


République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme

Master en Informatique

Option : Système d'Information et connaissance.

Thème

Application des codes correcteurs d'erreurs Reed Muller

Présenté par :

M^lle^. Khadidja SERIR

Encadré par :

M^r^.MourtadaBENAZZOUZ

Soutenu en octobre 2011 devant la commission SIC

Année Universitaire 2010 - 2011

REMERCIEMENTS

Nous remercions vivement notre encadreur, Mr Mourtada BENAZZOUZ, pour nous avoir proposé ce sujet d'actualité assez passionnant ; sa patience et son organisation nous ont permis de surmonter de nombreuses difficultés liées à ce travail.

Nous tenons à lui exprimer nos sincères déférences pour son encadrement.

Merci également à tous ceux qui ont contribué à notre formation spirituelle.

Notre gratitude s'adresse également à tous ceux qui, de loin ou de prés, ont participé à la réalisation de ce travail.

Nous tenons aussi à remercier toute l'équipe pédagogique pour nous avoir transmis leur savoir tout au long de notre cycle d'étude.

Merci également à tous mes collègues de travail dans la direction de wilaya de la poste et des technologies de l'information et de communication qui m'ont soutenu avec leurs encouragements.

DEDICACES

Je dédie ce mémoire :

A mes chers parents qui m'ont soutenu avec leurs encouragement.

A mes chères sœurs Nawel, Souad.

A mes chers frères Sidi mohamed, Sid ahmed.

A mes beaux frères Abdelaziz, benamar.

A mes neveux Oussama, Noureddine.

A mes nièces Sanaa, Wafaa.

A tous ceux que je porte dans mon cœur.

Serir khadidja

Table des Matières

Chapitre 3 : Codes Reed-Muller

Chapitre 4 : Implémentation logicielle

Introduction Générale

De nos jours, nous vivons dans un monde où les communications jouent un rôle primordial tant par la place qu'elles occupent dans le quotidien de chacun, que par les enjeux économiques et technologiques dont elles font l'objet. Nous avons sans cesse besoin d'augmenter les débits de transmission tout en gardant ou en améliorant la qualité de ceux-ci. Mais sans un souci de fiabilité, tous les efforts d'amélioration seraient ridicules car cela impliquerait forcément à ce que certaines données soient retransmises. C'est dans la course au débit et à la fiabilité que les codes correcteurs entrent en jeux.

La communication avec les sondes spatiales, à l'autre bout du système solaire, pose le problème de la fiabilité du message transmis. Une transmission sur une telle distance est obligatoirement parasitée (notamment à cause de diverses sources de perturbations électromagnétiques). Pourtant, dans ce domaine et dans bien d'autres, il est primordial que les informations collectées par les sondes soient bien reçues. Il y a donc nécessité de « sécuriser » la transmission : c'est le rôle des codes correcteurs d'erreurs. On rajoute au message à transmettre des informations supplémentaires, qui permettent de reconstituer le message au niveau du récepteur.

Un code correcteur d'erreur permet de corriger une ou plusieurs erreurs dans un mot de code en ajoutant aux informations des symboles redondants, autrement dits, des symboles de contrôle. Différents codes possibles existent mais dans ce document nous traiterons seulement les codes Reed – Muller, ainsi de faire une petite comparaison entre ces types des codes et les codes Reed Solomon qu'on a déjà les implémenter avant dans un autre projet.

L’étude de notre présent mémoire est organisée en 4 chapitres :

Le premier chapitre se consacre aux codes correcteurs d’erreurs en générale et la théorie d’information qui présente le point de départ de ces codes ; ensuite le deuxième qui décrit les codes en blocs en général et des exemples de ces codes qui existent avec leurs principes de codage et décodage, en troisième chapitre, une présentation bien détaillée des codes Reed-Muller, en décrivant comment se déroule le codage et le décodage dans ce type des codes correcteurs, mais avant expliquer ces types de codes, nous avons donné une brève introduction pour les mathématiques utilisés et les corps finis, et enfin, le chapitre quatre, qui traite nos expérimentations menées afin d’évaluer les performances de ce type de codes avec une synthèse comparative entre les codes Reed solomon et Reed Muller. Et nous terminons par une conclusion générale.

Chapitre 1
Codes Correcteurs d'Erreurs

I. Problématique des codes correcteurs d'erreurs

Utilisation : les codes correcteurs d'erreurs sont utilisés pour la transmission (par satellite, téléphonie, disque laser, TV haute définition), qui luttent contre le bruit, traitement d'image et de la parole, cryptographie (signature, carte à puce), compression de données, etc.…

Origine des codes correcteurs d'erreurs : C'est la théorie de l'information initiée par C. Shannon dans les années 50.

Problème : La problématique des codes correcteurs d'erreur est la suivante : un expéditeur *A* envoie un message *m* à *B*, durant la transmission de ce message des erreurs se produisent éventuellement, et *B* reçoit un message *m*' qui comporte peut-être des erreurs. Il s'agit de trouver comment faire pour que *B*,

1- d'une part détecte l'existence d'erreurs,
2- d'autre part, si elles ne sont pas trop nombreuses, sache les corriger.

