^9x^2 + \alpha^5x^1 + \alpha^8 \end{aligned} \tag{3.1}$$

On propose de calculer la valeur du premier coefficient du syndrome pour un polynôme reçu $r(x)$. Pour cela, on utilise le schéma de la figure3.5.
les résultats de calcul du l'élément S_1 du syndrome sont illustrés dans le tableau3.2

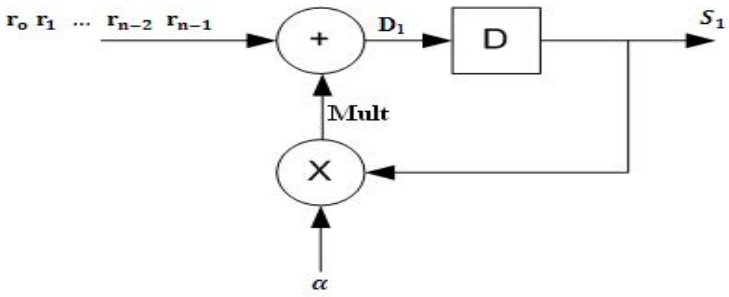

Figure 3.5 – Schéma de calcul de l'élément S_1 du syndrome.

Cycle d'horloge	r_i	Mult	D_1	S_1
1	α^9	0	α^9	α^9
2	α	α^{10}	α^8	α^8
3	α^{11}	α^9	α^2	α^2
4	α^{14}	α^3	1	1
5	α^6	α	α^{11}	α^{11}
6	α^8	α^{12}	α^9	α^9
7	α^{10}	α^{10}	0	0
8	0	0	0	0
9	α^5	0	α^5	α^5
10	α^4	α^6	α^{12}	α^{12}
11	α	α^{13}	α^{12}	α^{12}
12	α^8	α^{13}	α^3	α^3
13	α^9	α^4	α^{14}	α^{14}
14	α^5	1	α^{10}	α^{10}
15	α^8	α^{11}	α^7	α^7

Tableau 3.2 – Tableau : Calcul de l'élément S_1 du syndrome.

Les valeurs de tous les coefficients (S_i) du syndrome sont montrés dans le tableau3.3.

i	**6**	**5**	**4**	**3**	**2**	**1**
α^i	α^6	α^5	α^4	α^3	α^2	α^1
S_i	α^{11}	1	α^{14}	α^6	α^{13}	α^7

Tableau 3.3 – Tableau : Valeurs de tous les éléments du syndrome pour un message reçu $r(x)$.

3.3.2 Bloc du Berlekamp-Massey

La cellule élémentaire qui réalise une itération selon l'algorithme EIBM est montrée sur la figure3.6.

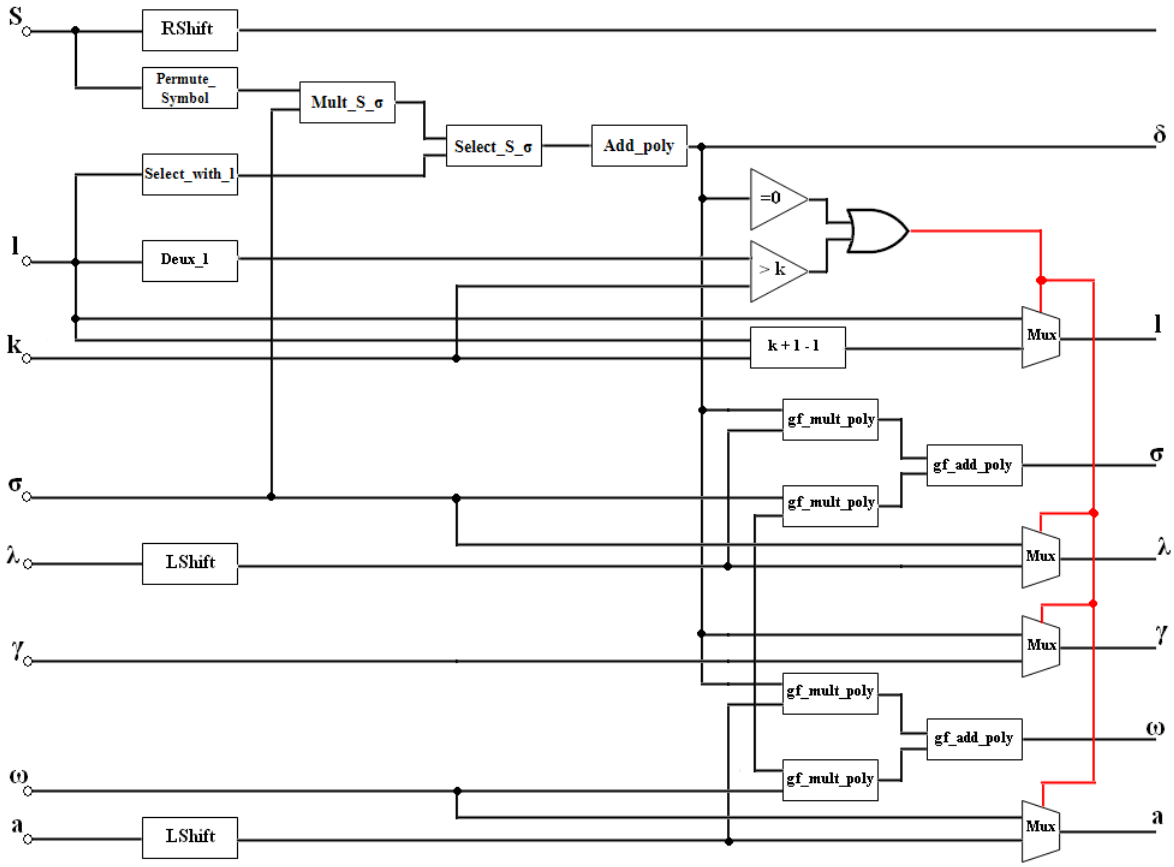

Figure 3.6 – Shéma de la cellule EIBM

Le bloc « RShift » sert à faire décaler les symboles du syndrome à droite. En effet, l'entrée de ce bloc est un signal S de $(3t)$ symboles, chaque symbole est sur m bits (le signal S est alors sur $(3t) * m$ bits). Initialement, les symboles du syndrome sont placés sur les bits du poids forts du signal S, et les autres bits (t*m bits) sont mis à zéro. À chaque itération, une réalise une opération de décalage à droite du signal S. Le nombre total des itérations est (2t). Le bloc « Permute_Symbol » a comme entrée les $(t+1)$ premiers symboles de signal$S$ $((t+1) * m$ bits). Ce bloc réalise une permutation entre les symboles d'entrée ; les symboles du poids forts deviennent ceux du poids faibles et les symboles du poids faibles deviennent ceux du poids forts. On réalise ensuite une multiplication entre la sortie du ce bloc (le bloc « Permute_Symbol ») et le polynôme de localisation des erreurs $\sigma(x)$. Cette opération est effectuée à l'aide du bloc « Mult_S_σ ». Ce bloc réalise une

multiplication dans le champ de Galois ($GF(2^m)$)symbole par symbole comme montré sur la figure3.7.

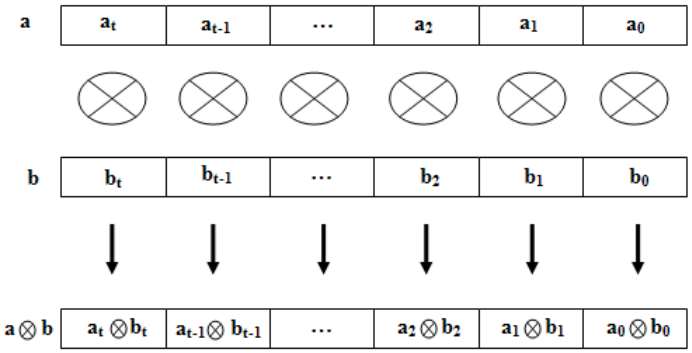

Figure 3.7 – Principe de la multiplication effectuée par le bloc « Mult_S_σ »

Les blocs « Select_with_l » et « Select_S_σ » servent à sélectionner les symboles de la sortie du bloc « Mult_S_σ » à additionner pour trouver la valeur de δ. Le bloc « Select_with_l » fait sortir un signal sur « m bits » tout dépend du signal de son entrée l. Le bloc « Select_with_l » mis à zéro les symboles qui ne devraient pas entrer dans le calcul de δ et garde les autres inchangés. Le bloc « Add_poly» réalise la somme des symboles de la sortie du bloc « Select_S_σ » pour avoir le signal δ.

Pour le calcul du polynôme de localisation des erreurs $\sigma(x)$ et le polynôme d'amplitude des erreurs $\omega(x)$, différents blocs sont nécessaires. Le bloc « LShift » sert à réaliser un décalage à gauche du siganl λ ou b. Ces deux polynômes sont représentés par les deux signaux σ et b respectivement. Ces signaux (σ et b) sont sur ((t+1)*m) bits (concaténation du (t+1) symboles). Le bloc « gf_mult_poly » réalise une multiplication d'un signal sur ((t+1)*m) bits avec un signal sur (m) bits. Le premier signal est la représentation d'un polynôme de degré (t) ((t+1) symboles) et le deuxième est un symbole (il peut être δ ou γ). 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 test ($\delta = 0$) est effectué par le bloc « is_zero ». La sortie de ce bloc est un signal de niveau logique haut quand la condition est vérifiée. Le test ($2l > k$) est réalisé avec

le bloc « Comparateur ». Ce bloc compare la sortie du bloc « Deux_l », qui comme son nom indique, calcul la valeur $2l$, avec le signal k. Le signal k est valeur de l'itération en cours. Il est généré par un compteur, et prend des valeurs de 0 jusqu'au 2t. La sortie du bloc « Comparateur » est mis à 1 quand $2l > k$. Les sorites du bloc « is_zero » et « Comparateur » sont les entrées d'une porte OU. La sortie cette porte sert comme une entrée de sélection pour les multiplexeurs « Mux ». Ainsi, ces multiplexeurs sélectionne les bonnes valeurs des signaux l, λ,γ et b.

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 σ tandis que le polynôme d'amplitude des erreurs est obtenu en prenant le signal ω sans le symbole du poids faible (sans les premiers m bits).

3.3.3 Bloc d'Euclid

L'algorithme d'Euclid modifié est présenté dans le chapitre 2. La cellule élémentaire qui réalise une itération selon cet algorithme est montrée sur la figure3.8.

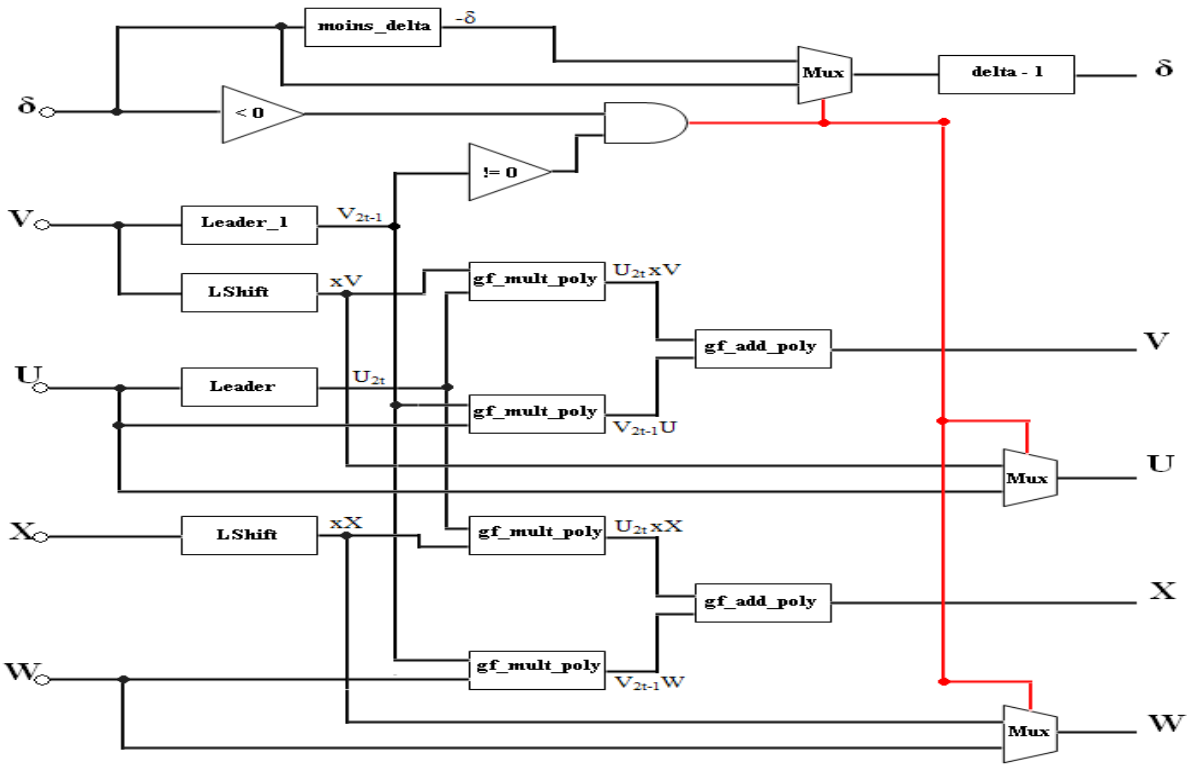

Figure 3.8 – shéma de la cellule d'Euclid

Cette cellule contient divers blocs permettant à chaque itération de déterminer le 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 figure3.9.

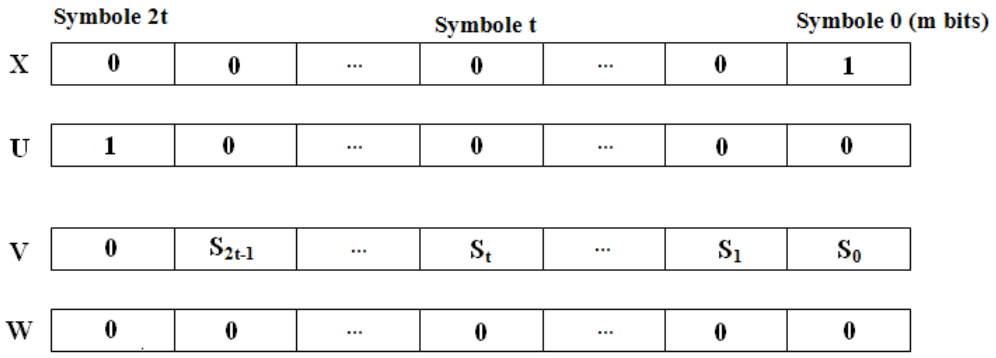

Figure 3.9 – Valeurs initiales des signaux X, U, V et W.

Le bloc « Leader_1 » sert à extraire le symbole V_{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 va leurs 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 Figure 3.10).

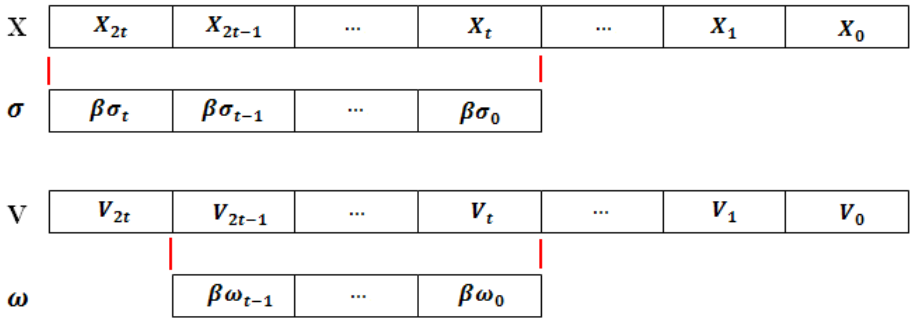

Figure 3.10 – Extraction des polynômes de localisation des erreurs et d'amplitude des erreurs à partir du X et V.

3.3.4 Le bloc du Chien Search

La cellule élémentaire utilisée pour le calcul du syndrome peut être utilisée pour calculer les racines pour un polynôme de localisation des erreurs et pour sa dérivée. Elle peut ainsi être utilisée pour évaluer le polynôme d'amplitude des erreurs.

L'évaluation va se faire pour tous les éléments du champs de Galois ($GF(2^m)$) sauf pour l'élément nul, donc on a besoin de n cellules élémentaires pour évaluer chaque polynôme.

Le synoptique du circuit élémentaire d'évaluation de polynôme de localisation des erreurs est montré sur la figure3.11. Au bout de $(t+1)$ cycles d'horloge, on aura le résultat de l'évaluation.

Pour l'évaluation des autres polynômes (le polynôme d'amplitude des erreurs et la dérivée du polynôme de localisation des erreurs), il suffit de modifier l'entrée de l'addi-

tionneur dans le champ de Galois, et de la remplacer par les symboles de $\omega(x)$ ou ceux de $\sigma\prime(x)$.

Comme on s'intéresse seulement à détecter et corriger les erreurs introduites dans les symboles d'information, on peut évaluer les polynômes $\sigma(x)$, $\sigma'(x)$ et $\omega(x)$ pour k éléments seulement du champ de Galois $(GF(2^m))$, k étant le nombre de symbole du mot d'information. Ces k éléments sont les α^i tel que $i = 1, 2, \cdots, k$. Ainsi, on aura besoin de seulement k cellules élémentaires pour évaluer chaque polynôme.

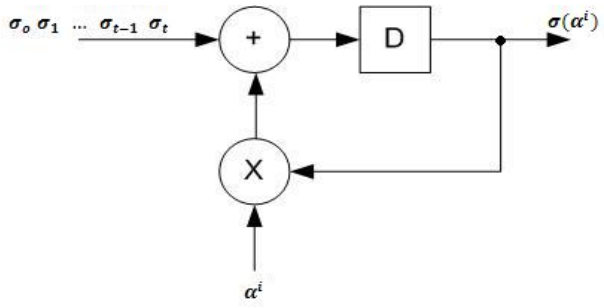

Figure 3.11 – Cellule élémentaire utilisée pour l'évaluation des racines du polynôme de localisation des erreurs.

Exemple de calcul

Soit le polynôme localisateur des erreurs $\sigma(x)$ calculé précédemment par le bloc de Berlekamp-Massey.
On évalue le polynôme $\sigma(x)$ pour α^5 selon le schéma de la figure3.12.

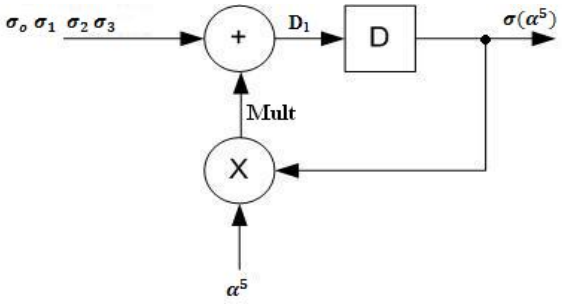

Figure 3.12 – Schéma de calcul de l'élément $\sigma(\alpha^5)$.

Les résultats du calcul sont montrés dans le tableau3.4.

Cycle d'horloge	α_i	**Mult**	D_1	$\sigma(\alpha^5)$
1	α^7	0	α^7	α^7
2	α^4	α^{12}	α^6	α^6
3	α^2	α^{11}	α^9	α^9
4	α^4	α^{14}	α^9	α^9

Tableau 3.4 – Tableau : Calcul de l'élément $\sigma(\alpha^5)$.

Les résultats de l'évaluation des trois polynômes (polynôme de localisation des erreurs, polynôme d'amplitude des erreurs et la dérivée du polynôme de localisation des erreurs) selon le schéma de la figure3.13 sont montrés dans le tableau3.5.

α^i	α^9	α^8	α^7	α^6	α^5	α^4	α^3	α^2	α^1
$\sigma(\alpha)$	0	α^3	0	α^4	α^9	α^4	α^6	0	α^8
$\sigma\prime(\alpha^i)$	α^{11}	α^9	α^{14}	α^8	0	α^{10}	α^3	1	α^4
$\omega(\alpha^i)$	α^{10}	α^3	α^{13}	1	1	α^7	α	α^{11}	α^2

Tableau 3.5 – Tableau : Évaluation des polynômes $\sigma(x)$,$\sigma\prime(x)$ et $\omega(x)$.

3.3.5 Bloc de Forney

Une fois les différentes valeurs de $\sigma\prime(\alpha^i)$ et $\omega(\alpha^i)$ calculées, on applique l'algorithme de Forney.

Une division peut être calculée en faisant une multiplication par l'élément inverse du dénominateur. On crée un ROM avec tous les éléments inverses de $GF(2^m)$ de manière à pouvoir effectuer la division avec une multiplication par un symbole inverse. Le calcul de l'inverse dans un « champ de Galois » est défini comme suit :

$$\alpha^{-i} = \alpha^{n-i} \tag{3.2}$$

Avec : $n = 2^m - 1$ et $i = 1, 2, \cdots, (n-1)$.

Le schéma de la cellule élémentaire utilisée pour le calcul de l'algorithme de Forney est montré sur la figure3.13. k schémas similaires sont nécessaire pour parcourir, à la fois tous lés éléments évalués.

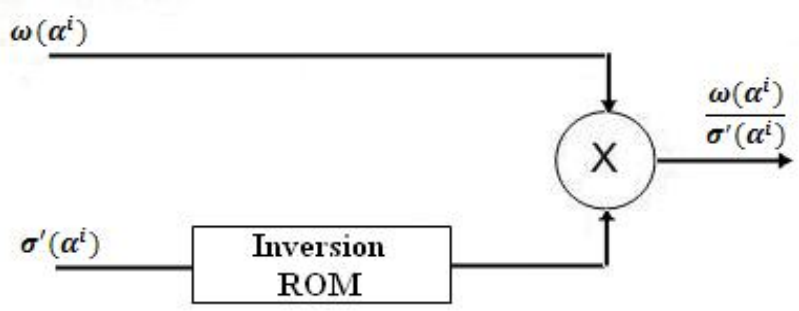

Figure 3.13 – Synoptique du circuit élémentaire utilisé pour l'implémentation de l'algorithme de Forney.

Exemple de calcul de l'algorithme de Forney

Les résultats de calcul de l'algorithme de Forney sont représentés dans le tableau3.6

α^i	α^9	α^8	α^7	α^6	α^5	α^4	α^3	α^2	α^1
$\omega(\alpha^i)$	α^{10}	α^3	α^{13}	1	1	α^7	α	α^{11}	α^1
$\sigma\prime(\alpha^i)$	α^{11}	α^9	α^{14}	α^8	α^{10}	0	α^3	1	α^4
$\sigma\prime(\alpha^i)^{-1}$	α^4	α^6	α^1	α^7	0	α^5	α^{12}	1	α^{11}
$\omega(\alpha^i)/\sigma\prime(\alpha^i)$	α^{14}	α^9	α^{14}	α^7	0	α^{12}	α^{13}	α^{11}	α^{13}

Tableau 3.6 – Tableau : Calcul de l'algorithme de Forney.

3.3.6 Bloc de correction

Le schéma élémentaire de correction est montré sur la figure3.14. La détection du zéro est effectuée avec une porte logique « NOR ». Lorsque l'on a un élément nul à l'entrée, donc une racine, on va avoir un « 1 » à la sortie. Cette détection sert uniquement à sélectionner les éléments pour la correction des erreurs. La porte logique «AND» sert à sélectionner seulement les symboles qui devraient être corrigés. De cette façon on élimine les erreurs du mot-code reçu. L'addition dans le champ de Galois ($GF(2^m)$) donne toujours le symbole du polynôme reçu lorsque la valeur de $\sigma(\alpha^i$ est nul. Quand la valeur de $\sigma(\alpha^i)$ est différente de zéro, on additionne le symbole de e_i avec celui du symbole de mot reçu correspondant (r_i) pour éliminer l'erreur.

Pour effectuer l'opération de correction, k cellules élémentaires sont requises. La sortie du bloc de correction sera commandée par un compteur. Cela assurera que les premières sorties seront les symboles de dégrées le plus hauts.

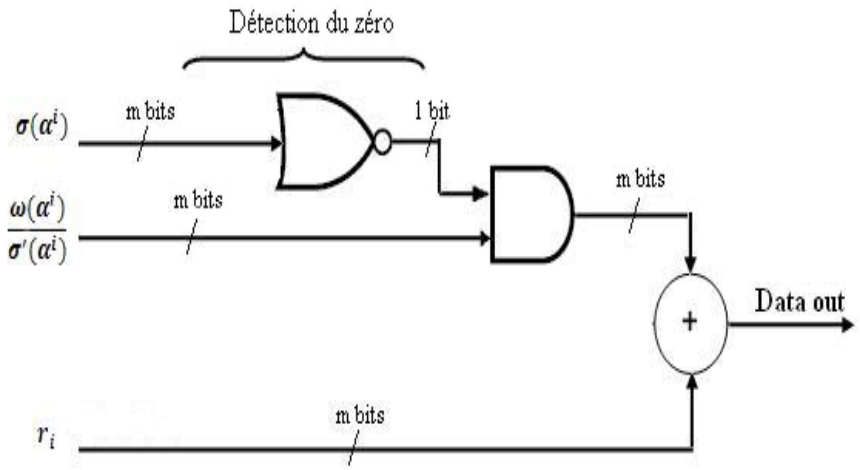

Figure 3.14 – Cellule élémentaire utilisée pour la correction des erreurs.

Exemple de calcul de correction

les résultats de calul des différentes étapes de correction sont représnetés dams le tableau3.7.