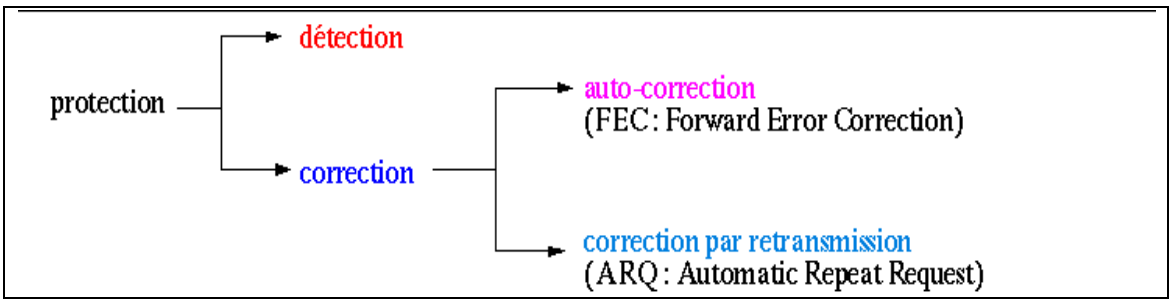


Figure1.1 : Stratégies de protection contre les erreurs de transmission. [33]

Dans certains cas, lorsqu'il est rapide de réexpédier le message, la détection des erreurs suffit. Néanmoins, dans d'autres cas, la correction des erreurs s'avère indispensable, par exemple, si *A* est une fusée intersidérale qui envoie une photographie d'une météorite, et s'il faut par exemple une semaine pour que cette photographie atteigne *B*, à savoir la terre, on ne pourra pas demander à la fusée de reprendre la même photographie de la

météorite qui sera bien loin de la fusée quinze jours après.Le même cas de figure se produit pour un disque compact commercial, s'il possède quelques défauts, il est plus rentable d'avoir un procédé pour corriger les erreurs que de devoir le racheter.

On se propose de détecter les erreurs et de les corriger de façon automatique (si le nombre d'erreurs n'est pas trop grand).

Principe : Pour se faire nous procédons de la façon suivante :

- nous choisit un alphabet fini F de cardinal q et on représente l'information comme une suite d'éléments de F.
- On découpe la suite obtenue en blocs de longueur fixe k.
- Un message m sera un bloc de longueur k, appartient à l'ensemble F^k.

$$m := (m_{1,}\ldots\ldots,m_k)$$

L'ensemble F^k est l'espace des messages. Le nombre maximum de messages différents possibles est égal à q^k. On peut aussi supposer que l'espace des messages est un sous ensemble M de F^k, dans ce cas le nombre maximum de messages différents possibles est égal à M.

- Nécessité d'un codage. Si on transmet le message m par le canal et si ce message arrive à destination avec des erreurs, le récepteur ne pourra pas retrouver le message émis.
- Phase d'encodage, on se donne un entier $n > k$ et C inclus dans F^n (C est le code). Un encodage associé au code C est une application injective E.

$$\begin{array}{c} \overset{E}{} \\ F^k \rightarrow F^n \\ m = (m_1,\ldots\ldots,m_k) \rightarrow c = (c_1,\ldots\ldots,c_n). \end{array}$$

Telle que $E(F^k) = C$. L'application E est donc une bijection de F^k sur C.

Remarque : nous transmettons c (et non m) à travers le canal.

Par l'encodage, nous avons augmenté la longueur des messages émis, on dit que nous avons introduit de la redondance (= n-k) mais, comme ainsi les messages sont plus différents les uns des autres, on pourra à la réception corriger les erreurs éventuelles (dans une certaine mesure).

- Phase de décodage, si le récepteur reçoit y qui est dans F^n, il souhaite retrouver le message m envoyé.

II. Théorie de l'information

On définit un canal de transmission comme un système physique permettant la transmission d'une information entre deux points distants. Le taux d'erreurs binaire (TEB) d'un message est le rapport du nombre de bits erronés par le nombre de bits du message.

En 1948, Shannon énonce dans « A Mathematical Theory of Information » le théorème fondamental de la théorie de l'information :

Tout canal de transmission admet un paramètre C, appelé capacité du canal, tel que pour tout $\varepsilon > 0$ *et pour tout* $R < C$*, il existe un code de taux R permettant la transmission du message avec un taux d'erreurs binaire de* ε*.* **[27]**

En d'autres termes, nous pouvons obtenir des transmissions aussi fiables que l'on veut, en utilisant des codes de taux plus petits que la capacité du canal.

Cependant, ce théorème n'indique pas le moyen de construire de tels codes, nous cherchons donc à construire des codes ayant un taux le plus élevé possible (pour des raisons de temps et de coût) et permettant une fiabilité arbitrairement grande.
Les codes classiques ne permettent pas d'atteindre cette limite. Nous avons donc développé d'autres systèmes de codages.

III. Classification des codes Correcteurs

1. Présentation

Pour un traitement informatique, c'est-à-dire automatisé, de l'information, nous numérisons le signal à transmettre (une image, un son…). Nous ramenons ainsi celui-ci à une séquence de bits e_1e_2…. A cause des inévitables parasites qui détériorent le message, nous ne pouvons pas envoyer cette séquence telle quelle.

Pour améliorer la fiabilité de la transmission des données, une des méthodes de codage les plus simples est alors de répéter chaque bit.