α^i	α^9	α^8	α^7	α^6	α^5	α^4	α^3	α^2	α^1
$\sigma(\alpha^i)$	0	α^3	0	α^4	α^9	α^4	α^6	0	α^8
$\omega(\alpha^i)/\sigma\prime(\alpha^i)$	α^{14}	α^9	α^{14}	α^7	0	α^{12}	α^{13}	α^{11}	α^{13}
r_i	α^9	α	α^{11}	α^{14}	α^6	α^8	α^{10}	0	α^5
Data out	α^9	α	α^{10}	α^{14}	α^6	α^8	α^{10}	α^{11}	α^5

Tableau 3.7 – Tableau : Calcul de la correction.

3.4 Conclusion

Dans ce chapitre, nous avons présentés l'architecture de codeur et décodeur de Reed-Solomon en décrivant le fonctionnement des différents blocs qui les constituent. Nous avons illustrer chaque bloc par un exemple de calcul pour mieux expliquer son fonctionnement.

Le chapitre suivant sera consacré pour l'implémentation de cette architecture sur circuit reconfigurable.

Chapitre 4

Simulation, implémentation et résultats

4.1 Introduction

Nombreuse applications surtout dans le domaine des communications sans fils évoluent vers des systèmes miniaturisés à hautes performances, les circuits reconfigurables fournissent une plateforme flexible et efficace pour répondre aux exigences des conceptions complexes en termes de vitesse d'opération, d'espace occupé, de consommation d'énergie, du coût de conception et de temps de mise sur le marché.

Dans ce chapitre, on développera dans un premier temps l'implémentation des opérations de bases dans les corps de Galois. Ensuite, on donnera celle des différents blocs utilisés soit dans le codeur ou bien dans le décodeur.

4.2 Présentation De La Carte FPGA

Nous avons utilisé les cartes FPGA suivantes :

- Le circuit XC2V3000-4FG676 de la famille Virtex II de Xilinx. Ce circuit est intégré autour d'une carte multimédia Celoxica RC203E.
- Le circuit XC2V1000-4FG456C de la famille Virtex II. Ce circuit est intégré autour d'une carte Memec DS-BD-V2MB1000.

Architecture du Virtex II

Le FPGA Virtex 2 est principalement composé d'une matrice de CLBs. Chaque bloc configurable est composé de 4 éléments identiques appelés slices et de deux buffers. Les slices sont identiques et contiennent :

- Deux générateurs de fonctions (composés de LUTs à quatre entrées).
- Deux éléments de mémorisation (bascules D).
- Des portes logiques.
- Des multiplexeurs.
- Une chaîne de cascade horizontale (portes OR).

Chaque bloc est connecté à une matrice de commutation pour accéder aux ressources de routage général.

Les blocs mémoires sont de 18kb assignables de 16k * 1bit à 512*36bits. Chaque mémoire possède deux ports synchrones et indépendants l'un de l'autre. A chaque bloc mémoire est associé un multiplieur 18*18 optimisé pour des opérations sur le bloc mémoire. Chaque multiplieur et bloc mémoire est relié à quatre matrices de commutation pour accéder aux ressources de routage.

Les blocs DCM (Digital Clock Multiplexer) et le multiplexeur d'horloge global fournissent des fonctionnalités d'horloges évoluées. Au total il y a 12 DCM sur l'ensemble du FPGA.

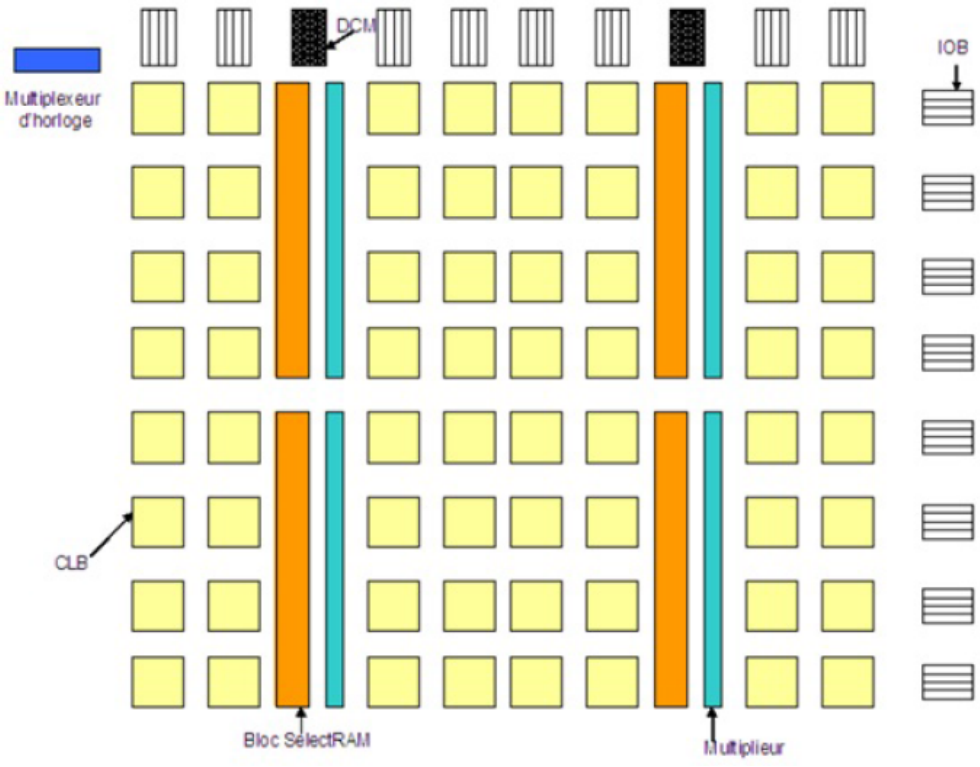

Figure 4.1 – Architecture interne d'un FPGA.

4.3 L'outil ISE de Xilinx

ISE Foundation 9.2 est un environnement intégré de développement de systèmes numériques ayant pour but une implémentation matérielle sur FPGA de la compagnie Xilinx.

Les designs peuvent être décrits sous trois formes principales :

- Schémas.
- Langage de description matérielle (HDL) comme VHDL et Verilog.
- Diagrammes d'états.

ISE intègre donc différents outils permettant de passer à travers tout le flot de conception d'un système numérique :

- Un éditeur de textes, de schémas et de diagrammes d'état.
- Un compilateur VHDL/Verilog.
- Un outil de simulation.
- Des outils pour la gestion des contraintes temporelles.
- Des outils pour la vérification.
- Des outils pour l'implémentation sur FPGA.

4.4 Implémentation des opérations dans les corps de Galois

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

4.4.1 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. Les codes VHDL de l'addition dans les $GF(2^4)$ et $GF(2^8)$ sont présentés dans l'annexe.

la cellule élémentaire pour le bloc de l'addition dans les gf(2^4) est représentée sur la figure4.2

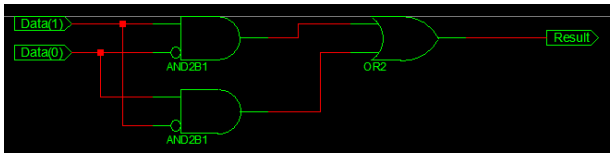

Figure 4.2 – cellule élémentaire pour le bloc de l'addition dans les GF(2^4)

4.4.2 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. Les codes VHDL de la multiplication dans les $GF(2^4)$ et $GF(2^8)$ sont présentés dans l'annexe.

la cellule élémentaire pour le bloc de la multiplication dans les gf(2^4) est représentée sur la figure 4.3

4.4.3 Implémentation de l'inversion dans $GF(2^m)$

L'inversion « dans les corps de Galois » est difficile à simplifier en équations booléens. Pour la réaliser on a utilisé une ROM de taille $2m$. Les codes VHDL de la multiplication

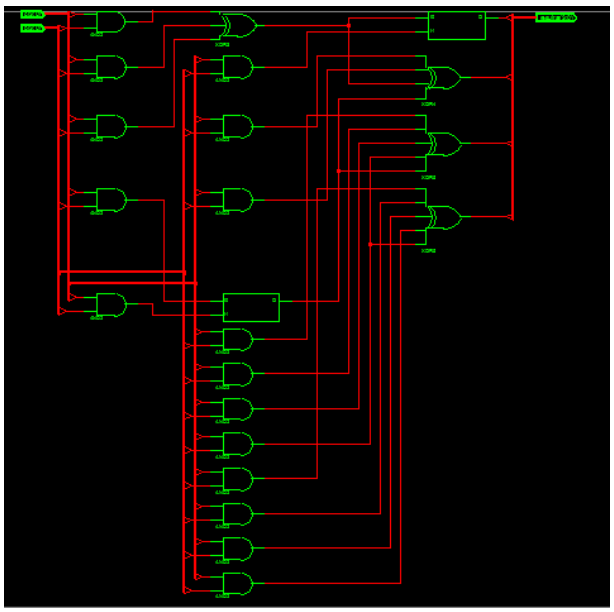

Figure 4.3 – cellule élémentaire pour le bloc de la multiplication dans les gf(2^4)

dans les $GF(2^4)$ et $GF(2^8)$ sont présentés dans l'annexe. Le bloc de l'inversion dans les GF(2^4) est représenté sur la figure 4.40

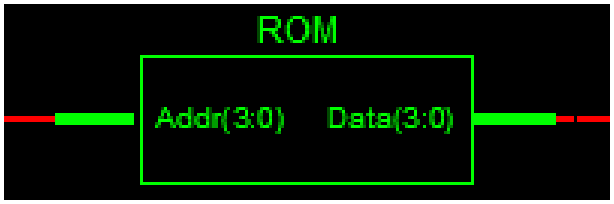

Figure 4.4 – Le bloc de l'inversion dans les GF(2^4)

4.5 Implémentation de codeur de Reed-Solomon

Le schéma de codeur RS$(15, 9)$ est celui représenté sur la figure4.5.

Figure 4.5 – Schéma de bloc du codeur RS$(15, 9)$

4.5.1 Signaux de commande et d'entrée/sortie du codeur

Les signaux utilisés par le codeur se divisent en deux catégories différentes :

- les signaux de commandes.
- les signaux d'entrée/sortie.

Les signaux de commande représentent tous les signaux permettant au codeur de fonctionner correctement. Ces signaux sont :

- clk : signal d'horloge (std_logic).
- commande_data : signal qui permet d'activer le codeur (std_ logic), c'est une impulsion d'un cycle d'horloge qui précède le mot d'information.
- Reset : signal permettant la remise à zéro du codeur.(std_logic).

Les signaux d'entrée /sorties sont :

- donnees_entree : signal d'entrée des symboles à coder. Cette entrée est sur 4 bits pour le codeur RS(15,9) (std_logic_vector(3 downto 0)) et sur 8 bits pour le codeur RS(255,239). (std_logic_vector(7 downto 0)).
- donnees_sortie : signal de sortie des symboles codés. Cette sortie est sur 4 bits pour le codeur RS(15,9) (std_logic_vector(3 downto 0)) et sur 8 bits pour le codeur RS(255,239). (std_logic_vector(7 downto 0)).

4.5.2 Fonctionnement du codeur $RS(n, k)$

Quand le signal commande_data arrive, les registres de parités sont mis à 0. Ensuite, pendant les premiers k ($k = 9$ pour le RS$(15, 9)$ et $k = 239$ pour le RS$(255, 239)$) cycles d'horloges les symboles d'information entrent dans le codeur et ce dernier calcul le reste de la division. À ce niveau le signal « donnees_sortie » correspond aux symboles d'information. Pour les $(n - k)$ cycles d'horloges le signal « donnees_sortie » correspond aux symboles de parités calculés.

4.5.3 Simulation du codeur RS$(15, 9)$

La figure4.6 représente un exemple de simulation de codeur RS$(15, 9)$

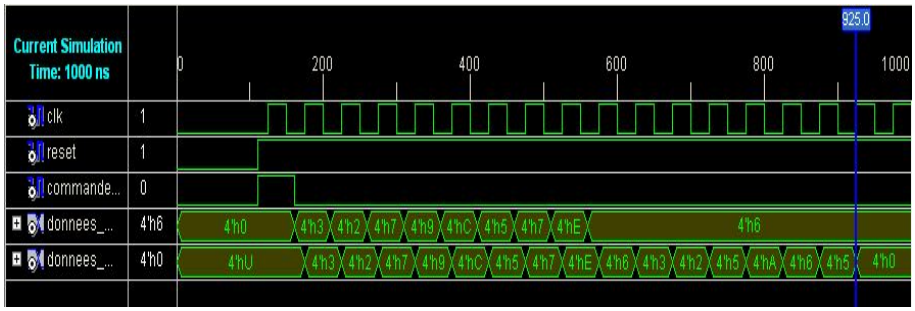

Figure 4.6 – Exemple de simulation du codeur RS(15,9) ;

4.5.4 Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour un codeur RS$(15, 9)$ et RS$(255, 239)$ après placement et routage sous Xilinx Virtex $2xc2v3000 - 4fg676$ sont représentées respectivement dans les tableaux 4.1 et 4.6 :

FPGA Device	**Virtex 2**		**xc2v3000-4fg676**	
Code	**RS(15,9)**		**RS(255,239)**	
Type de ressources	Utilisé	Taux d'utilisation	Utilisé	Taux d'utilisation
Slice Flip Flop	31 de 28672	1%	139 de 28672	1%
4 inputs LUTs	49 de 28672	1%	231 de 28672	1%
Bonded IOBs	11 de 684	1%	19 de 684	2%
Total occuped slices	28 de 14336	1%	128 de 14336	1%

Tableau 4.1 – Ressource hardware utilisé sous Xilinx Virtex 2 xc2v3000-4fg676 pour le bloc du codeur RS$(15, 9)$ et RS$(255, 239)$.