La séquence e_1e_2… sera ainsi transmise sous la forme $e_1e_1e_2e_2$…. Lors de la réception du message, le décodeur peut ainsi comparer chaque couple de bits reçus,s'ils sont différents alors il y a détection d'erreur.

Nous voyions ainsi qu'en doublant la longueur du message (mais aussi le temps de transmission), nous parvenons à détecter d'éventuelles erreurs.

Toutefois, ce codage simple ne permet pas de les corriger. Pour cela, nous pouvons tripler les bits. Si nous considérons (ce qui est plus que raisonnable) qu'il y a au maximum une erreur pour chaque séquence de 3 bits, alors il est possible de les corriger : le décodeur n'a qu'à choisir le symbole qui apparaît deux fois dans chaque triplet reçu.

Si le canal de transmission n'est pas trop parasité, il paraît inutile d'ajouter autant de redondance au message transmis. Nous pouvons ainsi utiliser le système du bit de parité (qui ne permet que la détection d'erreurs) : le message est découpé en blocs de k bits, auxquels nous ajoutons un bit tel qu'il y ait un nombre pair de 1 dans le bloc transmis.

Pour approcher au mieux la capacité C du canal si, conformément au théorème de Shannon, l'entropie de la source qui est inférieur à C nous pouvons ajouter de la redondance dans le codage de la source afin de diminuer les erreurs de transmission. C'est le problème du codage de canal. A côté des premiers codes empiriques (bit de

parité, répétition,…) deux grandes catégories de codes ont été développées et sont actuellement utilisées en faisant l'objet permanent de perfectionnements :

- Les codes en blocs.
- Les codes en treillis.

La figure ci-dessous donne un simple résumé de la grande famille de codage. Dans la première classe (à droite sur la figure), citons les codes les plus célèbres comme les codes BCH, Reed Muller, Reed Solomon et Goppa, Golay et Hamming. La deuxième classe (à gauche sur la figure) est moins riche en variété mais présente beaucoup plus de souplesse surtout dans le choix des paramètres et des algorithmes de décodage disponibles. Citons par exemple, les codes convolutifs binaires systématiques récursifs très utilisés dans les modulations codées (TCM) et les codes concaténés parallèles (Turbo Codes).

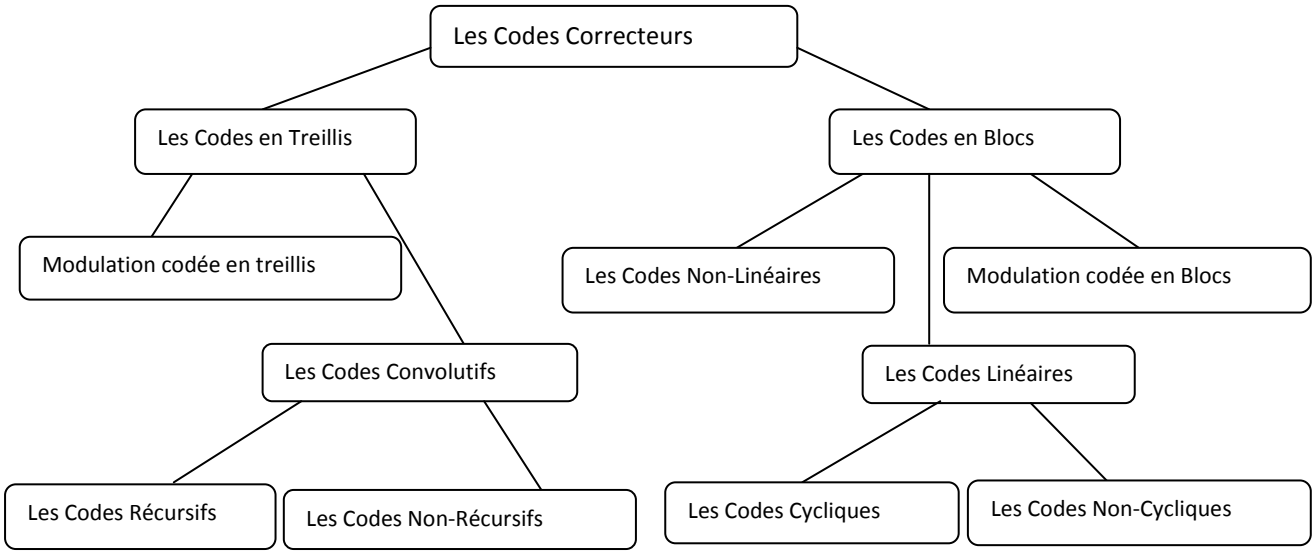


Figure 1.2 : la hiérarchie des codes correcteurs.[34]

IV. Les codes détecteurs et correcteurs d'erreurs

Les concepteurs de réseaux ont développé deux stratégies dans le domaine des erreurs de transmission. La première consiste à inclure dans les blocs de données suffisamment de redondance pour que le récepteur soit capable de restituer les données originales à partir des données reçues.

La deuxième consiste à ajouter juste assez de redondance dans les données pour que le récepteur puisse juste détecter les erreurs et demande alors la retransmission des trames erronées.

Dans l'étude de notre projet en se base surtout sur les codes correcteurs qui ont été introduits pour corriger les erreurs de transmission ou de lecture de données numériques, ou les erreurs survenant au cours de leur inscription sur un support physique (bande, CD) ou encore lorsque les données subissent une altération sur le support de stockage.
Voici quelques domaines où ils sont appliqués :

- transmissions spatiales,
- disque compact et DVD,
- communications par internet,

Un code correcteur est utilisé afin de diminuer le nombre des erreurs qui se produisent au cours des transmissions téléinformatiques.

L'itinéraire classique :
Emission → transmission → réception.
S'enrichit :

- d'une fonction de codage en amont de l'émission,
- d'un algorithme de décodage en aval de la réception (ces deux étapes sont évidemment suivies),
- de la restitution de l'information (la meilleure possible) au destinataire.

La logique de la méthode générale peut se résumer par le schéma qui suit,

- Au départ le mot d'information i est codé c qui est alors émit ; à la réception c'est un message m qui apparaît ;
- C étant le code utilisé, c_1, c_2 sont des mots de code et i_1, i_2 les mots d'information correspondants ;
- Le destinataire peut recevoir un autre mot que celui qui lui était destiné.

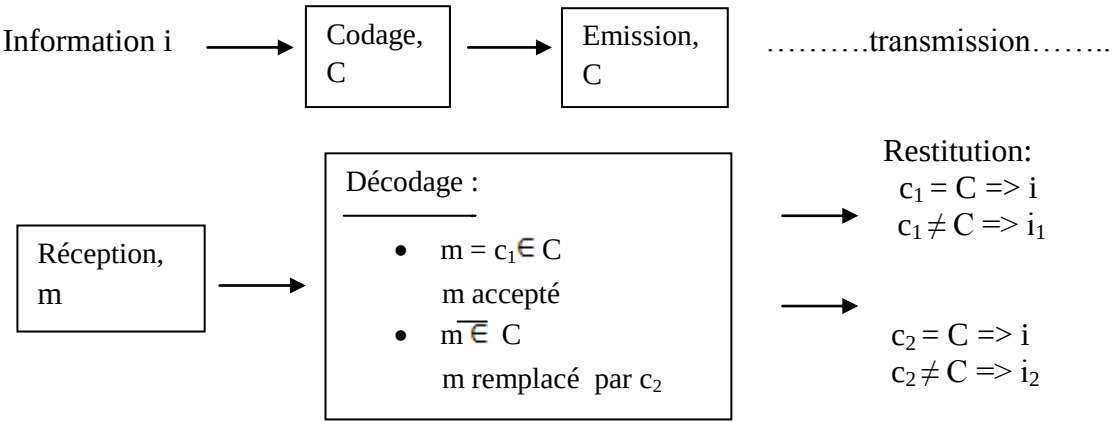


Figure 1.3 : la transmission avec code correcteur d'erreur.[3]

La recherche des codes réalisant les deux objectifs qui ont été fixés :

- ❖ détection et correction d'erreur, prend également en compte la simplicité du codage, du décodage et de la restitution de l'information.
- ❖ la valeur du rendement des codes.

L'utilisation des modèles et structures mathématiques, l'ingéniosité des chercheurs, ont permis d'élaborer des codes de plus en plus performants. C'est ce que nous allons découvrir au cours des chapitres suivants.

1. Quelques Codes en blocs

Un code par blocs codant tous les mots de longueur k par certains mots de longueur n, supérieur à k, sera noté $C_{n,k}$. Le nombre n est appelé longueur du code.
Toute suite binaire de longueur n peut apparaître à la réception, nous lui réserveront dorénavant le terme de message, qu'elle représente ou non un mot de code.

Bien qu'absolument nécessaire, la redondance alourdit la transmission en temps et charge la mémoire du système, il est donc important d'évaluer le rendement nommé encore taux de transmission du code donné par le rapport :

$$R = k / n \quad \text{.....1.1}$$

Et puisqu' il faut utiliser n bits pour k bits d'information à communiquer, le coût du codage est donc proportionnel à **1 / R**.

Il est évident que tout codage doit être « injectif », c'est-à-dire que chaque mot de code ne peut coder qu'un seul mot d'information.

On note que pour un code en blocs $C_{n,k}$ il y'a :

2^k : Mots d'information de k bits à coder
2^n : Messages reçus possibles, de longueur n, dont :
2^k : Sont des mots du code.

Après l'encodage, le décodage est nécessaire au niveau de récepteur, cette opération consiste à calculer les distances entre c' (le mot de code reçu) et tous les mots du code. Le rapport **d / n** renseigne donc sur la fiabilité du code.[3],[30]

a. Codes systématiques

Il est pratique de construire un mot de code de longueur n en ajoutant à la suite des k bits $i_1 \ldots i_k$ d'information, r = (n-k) bits, les r bits appelées les bits de contrôle ou de redondance, formant la clé de contrôle. On obtient le mot de code :

$$c_1 \ldots c_k \, c_{k+1} \ldots c_n = i_1 \ldots i_k \, r_1 \ldots r_r \quad \ldots\ldots 1.2$$

Ce codage est dit systématique ainsi que tout code que l'on peut obtenir par un tel codage et qui est donc un code par bloc particulier.