FPGA Device	**Virtex 2xc2v3000-4fg676**	
Code	**RS(15,9)**	**RS(255,239)**
Vitesse	-4	-4
Période minimum	4.175 ns	5.346 ns
Fréquence maximale	239.492MHz	187.038 MHz

Tableau 4.2 – Performance temporelle pour le bloc du codeur RS$(15, 9)$ et RS$(255, 239)$.

4.6 Implémentation du décodeur RS$(15, 9)$

4.6.1 Implémentation de bloc du syndrome

Structure du syndrome

Le bloc de syndrome présenté dans le chapitre 3 est réalisé sous le circuit de syndrome est réalisé sous forme d'un seul fichier décrivant les connexions entre les différents blocs déclarés comme composants décrits dans des fichiers indépendants. Il contient aussi la machine d'état permettant de passer d'une étape à l'autre. Ces états sont idle (état initial) et cacul_du_syndrome, transfert_du_syndrome . Elle est représentée sur la figure 4.8
La structure est celle décrite dans la Figure4.7

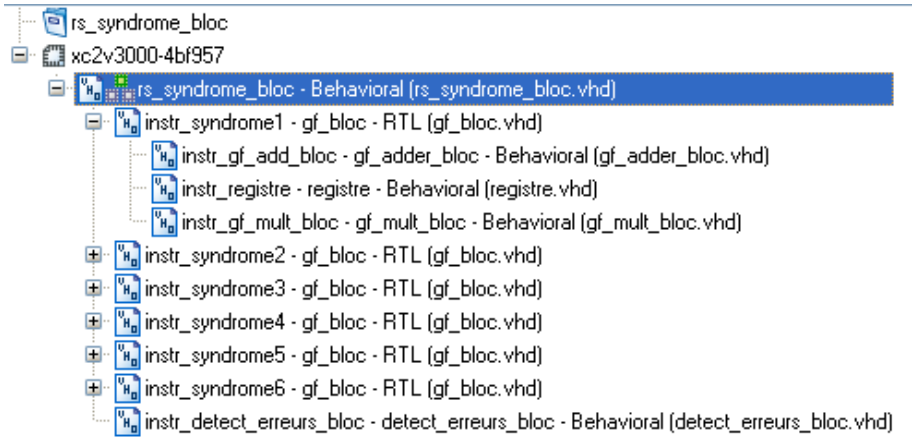

Figure 4.7 – Structure du syndrome

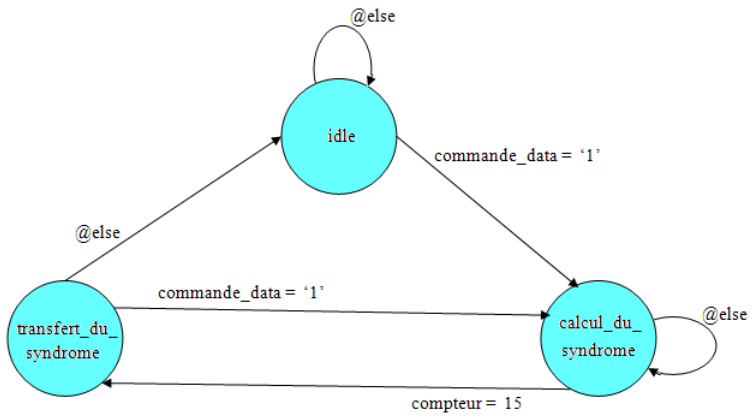

Figure 4.8 – Machine d'état du module de syndrome

Le bloc qui calcule les symboles du polynôme du syndrome est représenté sur la figure4.9 :

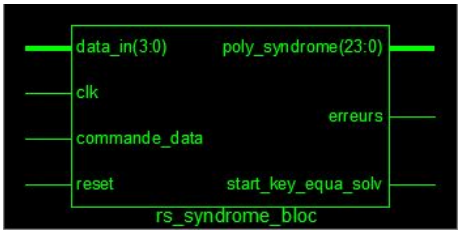

Figure 4.9 – Schéma de bloc du syndrome.

Signaux de commande et d'entrée/sortie de bloc du Syndrome

- clk : signal d'horloge (std_logic)
- commande_data : signal qui permet d'activer le syndrome (std_logic), c'est une impulsion d'un cycle d'horloge qui précède le mot d'information.
- Reset : signal permettant la remise à zéro du syndrome (std_logic).
- data_in : signal d'entrée des symboles à coder. Cette entrée est sur 4 bits (std_logic_vector(3 downto 0)).
- poly_syndrome : signal de sortie de polynôme du syndrome. Cette sortie est sur 24 bits (std_logic_vector(23 downto 0)).

Le bloc du syndrome contient six «6» cellules élémentaires comme celle représentée dans la figure4.10. Ces cellules respectent parfaitement l'architecture décrite dans le chapitre3.

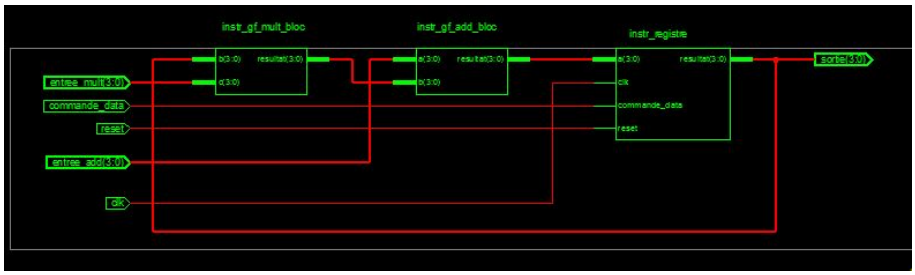

Figure 4.10 – cellule élémentaire pour le calcul du syndrome.

Le bloc « detect_erreurs_bloc» sert à détecter la présence d'éventuelles erreurs. La sortie est mise à '0' lorsque tous les syndromes sont nuls.

Simulation du syndrome

Un exemple de simulation de bloc du syndrome est représenté sur la figure4.11

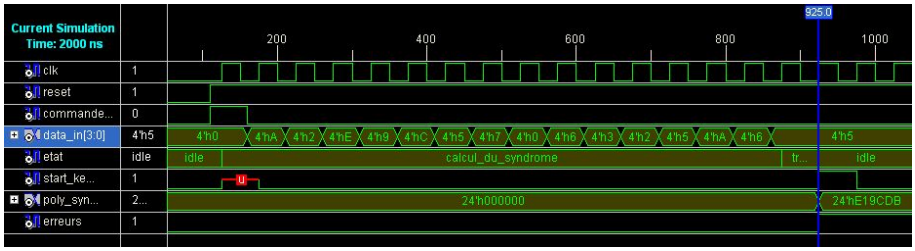

Figure 4.11 – Exemple de simulation de bloc du syndrome.

Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour le bloc du syndrome RS$(15, 9)$ après placement et routage sous Xilinx Virtex 2 xc2v3000-4fg676 sont représentées respectivement dans les tableaux 4.3 et 4.4 :

FPGA Device	**Virtex 2 xc2v3000-4fg676**	
Type de ressources	Utilisé	Taux d'utilisation
Slice Flip Flop	33 de 28672	1%
4 inputs LUTs	44 de 28672	1%
Bonded IOBs	33 de 484	6%
Total occuped slices	26 de 14336	1%

Tableau 4.3 – Ressource hardware utilisé sous Xilinx Virtex 2 xc2v3000-4fg676 pour le bloc du syndrome.

FPGA Device	**Virtex 2xc2v3000-4fg676**
Vitesse	-4
Période minimum	4.175 ns
Fréquence maximale	239.492 MHz

Tableau 4.4 – Performance temporelle pour le bloc du syndrome.

4.6.2 Implémentation de bloc du Berlekamp-Massey

Structure de bloc Berlekamp-Massey

Le circuit de bloc du Berlekamp-Massey présenté dans le chapitre 3 est composé d'un seul fichier qui contient les connexions entre les différents blocs déclarés comme composant.

Ces derniers sont décrits dans des fichiers indépendants.Ce circuit est commandé par une machine d'état représenté sur la figure 4.13 La structure de bloc du Berlekamp-Massey est celle représenté sur la figure4.12

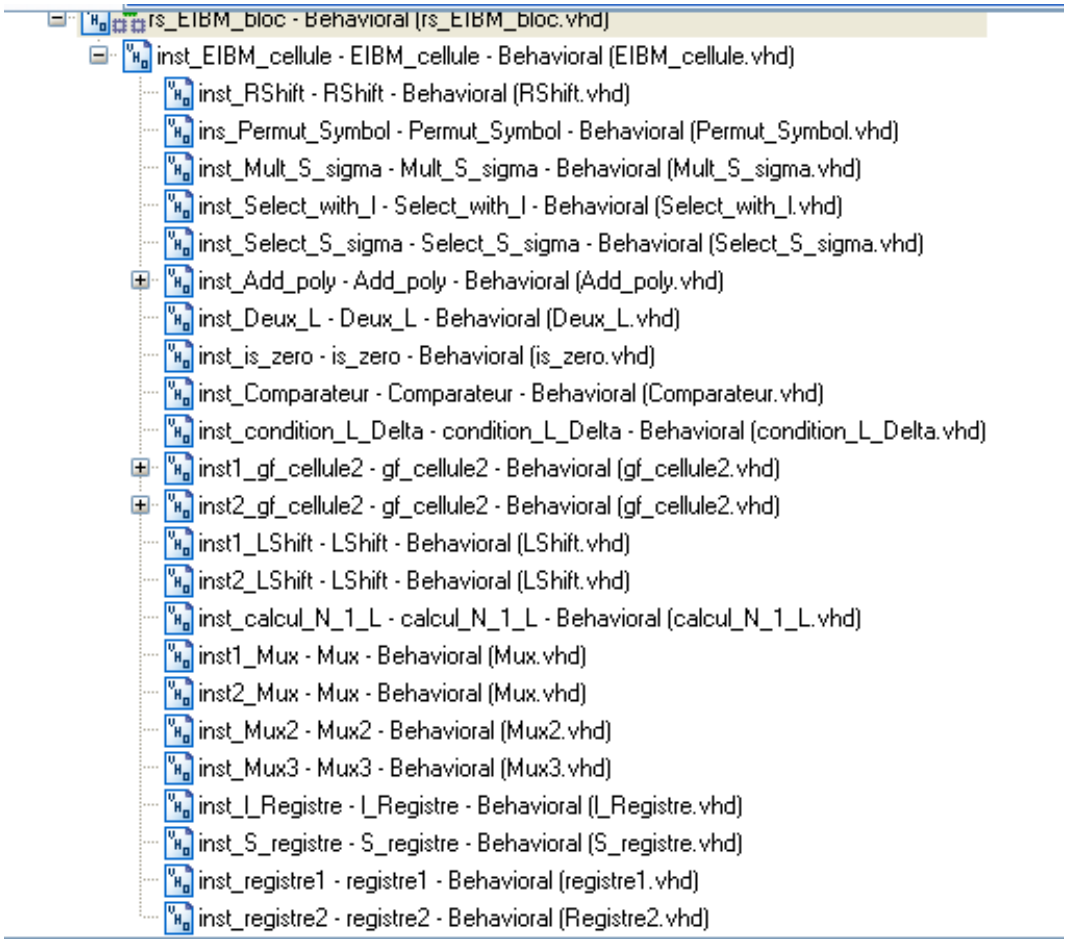

Figure 4.12 – Structure de bloc du Berlekamp-Massey.