Le contrôle est alors simple, toute suite de longueur k étant un mot d'information, le récepteur calcule la clé correspondant au mot formé par les k premiers bits du message. si cette clé est différente de celle qui se trouve en fin du mot reçu, il y a anomalie, signe d'erreur au cours de la transmission.[26]

b. Code de parité

Dans ces types des codes, intéressons nous à la transmission des caractères de texte ; un caractère (une lettre ou un signe) est conventionnellement une suite de 7 chiffres binaire. Un texte exprimé sous forme binaire est découpé en mots d'information qui sont ici des caractères.

Le codage appelé par bits de parité consiste à ajouter à la suite de chaque mot d'information, un nouveau bit, de telle sorte que le nombre total de chiffres 1 soit alors de parité fixé (paire ou impaire).

Il s'agit d'un code systématique dont la clé de contrôle ne possède qu'un seul bit.

- **Probabilité d'erreurs détectées**

Les messages erronés n'étant pas détectés dans tous les cas, il est nécessairement de calculer une probabilité de les reconnaître.

Rappelons qu'un message est erroné s'il n'est pas identique au mot de code émis, (même si la partie qui comporte l'information est erronée).
Supposons que :

- La probabilité d'erreur[3] soit la même, p, pour chaque bit (la valeur de p est évidemment très petite) : la probabilité qu'un bit soit correctement transmis est donc $q = 1- P$.
- Les erreurs sur les bits soient indépendante les unes des autres (ce n'est pas toujours le cas) : le modèle de la situation est alors « un schéma de Bernoulli ».

Soit X le nombre de bits erronés dans un message, X vari de 0 à n = 8 en suivant la loi binomiale de paramètres n et p.
Une configuration où k bits sur n sont inexacts a une probabilité $P^{k}(1-P)^{n-k}$.
Si C_n^k désigne le nombre de ces configurations, la probabilité $P(k)$ que l'erreur d'un message porte sur k positions est donc :

$$P(k) = Pr(X = k) = C_n^k\, p^k\, (1-P)^{n-k} = C_8^k p^k (1-P)^{8-k} \quad \text{.....1.3}$$

D'où la probabilité $P(0)$ que la transmission soit parfaite :

$$P(0) = Pr(X = 0) = q^8 \quad \text{.....1.4}$$

Et la probabilité d'erreur d'un message : $P_{err} = 1 - q^8$……..….1.5

c. Code linéaire

Un code linéaire sert à faire correspondre à chaque mot d'information un mot de code, par une fonction linéaire, facilite la construction du code aussi bien que le contrôle des messages reçus. Les codes linéaires sont des codes par blocs construits à l'aide d'une telle fonction.

On note F_2 le corps à deux éléments 0 et 1. Les mots de longueurs n sont les éléments de F_2^n, que l'on écrira comme des vecteurs lignes. Un code linéaire de longueur n est un sous espace vectoriel C inclus dans F_2^n.

La lettre k désignera toujours la dimension de C (comme espace vectoriel). Le nombre de mots du code C est 2^k. Le poids d'un mot $x = (x_1 \dots x_n) \in F_2^n$ noté w(x), est le nombre d'indices i tels que $x_i \neq 0$. Comme d(x, y) = w(x - y), la distance minimal d d'un code linéaire C est le minimum des poids w(x) pour $x \in C$ non nul. (On suppose que C n'est pas le code nul.) On regroupe les trois paramètres n, k et d d'un code linéaire C en disant que C est de type (n, k, d), le codage des codes linéaires repose sur

- **Matrice génératrice**

On peut se donner un sous-espace vectoriel (et donc un code) par une base. Soit C un code linéaire. Une matrice génératrice de C est une matrice dont les lignes forment une base de C. Une matrice génératrice G est donc de taille k*n et de rang k. Si m est un vecteur ligne de F_2^k, le produit m.G est un mot du code C (que l'on peut voir comme une opération de codage).

Si la matrice G est de la forme (I; P), on dit que le codage est systématique. Les k premiers bits d'un mot de code portent l'information (on y recopie le vecteur de F_2^k), les n-k suivants sont de la redondance.

On dit que deux codes linéaires de même longueur sont équivalents si l'un s'obtient à partir de l'autre par une permutation des coordonnées. On peut vérifier que deux codes équivalents ont même type. De plus tout code est équivalent à un code donné par un codage systématique.

2. Code de Hamming

Un code de Hamming est un code correcteur linéaire. Il permet la détection et la correction automatique d'une erreur si elle ne porte que sur une lettre du message.

Un code de Hamming est parfait, ce qui signifie que pour une longueur de code donnée, il n'existe pas d'autre code plus compact ayant la même capacité de correction. En ce sens, son rendement est maximal.

Le code de Hamming est de paramètres ($2^m - 1$, $2^m - m - 1$,3), il permet de détecter 3 erreurs et de corriger une seule.

Plusieurs méthodes permettent de construire un code de Hamming. Une approche consiste à rechercher les codes cycliques de distance minimale égale à trois, le code apparaît alors comme un cas particulier de code BCH.

❖ Règle de Hamming

Pour détecter des erreurs, il faut que le code ait une distance de Hamming de **(d+1).** Par contre si on veut être en mesure de corriger des erreurs, il faut une distance de Hamming de **(2d+1)**.

❖ Distance de Hamming

Les messages transmis sont supposés découpés en blocs (ou mots) de longueur n écrits avec l'alphabet {0,1}. Un code (binaire) est un sous-ensemble C de l'ensemble $\{0,1\}^n$ de tous les mots possibles. On dit que n est la longueur de C.
La distance de Hamming entre deux mots $x = (x_1 \dots x_n)$ et $y = (y_1 \dots y_n)$, que l'on notera d(x,y) est le nombre d'indices i tels que $x_i \neq y_i$. C'est bien une distance sur $\{0,1\}^n$.La distance minimale du code C est le minimum des d(x,y) pour x et y des mots différents de C (on suppose que C a au moins 2 mots !). On la notera toujours d.

Exemple : Considérons $C = \{c_0, c_1, c_2, c_3\}$ avec :
$c_0 = (00000)$; $c_1 = (10110)$; $c_2 = (01011)$; $c_3 = (11101)$
C'est un code de longueur 5 et de distance d = 3.

3. Codes polynomiaux

Dans ces codes, on suppose que les bits d'une chaîne de caractères sont les coefficients d'un polynôme. Ces coefficients ne prennent que deux valeurs : 0 ou 1. Un bloc de k bits est vu comme la série des coefficients d'un polynôme comprenant les termes allant de x^{k-1} à x^0. **[22]**

Les codes polynomiaux forment aussi une catégorie des codes linéaires dont tous les mots peuvent se déduire d'un seul d'entre eux. En plus d'une grande facilité de codage et de contrôle, ils permettent une bonne détection des messages erronés. Ils sont liés à une représentation des n-uples binaire par des polynômes, ce qui leur a valu leur dénomination ou nous allons le détailler dans le chapitre suivant.

4. Code Cyclique

Les codes cycliques ont la propriété d'être stable par permutation circulaire des mots.

T : $F_2^n \rightarrow F_2^n$

$(x_1,x_2,.....,x_n) \rightarrow (x_2,.....,x_n,x_1)$

On identifie F^n à l'algèbre $F_2[X]/(X^n-1)$ par $(x_1,x_2,.....,x_n) \rightarrow x_1X^{n-1}+.....+x_{n-1}X+x_n$.

Ce sont des codes polynomiaux dont le polynôme générateur $g(x)$ divise (x^n+1)où n est la longueur du code.

Cette particularité permet la construction immédiate d'une matrice de contrôle caractéristique de ce type de code et une simplification de la méthode de correction automatique. Pour la détermination des codes cycliques de longueur n, la connaissance diviseurs (x^n+1) est essentielle. **[22]**

a. Codes BCH

Les codes BCH(Bose-Chaudhuri-Hocqenghem) sont une extension des codes cycliques, ils sont construits sur un alphabet composé d'un grand ensemble de symboles basés sur les propriétés des corps finis.

Ils sont considérés comme ceux qui ont la plus grande capacité de correction d'erreurs indépendantes, pour une redondance et une longueur donnée. **[21]**

b. Codes Reed -Solomon

Les codes de Reed-Solomon RS(n,k) sont formés de n symboles, avec n = q - 1 au maximum et q = 2^s, chaque symbole appartenant à GF(q) qui est le corps de Galois (Galois Field) à q éléments, s représente donc le nombre de bits par symbole. Le nombre t est égal à (n-k)/2 représente le nombre de symboles d'erreurs que ce code sera capable de corriger.[33]

c. Codes CRC (Cyclic Redundancy Check)

Une séquence de *k* symboles binaires d'information est représentée par le polynôme $i(x)$. Soit $g(x)$ un polynôme de degré s. Le mot de code correspondant à $i(x)$ est égal à$c(x) = x^s i(x) + r(x)$ où $r(x)$ est le reste de la division de $x^s i(x)$ par $g(x)$.

Ces codes permettent de détecter toute rafale (erreurs consécutives) d'erreurs de longueur inférieure ou égale à s.[9]

V. Conclusion

Ce chapitre était consacré sur les codes détecteurs et correcteurs d'erreurs en générale, et leurs origines c'est-à-dire le théorème d'information qui était énoncé par C.Shannon dans les années 1950.

Nous avons présenté aussi les deux familles des codes correcteurs d'erreurs (les codes en blocs et les codes convolutifs) proposant, de diverses manières pour le codage et le décodage.

Chapitre 2
Codes En Blocs

I. Introduction

Une grande famille de codes correcteurs d'erreurs est constituée des codes par blocs. Dans notre projet nous ne traiterons que ces types des codes.

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.

II. Définition générale

L'information de la source est mise en trames de longueur fixe que nous devons transmettre c'est le **message** ou **le mot initial.** Le codage prend ce message pour en faire un mot de code :

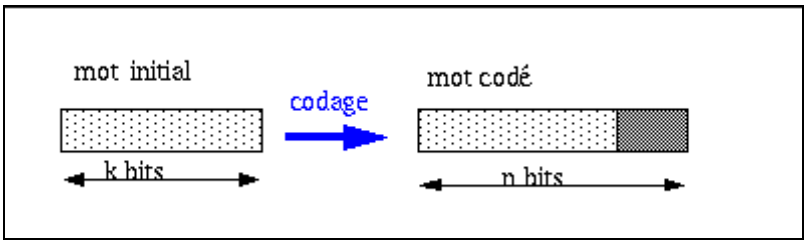


Figure 2.1 : le codage d'un message.[33]

Nous appelons mot de code, la suite de n bits obtenue après un codage (k, n). Le nombre n de bits qui composent un mot du code est appelé la longueur du code. La dimension k étant la longueur initiale des mots.