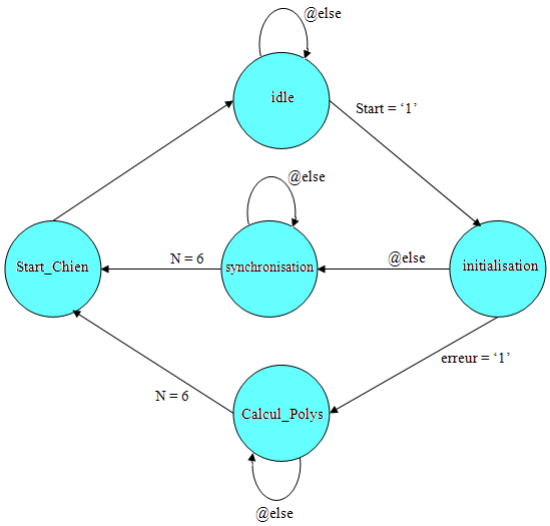

Figure 4.13 – Machine d'état de module du Berlekamp-Massey.

La cellule élémentaire pour le calcul de bloc du Berlekamp-Massey "EIBM" est représentée sur la figure4.14.

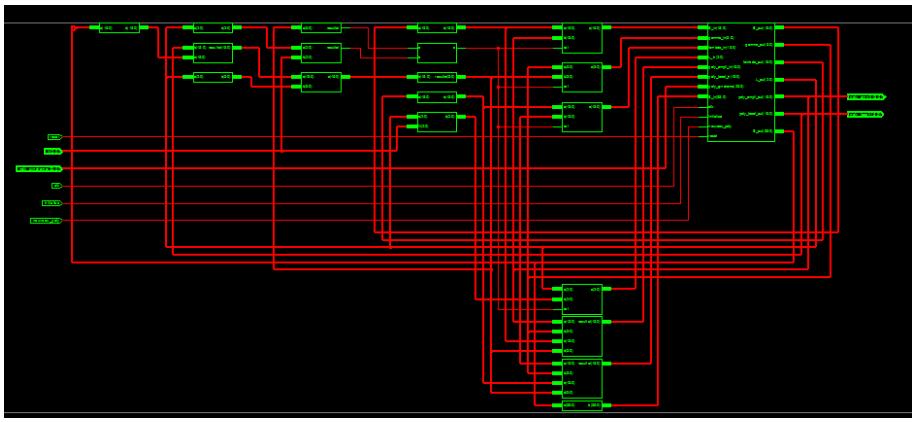

Figure 4.14 – Cellule élémentaire pour le calcul de bloc du Berlekamp-Massey.

Le bloc qui calcule le polynôme d'amplitude et de localisation des erreurs suivant l'algorithme de Berlekamp-Massey "EIBM" est :

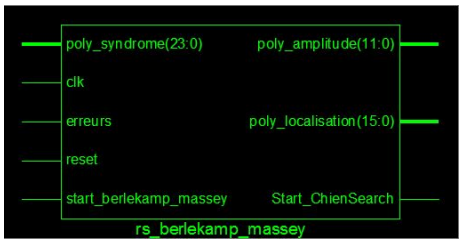

Figure 4.15 – Schéma de bloc du Berlekamp-Massey.

Signaux de commande et d'entrée/sortie de bloc du Berlekamp-Massey.

- clk : signal d'horloge (std_logic)
- start_berlekamp_massey : signal qui permet d'activer le bloc du berlekamp-massey (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 du Berlekamp-Massey (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 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 du Berlekamp-Massey

La figure4.16 représente un exemple de simulation de bloc du berlekamp-massey.

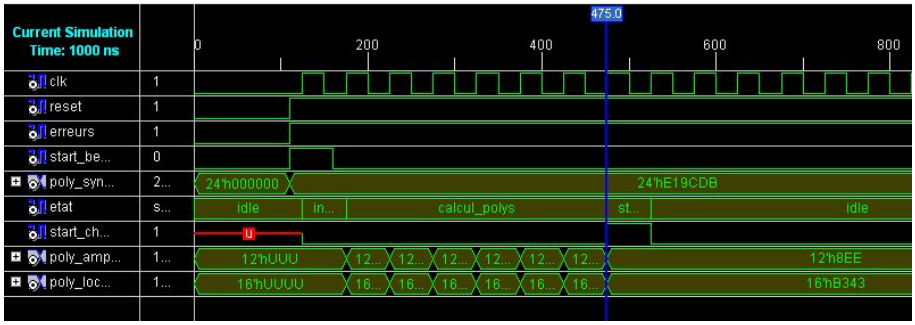

Figure 4.16 – Exemple de simulation de bloc du Berlekamp-Massey.

Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour le bloc du Berlekamp-Massey RS$(15, 9)$ après placement et routage sous Xilinx Virtex 2 xc2v3000-4fg676 sont représentées respectivement dans les tableaux 4.5 et 4.6 :

FPGA Device	**Virtex 2 xc2v3000-4fg676**	
Type de ressources	Utilisé	Taux d'utilisation
Slice Flip Flop	109 de 28672	1%
4 inputs LUTs	317 de 28672	1%
Bonded IOBs	57 de 484	11%
Total occuped slices	170 de 14336	1%

Tableau 4.5 – Ressource hardware utilisé sous Xilinx Virtex 2 xc2v3000-4fg676 pour le bloc du Berlekamp-Massey.

FPGA Device	**Virtex 2xc2v3000-4fg676**
Vitesse	-4
Période minimum	8.416 ns
Fréquence maximale	118.821 MHz

Tableau 4.6 – Performance temporelle pour le bloc du Berlekamp-Massey.

4.6.3 Implémentation de bloc d'Euclid

Structure de bloc d'Euclid

La structure de bloc d'Euclid présenté dans le chapitre 3 est celle décrite dans la Figure4.17

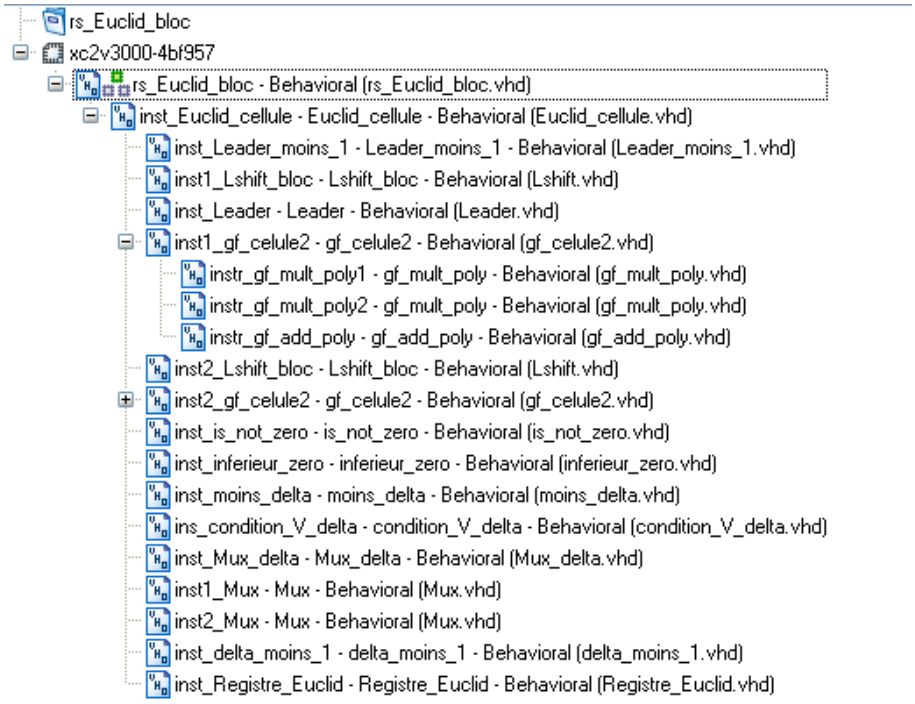

Figure 4.17 – Structure de bloc d'Euclid.

La cellule élémentaire pour le calcul de bloc d'Euclid "MEA" est représentée sur la figure4.18.

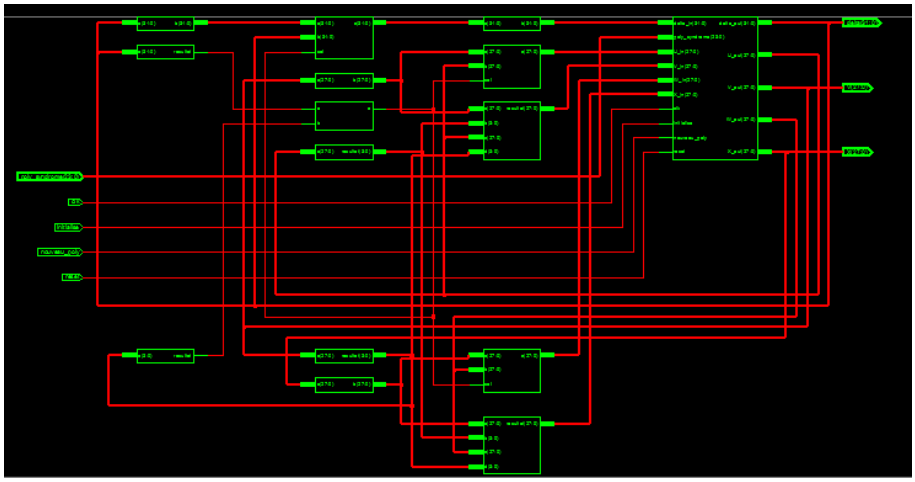

Figure 4.18 – Cellule élémentaire pour le calcul de bloc d'Euclid.

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

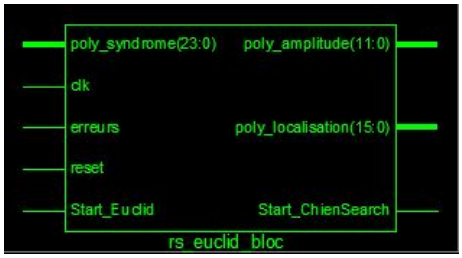

Figure 4.19 – Schéma de bloc d'Euclid.

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 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 figure4.20

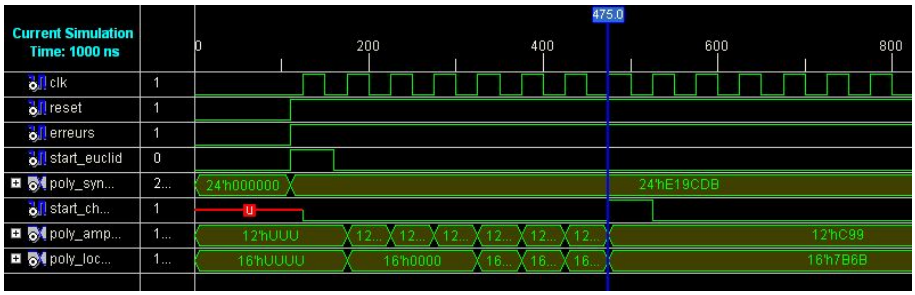

Figure 4.20 – Exemple de simulation de bloc d'Euclid.

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 2xc2v3000-4fg676 sont représentées respectivement dans les tableaux 4.7 et 4.8 :

FPGA Device	**Virtex 2 xc2v3000-4fg676**	
Type de ressources	Utilisé	Taux d'utilisation
Slice Flip Flop	134 de 28672	1%
4 inputs LUTs	400 de 28672	1%
Bonded IOBs	57 de 484	11%
Total occuped slices	229 de 14336	1%

Tableau 4.7 – Ressource hardware utilisé sous Xilinx Virtex 2 xc2v3000-4fg676 pour le bloc d'Euclid.

FPGA Device	Virtex 2xc2v3000-4fg676
Vitesse	-4
Période minimum	8.035 ns
Fréquence maximale	124.456 MHz

Tableau 4.8 – Performance temporelle pour le bloc d'Euclid.

4.6.4 Implémentation de bloc du chiensearch

Structure de bloc du chiensearch

Le bloc de chiensearch est présenté dans le chapitre 3. Son circuit est réalisé sous forme structurel. Il fait appel à trois principaux blocs : « eval_local », « eval_local_prime » et « eval_ampli ». Ces derniers sont aussi implémentés avec des différents blocs déclarés comme composants décrits dans des fichiers indépendants.

Une machine d'états est décrite dans le bloc principal (chiensearch_bloc). Elle génère les différents signaux de commande et assure le passage d'une étape à l'autre.

La structure est celle décrite dans la Figure4.21

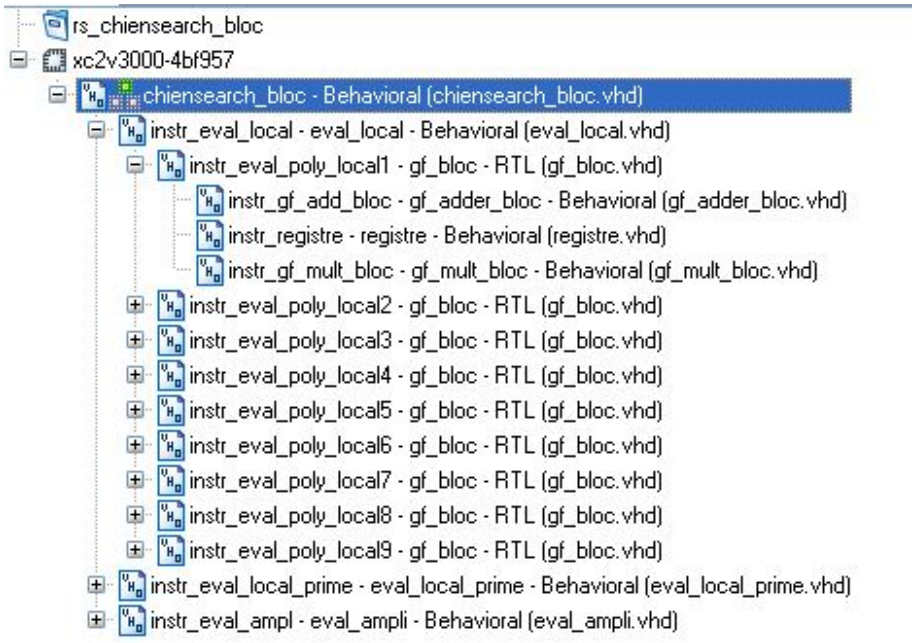

Figure 4.21 – Structure de bloc du chiensearch.

Le bloc qui calcule l'évaluation de polynôme d'amplitude ainsi que de polynôme localisation des erreurs et sa dérivé est :

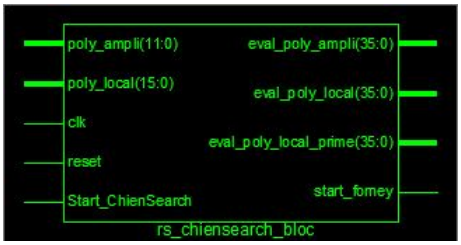

Figure 4.22 – Schéma de bloc du chiensearch.

Signaux de commande et d'entrée/sortie de bloc du chiensearch

Les signaux d'entrée/sortie sont :

- clk : signal d'horloge (std_logic).
- Start_chienSearch : signal qui permet d'activer le bloc du chienSearch (std_logic), c'est une impulsion d'un cycle d'horloge qui précède les polynômes de localisation et d'amplitude des erreurs.
- reset : signal permettant la remise à zéro du syndrome (std_logic).
- poly_amplitude : signal d'entrée de polynôme d'amplitude des erreurs. Cette sortie est sur 12 bits (std_logic_vector(11 downto 0)).
- poly_localisation : signal de d'entrée de polynôme de localisation des erreurs. Cette sortie est sur 16 bits (std_logic_vector(15 downto 0)).
- Start_forney : signal de sortie qui permet l'activation du bloc de forney (std_logic).
- eval_poly_ampl : signal de sortie de l'évaluation de polynôme d'amplitude des erreurs. Cette sortie est sur 36 bits (std_logic_vector (35 downto 0)).
- eval_poly_local : signal de sortie de l'évaluation de polynôme de localisation des erreurs. Cette sortie est sur 36 bits (std_logic_vector (35 downto 0)).
- eval_poly_local_prime : signal de sortie de l'évaluation de la dérivé de polynôme de localisation des erreurs. Cette sortie est sur 36 bits (std_logic_vector (35 downto 0)).

La cellule élémentaire pour le calcul de bloc de chiensearch est représentée dans la figure4.23. Elle correspond à celle décrite dans le chapitre 3.

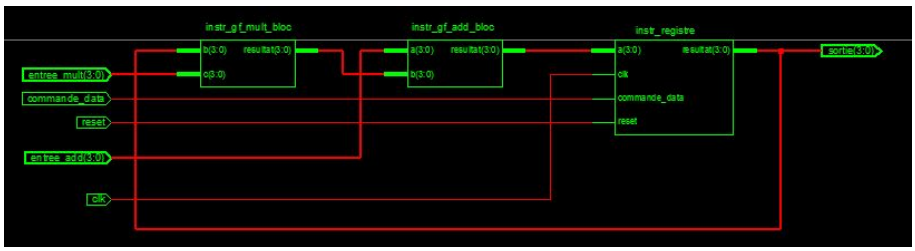

Figure 4.23 – cellule élémentaire pour le calcul du chiensearch.

Simulation de bloc du chiensearch

Des exemples de simulation de bloc du chiensearch en utlisant les polynômes de localisation et d'amplitude des erreurs calculés par l'algorithmes de berlekamp-massey et d'euclid sont représentés sur les figures 4.24 et 4.25 respectivement.

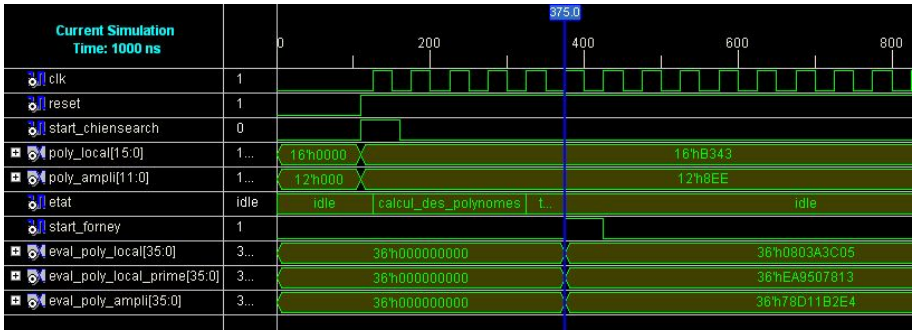

Figure 4.24 – Exemple de simulation de bloc du chiensearch avec le polynôme de localisation et d'amplitude des erreurs calculés par l'algorithme de Berlekamp-Massey.

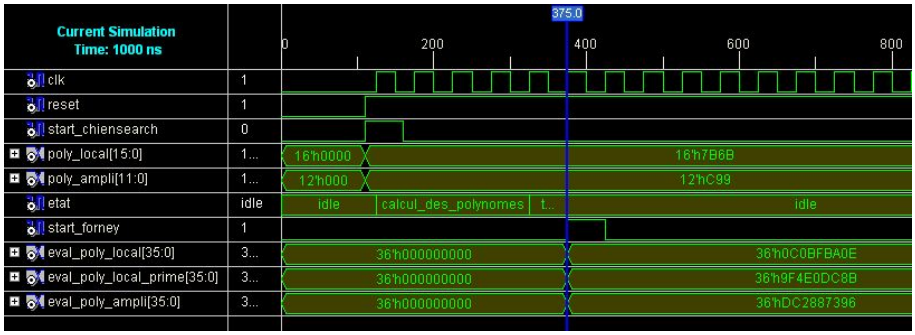

Figure 4.25 – Exemple de simulation de bloc du chiensearch avec le polynôme de localisation et d'amplitude des erreurs calculés par l'algorithme d'Euclid.

Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour le bloc du chiensearch RS(15,9) après placement et routage sous Xilinx Virtex 2 xc2v3000-4fg676 sont représentées respectivement dans les tableaux 4.9 et 4.10 :

FPGA Device	Virtex 2 xc2v3000-4fg676	
Type de ressources	Utilisé	Taux d'utilisation
Slice Flip Flop	163 de 28672	1%
4 inputs LUTs	200 de 28672	1%
Bonded IOBs	140 de 484	28%
Total occuped slices	105 de 14336	1%

Tableau 4.9 – Ressource hardware utilisé sous Xilinx Virtex 2 xc2v3000-4bf95 pour le bloc du chiensearch.

FPGA Device	Virtex 2xc2v3000-4fg676
Vitesse	-4
Période minimum	3.406 ns
Fréquence maximale	239.600 MHz

Tableau 4.10 – Performance temporelle pour le bloc du chiensearch.

4.6.5 Implémentation de bloc du Forney

Structure de bloc du Forney

Le circuit de bloc du forney ainsi présenté dans le chapitre 3 est implémenté à l'aide de 9 cellules élémentaires (forney_cellule). Les différents signaux de commande sont générés à l'aide d'une machine d'état.

La structure est celle représentée sous la figure4.26.

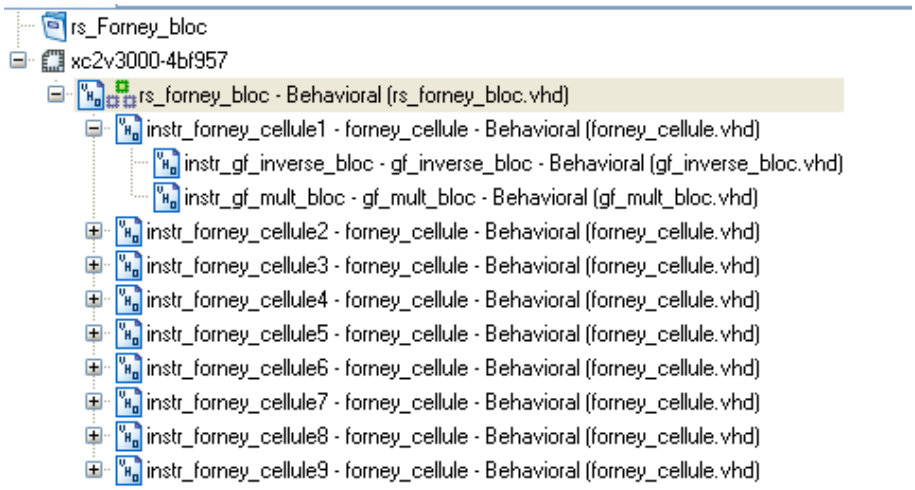

Figure 4.26 – Structure de bloc du Forney.

Le bloc qui calcule la division de polynôme d'amplitude des erreurs sur la dérivée de polynôme de localisation des erreurs est :

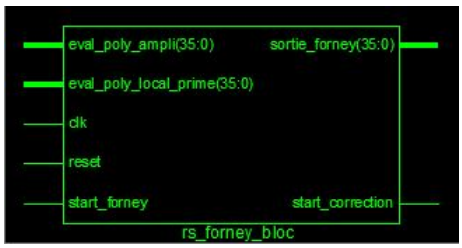

Figure 4.27 – Schéma de bloc du Forney.

Signaux de commande et d'entrée/sortie de bloc du Forney

Les signaux d'entrée/sortie sont :

- clk : signal d'horloge (std_logic).
- Start_forney : signal d'entrée qui permet d'activer le bloc du forney (std_logic), c'est une impulsion d'un cycle d'horloge qui précède les polynômes de localisation et d'amplitude des erreurs.
- reset : signal permettant la remise à zéro du syndrome (std_logic).
- eval_poly_ampl : signal d'entrée de l'évaluation de polynôme d'amplitude des erreurs. Cette sortie est sur 36 bits (std_logic_vector (35 downto 0)).
- eval_poly_local_prime : signal d'entrée de l'évaluation de la dérivé de polynôme de localisation des erreurs. Cette sortie est sur 36 bits (std_logic_vector (35 downto 0)).
- Sorite_forney : signal de sortiede bloc de forney. Cette sortie est sur 36 bits (std_logic_vector (35 downto 0)).
- Start_correction : signal de sortie qui permet l'activation du bloc de forney (std_logic).

La cellule élémentaire pour le calcul de bloc du forney est représentée sur la figure4.28.

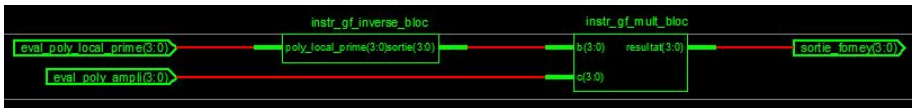

Figure 4.28 – Cellule élémentaire pour le calcul de bloc du forney.

Simulation de bloc du Forney

Des exemples de simulation de bloc de forney en utlisant le polynôme de localisation et d'amplitude des erreurs calculés par les algorithmes de berlekamp-massey et d'euclid sont représentés sur les figures 4.29 et 4.30 respectivement.

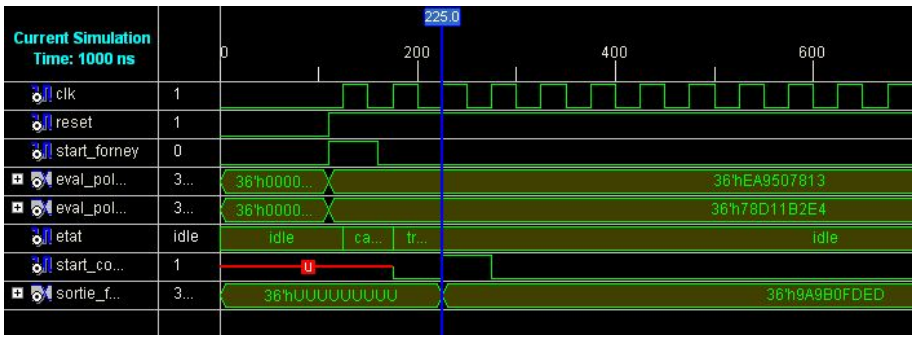

Figure 4.29 – Exemple de simulation de bloc du forney avec les polynômes de localisation et d'amplitude des erreurs calculés par l'algorithme de Berlekamp-Massey.