Le message est constitué de k caractères soit 2^k messages possibles. Le mot de code utilisé sera lui aussi de longueur fixe de n caractères soit 2^n mots de code possibles. Avec n > k il y'aura donc n-k caractères du mot de code qui sont redondants et serviront à traiter les erreurs éventuelles. Par ailleurs, $2^n > 2^k$ donc un certain nombre de mots de code ne correspondent pas à un message mais seulement à des erreurs de transmission.

On parle ainsi d'un **code de bloc (n , k)** et du rapport ou le rendement du code défini par le rapport entre k et n.

Un cas particulier des codes en blocs est celui des codes où le message apparaît explicitement sur ses k caractères c'est à dire "en clair". A côté de ces k caractères seront donc ajoutés n-k caractères redondants. Nous obtenons alors un code dit **code systématique** et ce sous ensemble que nous étudierons.

III. Poids d'un code

Le poids de Hamming d'un mot est le nombre de bits à 1 qu'il contient appelé aussi moment d'un code. La distance de Hamming entre deux mots de même longueur est définie par le nombre de positions binaires qui diffèrent entre ces deux mots. On l'obtient par le poids de Hamming de la somme binaire des 2 mots. La distance de Hamming d'un code est la distance minimum entre tous les mots du code.

Exemple :

- poids_de_Hamming(01001100) = 3,

A l'ensemble des mots d'un code on associe l'ensemble des poids qui est la distribution de poids du code. Un code est dit de poids fixe (ou poids constant) si tous les mots du code ont le même poids.

La capacité de détection (de correction) d'un code est définie par les configurations erronées qu'il est capable de détecter (corriger). Une erreur simple (resp. double, ou d'ordre p) affecte une seule (resp. 2, ou p) position(s) binaire(s) d'un mot. Pour qu'un code ait une capacité de détection (resp. correction) des erreurs d'ordre e, il faut que sa distance de Hamming soit supérieure à 1+e (resp. 1 + 2e).

Exemple :

- distance =3 ==> capacité de détection <= 2, capacité de correction <= 1.

IV. Exemples simples de code par bloc

5. Code de répétition

Une première idée intuitive est de transmettre deux fois chaque bit. Si on ne reçoit pas deux fois la même chose, il y a eu une erreur, mais on ne sait pas la corriger. Il faut donc le transmettre trois fois. On pourra ainsi corriger une erreur par groupe de trois bits, mais deux erreurs dans le même groupe conduiront à une interprétation erronée.

6. Code de parité

Rappelons que, la parité paire (impaire) : le poids de Hamming des mots du code est pair (impair).C'est un code systématique (k, k+1) dans lequel un bit (le bit de parité) est ajouté au mot initial pour assurer la parité. Son rendement est faible lorsque k est petit.

7. Parité longitudinale et transversal

Association d'un double codage de la parité :

- LRC : "Longitudinal Redundancy Check "
- VRC : " VerticalRedundancyCheck"

Le bloc de données est disposé sous une forme matricielle (k=a.b). On applique la parité (uniquement paire) sur chaque ligne et chaque colonne. On obtient une matrice (a+1, b+1).

```
          VRC
1010001 1
0110101 0
1000001 0
------------------
LRC 01001011
```

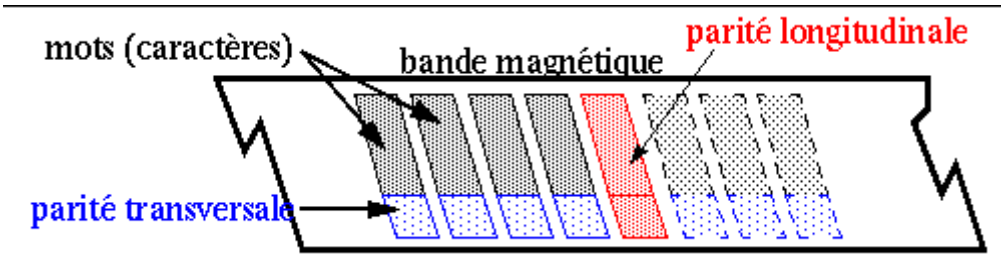


Figure 2.2 : parité longitudinale et transversale.[33]

Remarquons que :

- Le rendement est faible : a.b / (a+1).(b+1).
- Le délai de codage et décodage important : il faut recevoir tout le bloc.

V. Principes des codes linéaires

Définitions

Les codes linéaires sont des codes dont chaque mot du code (noté c) est obtenu après transformation linéaire des bits du mot initial (noté i ou message m).[32]

Comme nous avons vu dans le chapitre précédent ces codes sont caractérisés par leur matrice G(k, n) (appelée matrice génératrice) telle que :

$$i * G = c \quad \text{.....2.1}$$

Exemple :

$$[\,1\ 0\ 1\,].\begin{bmatrix}1&0&1&1\\0&1&0&1\\1&1&0&1\end{bmatrix} = [\,0\ 1\ 1\ 0\,]$$

Nous remarquons que, la multiplication de message [1 0 1] par la matrice génératrice nous a donné un mot de code (information + redondance = [0 1 1 0]) mais, il est lisible que l'information a été modifiée (à [0 1 1]), donc ce n'est pas un vrai codage.