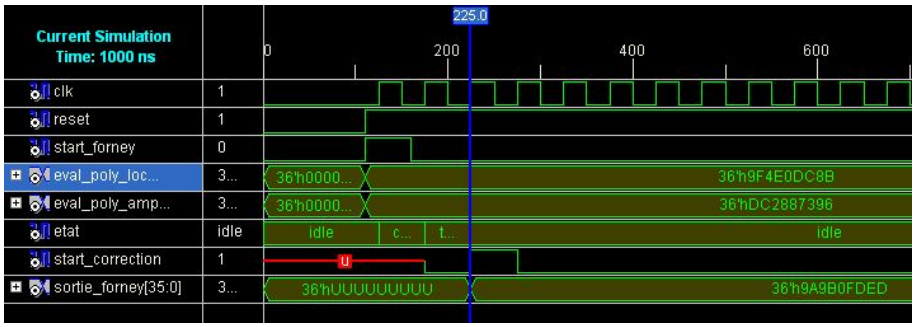

Figure 4.30 – Exemple de simulation de bloc du forney avec les polynômes de localisation et d'amplitude des erreurs calculés par l'algorithme d'Euclid.

Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour le bloc du forney RS(15,9) après placement et routage sous Xilinx Virtex 2xc2v3000-4fg676 sont représentées respectivement dans les tableaux 4.11 et 4.12 :

FPGA Device	**Virtex 2 xc2v3000-4fg676**	
Type de ressources	Utilisé	Taux d'utilisation
Slice Flip Flop	3 de 28672	1%
4 inputs LUTs	156 de 28672	1%
Bonded IOBs	112 de 484	23%
Total occuped slices	83 de 14336	1%

Tableau 4.11 – Ressource hardware utilisé sous Xilinx Virtex 2 xc2v3000-4fg676 pour le bloc du Forney.

FPGA Device	Virtex 2xc2v3000-4fg676
Vitesse	-4
Période minimum	5.230 ns
Fréquence maximale	191.186 MHz

Tableau 4.12 – Performance temporelle pour le bloc du forney.

4.6.6 Implémentation de bloc de correction

Structure de bloc de correction

Le bloc de correction est présenté dans le chapitre 3. Son circuit est réalisé sous forme structurel. Il est composé d'un seul fichier (rs_correction_bloc) décrivant les connexions entre les différents blocs déclarés comme composants. La sortie est commandée par le bloc « commande_sortie ». Ce bloc assure la sortie des données dans l'ordre de degré des données d'entrées.

La structure du bloc de correction est représenté dans la figure 4.31 :

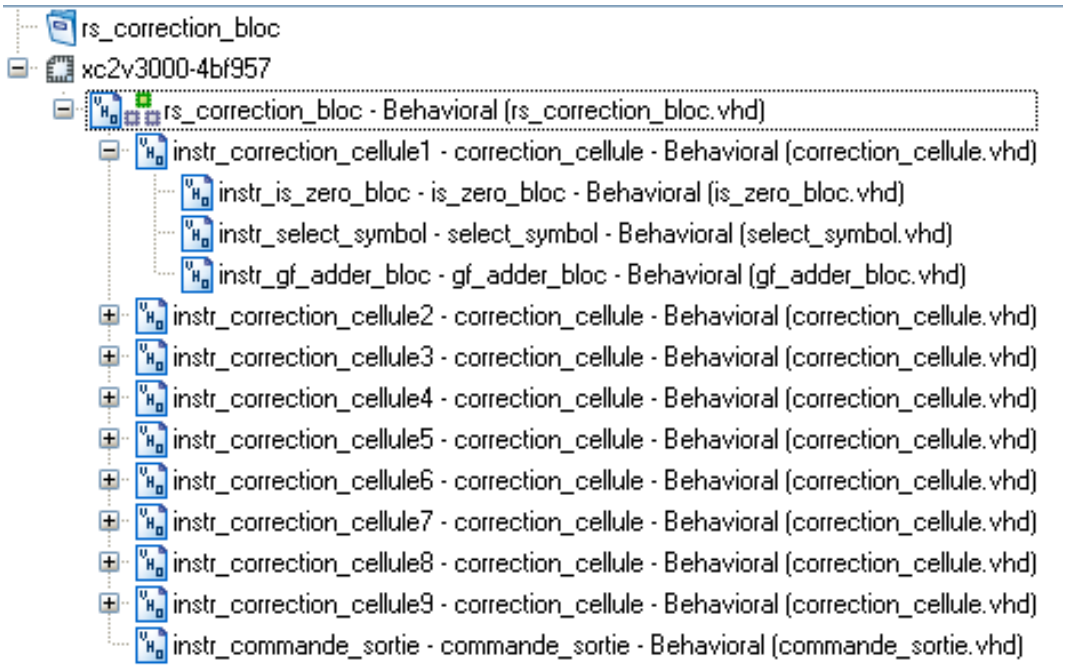

Figure 4.31 – Structure de bloc de correction.

Le bloc qui calcule la division de polynôme d'amplitude des erreurs sur la dérivée de polynôme de localisation des erreurs est :

Figure 4.32 – Schéma de bloc de correction.

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

Les signaux d'entrée/sortie sont :

- clk : signal d'horloge (std_logic).
- Start_correction : signal d'entrée qui permet d'activer le bloc du correction (std_logic).
- reset : signal permettant la remise à zéro de bloc de correction (std_logic).
- Sortie_forney : signal d'entrée de polynôme des erreurs. Cette entrée est sur 36 bits (std_logic_vector (35 downto 0)).
- data_in : signal d'entrée de bloc de correction qui contient les symboles de mot code avec des erreurs. Cette entrée est sur 36 bits (std_logic_vector(35 downto 0)).
- data_out : signal de sortie de bloc de correction qui contient les symboles de mot code corrigé. Cette sortie est sur 4 bits (std_logic_vector (3 downto 0)).

La cellule élémentaire pour le calcul de correction est représentée sur la figure 4.33.

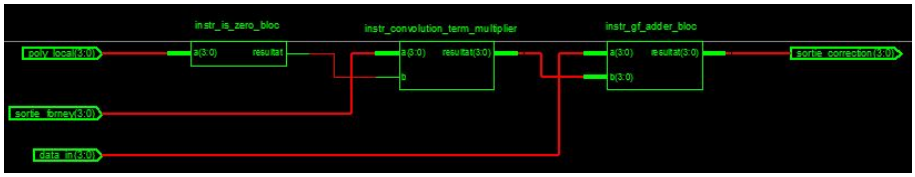

Figure 4.33 – Cellule élémentaire pour le calcul de correction.

Simulation de bloc de correction

La figure4.34 représente un exemple de simulation de bloc de correction.

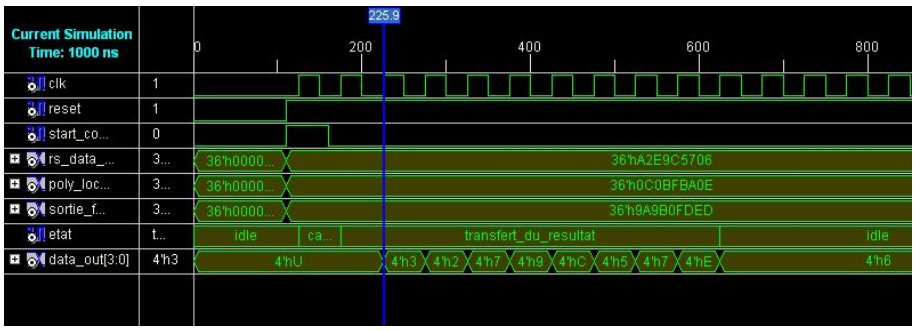

Figure 4.34 – Exemple de simulation de bloc de correction.

Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour le bloc du correction RS(15,9) après placement et routage sous Xilinx Virtex 2 xc2v3000-4fg676 sont représentées respectivement dans les tableaux 4.13 et 4.14 :

FPGA Device	**Virtex 2 xc2v3000-4fg676**	
Type de ressources	Utilisé	Taux d'utilisation
Slice Flip Flop	8 de 28672	1%
4 inputs LUTs	81 de 28672	1%
Bonded IOBs	115 de 484	23%
Total occuped slices	44 de 14336	1%

Tableau 4.13 – Ressource hardware utilisé sous Xilinx Virtex 2 xc2v3000-4fg676 pour le bloc de correction.

FPGA Device	**Virtex 2xc2v3000-4fg676**
Vitesse	-4
Période minimum	5.609 ns
Fréquence maximale	178.285 MHz

Tableau 4.14 – Performance temporelle pour le bloc de correction.

4.6.7 Implémentation de bloc du décodeur$(15, 9)$

Structure de bloc du décodeur$(15, 9)$

Le circuit de bloc de décodeur est réalisé sous forme structurel. Il est composé six principaux blocs : « rs_syndrome_bloc », « rs_euclid_bloc » ou « rs_Berlekamp_Massey_bloc » , « rs_chiensearch_bloc », « rs_forney_bloc », « rs_data_bloc » et « rs_correction_bloc ».

Ces derniers sont aussi implémentés avec des différents blocs déclarés comme composants décrits dans des fichiers indépendants.

La figure4.35 représente la structure de décodeur

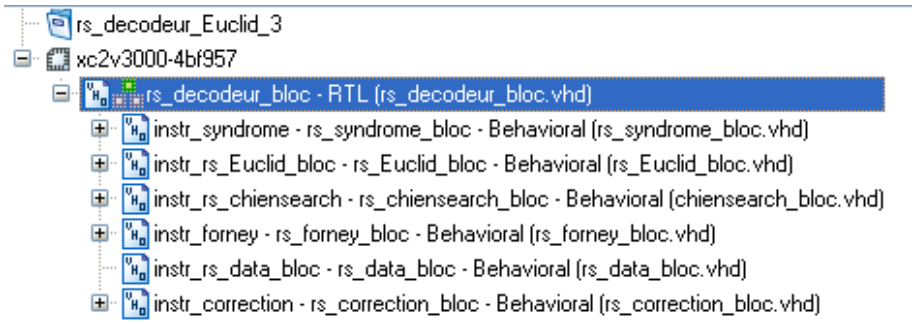

Figure 4.35 – structure de bloc du décodeur.

La figure4.36 illustre les différents blocs qui composent le bloc de décodeur

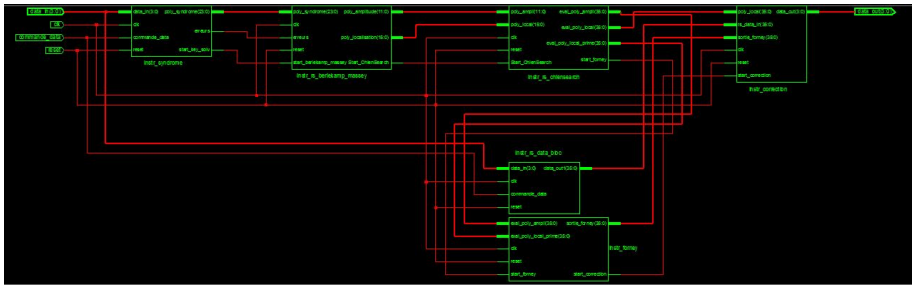

Figure 4.36 – Les blocs constituants de décodeur

Le bloc de décodeur RS(15,9) est représenté dans la figure4.37

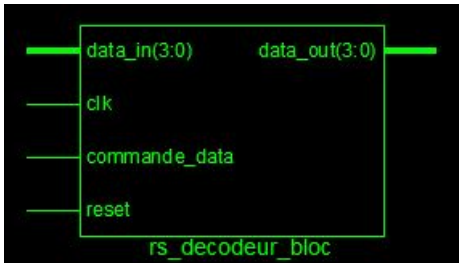

Figure 4.37 – Schéma de bloc du décodeur RS$(15, 9)$.

Signaux de commande et d'entrée/sortie de bloc du décodeur

Les signaux d'entrée/sortie sont :

- clk : signal d'horloge (std_logic).
- commande_data : signal d'entrée qui permet d'activer le décodeur (std_logic).
- reset : signal permettant la remise à zéro de décodeur (std_logic).
- data_in : signal d'entrée de mot code d'information. Cette entrée est sur 4 bits (std_logic_vector (3 downto 0)).
- data_out : signal de sortie de bloc de correction qui contient les symboles de mot code corrigé. Cette sortie est sur 4 bits (std_logic_vector (3 downto 0)).

Simulation de bloc du décodeur RS$(15, 9)$

Un exemple de simulation est celui représenté sur la figure4.38

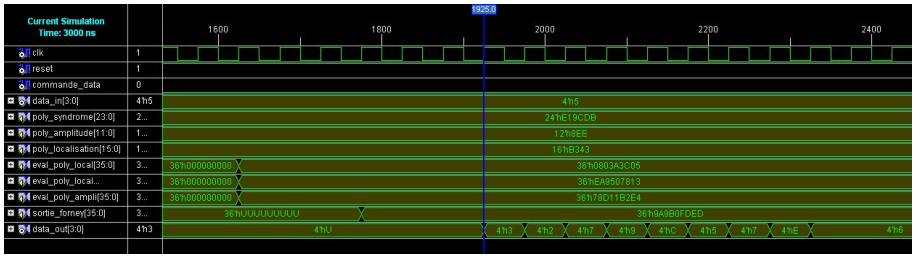

Figure 4.38 – Exemple de simulation de décodeur.

Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour le bloc de décodeur RS(15,9) utilisant le bloc de Berlekamp-Massey pour le calcul de $\alpha(x)$ et $\omega(x)$ après placement et routage sous Xilinx Virtex 2 xc2v3000-4fg676 sont représentées respectivement dans les tableaux 4.15 et 4.16 :

FPGA Device	**Virtex 2**		**xc2v3000-4fg676**	
Type d'algorithme	**EIBM**		**MEA**	
Type de ressources	**Utilisé**	**Taux d'utilisation**	**Utilisé**	**Taux d'utilisation**
Slice Flip Flop	743 de 28672	2%	772 de 28672	2%
4 inputs LUTs	797 de 28672	2%	853 de 28672	2%
Bonded IOBs	11 de 484	3%	11 de 484	3%
Total occuped slices	628 de 14336	4%	660 de 14336	4%

Tableau 4.15 – Ressource hardware utilisé sous Xilinx Virtex 2 xc2v3000-4fg676 pour le bloc de décodeur(15,9).

FPGA Device	Virtex 2xc2v3000-4fg676	
Type d'algorithme	EIBM	MEA
Vitesse	-4	-4
Période minimum	4.416 ns	8.035 ns
Fréquence maximale	118.821 MHz	124.465 MHz

Tableau 4.16 – Performance temporelle pour le bloc du décodeur(15,9).

4.7 Implémentation de bloc du décodeur$(255, 239)$

Le bloc de décodeur RS(255,239) est représenté dans la figure4.39 :

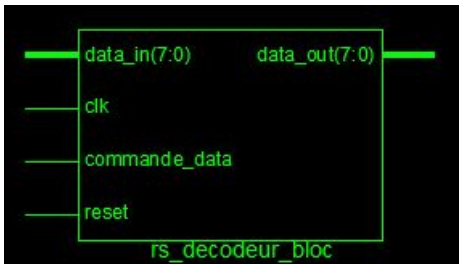

Figure 4.39 – structure de bloc du décodeur RS(255,239).

Signaux de commande et d'entrée/sortie de bloc du décodeur

Les signaux d'entrée/sortie sont :

- clk : signal d'horloge (std_logic).
- commande_data : signal d'entrée qui permet d'activer le décodeur (std_logic).
- reset : signal permettant la remise à zéro de décodeur (std_logic).
- data_in : signal d'entrée de mot code d'information. Cette entrée est sur 8 bits (std_logic_vector (7 downto 0)).
- data_out : signal de sortie de bloc de correction qui contient les symboles de mot code corrigé. Cette sortie est sur 8 bits (std_logic_vector (7 downto 0)).

Rapport de synthèse

Les ressources hardware utilisées et les performances temporelles pour le bloc de décodeur RS(255,239) utilisant le bloc de Berlekamp-Massey pour le calcul de $\sigma(x)$ et $\omega(x)$ après placement et routage sous Xilinx Virtex 2 xc2v3000-4fg676 sont représentées respectivement dans les tableaux 4.17 et 4.18 :

FPGA Device	Virtex 2		xc2v3000-4fg676	
Type d'algorithme	EIBM		MEA	
Type de ressources	Utilisé	Taux d'utilisation	Utilisé	Taux d'utilisation
Slice Flip Flop	7791 de 28672	27%	7403 de 28672	25%
4 inputs LUTs	7594 de 28672	26%	5695 de 28672	19%
Bonded IOBs	19 de 484	4%	19 de 483	4%
Total occuped slices	7292 de 14336	50%	6150 de 14336	42%

Tableau 4.17 – Ressource hardware utilisé sous Xilinx Virtex 2 xc2v3000-4fg676 pour le bloc de décodeur(255,239).

FPGA Device	Virtex 2xc2v3000-4fg676	
Type d'algorithme	EIBM	MEA
Vitesse	-4	-4
Période minimum	11.071 ns	8.930 ns
Fréquence maximale	90.325 MHz	111.982 MHz

Tableau 4.18 – Performance temporelle pour le bloc du décodeur(255,239).

4.8 Mise en œuvre sur la carte memec XC2V1000-4FG456C

Les tests sur la carte a permet de vérifier le bon fonctionnement des codeurs et décodeur RS(15,9) et (255,239). La carte possède deux horloges : une de fréquence de 100 MHz et l'autre de fréquence 24 MHz. Nous avons utilisé cette dernière comme signal d'horloge.

L'entité génératrice des stimuli génère les différents signaux permettant de tester les codeurs et décodeurs RS implémentés. Un diviseur d'horloge est ainsi implémenté, et la fréquence de vérification est alors 1,34 Hz. Les résultats de codage et décodage sont affichés sur l'afficheur sept-segment de la carte.

Les structures de test de codeur et décodeur sont représentées dans les figures 4.40,4.41, 4.42.

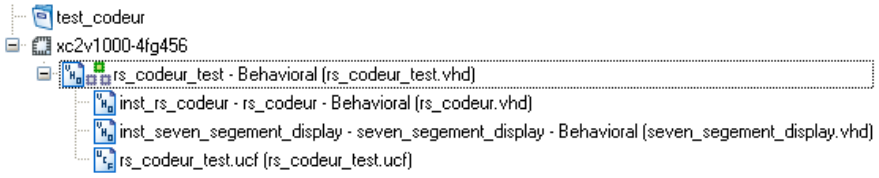

Figure 4.40 – Structure de test de bloc de codeur

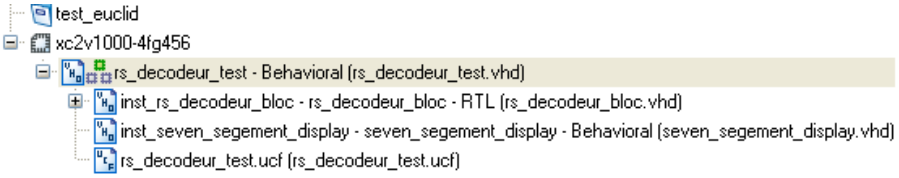

Figure 4.41 – Structure de test de bloc de décodeur utilisant l'algorithme d'Euclid.

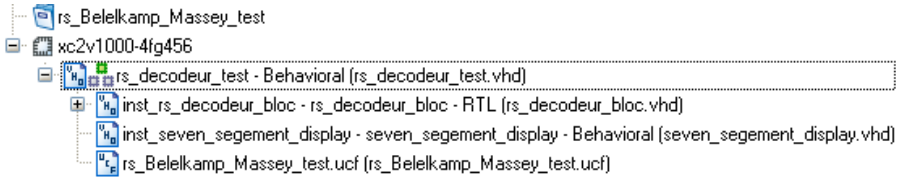

Figure 4.42 – Structure de test de bloc de décodeur utilisant l'algorithme de Berlekamp-Massey.

4.9 Conclusion

Dans ce chapitre, nous avons implémenté tout d'abord,les opérations de bases dans les corps de Galois $GF(2^4)$ et $GF(2^8)$. Ensuite, nous avons présenté les résultats de l'implémentation de codeur $RS(15, 9)$ et le codeur $RS(255, 239)$, ainsi que celle des différents blocs utilisés dans le bloc de décodeur $RS(15, 9)$. A la fin de ce chapitre, nous avons développé implementé de décodeur $RS(15, 9)$ et le décodeur $RS(255, 239)$.

Les résultats de l'implémentation montre que les ressources hardware requises augmentent avec l'augmentation de la taille du code RS. Le bloc résolvant l'équation clé (rs_EIBM_bloc ou rs_Euclid_bloc) utilise plus de ressources par rapport au autres blocs.

Dans le cas du RS(15, 9), les opérations nécessitent un temps de latence de 36 cycles d'horloge tandis que dans RS$(255, 239)$, elles nécessitent 291 cycles d'horloge. Lors de la simulation de chaque bloc seul, le temps de latence est moins de quatre cycles d'horloge. Ces derniers sont utilisés lors de transition dans les bascules dans le passage d'un bloc à l'autre.

Conclusion et Perspectives

L'objectif de ce mémoire est l'implémentation de codeur et décodeur de Reed-Solomon pour les communications sans fils « WiMax » sur un circuit reconfigurable « FPGA ». Nous avons décrit le principe de fonctionnement de codeur et décodeur RS, ainsi que les architectures adoptées.

Après une étude théorique, dans laquelle nous avons rappelé les principes de base de la communication numérique, de la théorie de l'information, de la théorie des codes correcteurs d'erreurs et plus particulièrement, des codes en blocs linéaires, nous avons étudié la théorie concernant le codage et de décodage de Reed-Solomon. Cette dernière présente la partie la plus complexe tant au niveau théorique qu'au niveau hardware.

La partie principale de décodage et le bloc qui résout l'équation clé permettant de calculer les polynômes de localisation des erreurs et d'amplitude des erreurs. Plusieurs algorithmes existent et les plus répondus sont l'algorithme de Berlekamp-Massey et l'algorithme d'Euclid.

Nous avons développé les codes en VHDL pour le codeur et 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.Les différents blocs constituant le codec RS sont simulés et synthétisés.

Les résultats de simulation et de la synthèse montrent que l'algorithme de Berlekamp-Massey, malgré qu'il présente une structure très complexe, présente les meilleures performances au niveau de ressources hardware que le décodeur utilisant le bloc d'Euclid. En revanche, ce dernier présente les meilleures performances au niveau de performance temporelle.

L'implémentation des deux codes de Reed-Solomon ont été mené jusqu'au bout sur la carte XC2V1000-4FG456C de la famille Vertex 2.

A l'avenir il serait pertinent d'implémenter le codeur et décodeur de Reed-Solomon pour la correction des erreurs et des effacements utilisé soit dans les communications sans fils ou bien dans d'autre domaine de communication.

Il serait aussi pertinent d'explorer d'autres architectures de codec RS visant à l'augmentation de la rapidité de ces codes dans le domaine des communications sans fils en utilisant les architectures en pipeline.

Bibliographie

[1] H. Nyquist, "Certain factors affecting telegraph speed", Bell System Technical Journal, vol. 3, pp. 324-346, 1924.

[2] R. Hartley, "Transmission of information", Bell System Technical Journal, pp. 535-563, July 1928.

[3] C. E. Shannon, "A mathematical theory of communication", Bell System Technical Journal, vol. 27, pp. 379-423, 623-656, Juillet, Octobre 1948.

[4] M. Temer Elias, "Etude des codes LDPC et application dans un système MIMO", Septembre 2009.

[5] D. Le Ruyet, "Théorie de l'information, Codage source-canal", Ecole Supérieure de Conception et de Production industrielles, France,p. 2, Jan. 2007.

[6] P. Godlewski, "Complexité de systèmes de correction d'erreurs", Traitement du Signal , vol. 1-n 2-2, pp. 180-183, 1984.

[7] M. Cote, "Reconnaissance de codes correcteurs d'erreurs", These de Doctorat, Ecole Polytechnique, France, p. 12, Mars 2010.

[8] Nicolas Sendrier, "Codes correcteurs d'erreurs à haut pouvoir de correction", These de Doctorat, Université Paris 6, 1991.

[9] 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.

[10] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," SI AM Journal of Applied Mathematics, Volume 8, pp. 300-304, Juin 1960.

[11] G. D. Forney, Concatenated Codes, These pour obtenir le degree de Docteur en Science, MIT, USA, Décembre 1965.

[12] Samuele Dietler, "Implémentation de codes de Reed-Solomon sur FPGA pour communications spatiales", Projet de diplôme, Haute École d'Ingénierie et de Gestion du Canton de Vaud, 2005.

[13] 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.

[14] 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.

[15] 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.

[16] D. V. Sarwate, Z. Yan, "Modified Euclidean Algorithms for Decoding Reed-Solomon Codes", IEEE, 2009.

[17] J. L. Massey, "Shift-Register Synthesis and BCH Decoding", IEEE Transactions on Information Theory, pp. 122-127, 1969.

[18] Mazen Ezzeddine, "Modélisation et réalisation de la couche physique du système de communication numérique sans fil, WiMax, sur du matériel reconfigurable", mémoire pour l'obtention du grade de Maître ès sciences (M.Sc.) en infonnatique, Université de Montréal, 2009.