Pour régler ce problème, il suffit de transférer la matrice génératrice à une autre matrice qui se commence par la matrice identité qui nous permet de conserver le même message, et pour cela il suffit d'appliqué la méthode d'élimination de Gauss par exemple.

Voilà une figure montrant ceci :

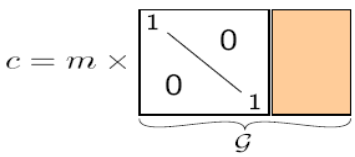


Figure 2.3 : G, matrice génératrice sous forme systématique.[33]

8. Propriétés des codes linéaires

La distance de Hamming d'un code linéaire est égale au plus petit poids de Hamming non nul des mots du code.

Si un code linéaire est systématique, sa matrice génératrice s'écrit :

G(k, n) = [I(k), P(k, n-k)] …….……...2.3

Exemples :

- Soit le code $C_1(4, 6)$ de matrice $G_1 = \begin{bmatrix} 1&0&0&0&1&0 \\ 0&1&0&0&0&1 \\ 0&0&1&0&1&0 \\ 0&0&0&1&0&1 \end{bmatrix}$.

Le mot de code c associé au mot initial I= [1 0 1 1] est le mot $c=i*G_1$ = [1 0 1 1 0 1]

9. Correction des erreurs

d. Correction par proximité

Le procédé de correction transforme un mot reçu et détecté comme erroné dans le mot de code le plus proche au sens de la distance de Hamming.

- Il peut exister plusieurs mots équidistants, on choisit un représentant.
- procédé de correction lourd.

Le nombre minimal de bits changeant entre chaque code est d_{min} la distance minimale du code.**[16]**

Appliquant les relations que nous avons présentées précédemment concernant la distance de Hamming nous avons e_D : le nombre maximal d'erreurs détectables est donc :

$e_D = d_{min} - 1$. ……...……2.5

Nous avons au maximum 2^k messages à transmettre donc 2^kmots de code corrects parmi les 2^npossibles.

Si X est le mot de code émis, E l'erreur de transmission et Y le mot de code reçu :

$$Y = X + E$$

S'il y a t erreurs de transmission donc la distance de Hamming entre Y et X est t.

Nous pouvons faire un schéma où un point représente un code, une croix un code correspondant àun message et la distance entre deux points, la distance de Hamming.

S'il y a t erreurs de transmission, tous les mots de codes possibles correspondant à cette erreur sont situés dans une sphère située autour du code réel transmis.

Soit deux codes X_i et X_j et t erreurs de transmission, si nous sommes dans la situation suivante :

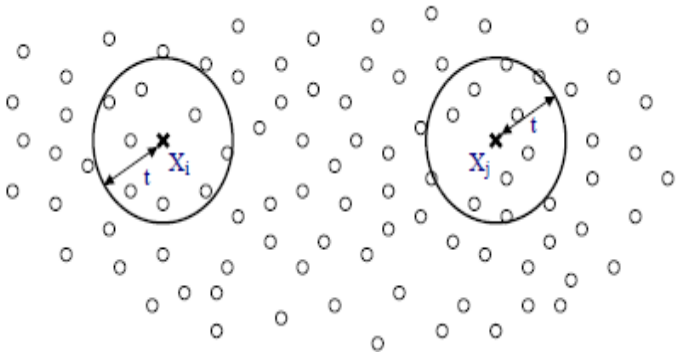


Figure 2.4 : correction sans ambiguïté. [16]

Tout code appartenant à une sphère peut être sans ambiguïté attribué au code le plus proche donc les erreurs de transmission sont toutes corrigeables.

Une seconde situation :

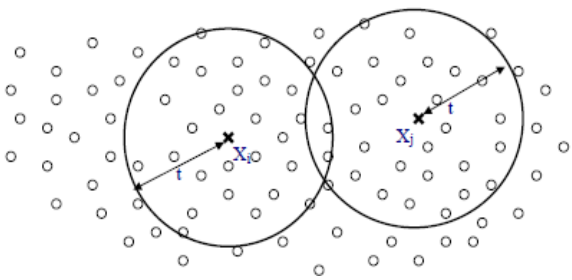


Figure 2.5 : correction avec ambiguïté. [16]

Dans ce cas, les codes appartenant à l'intersection des sphères ne peuvent être attribués. Pour éviter toute ambiguïté nous devrons donc respecter la condition :

$$\mathbf{2.t \leq d_{min}-1.} \quad \text{.....2.6}$$

t étant un entier, e_c le nombre maximum d'erreurs corrigeables sera :

$$\mathbf{e_C = Int\ [\ ½\ (d_{min}-1\ )\ ].} \quad \text{.....2.7}$$

Le nombre e_{DNC} d'erreurs détectables et non corrigeables est tel que : $\mathbf{e_{DNC}+ e_C= e_D}$. Compte tenu des relations précédentes nous aurons aussi la relation : $e_{DNC} \geq e_C$.

Si on prend par exemple $d_{min}= 7$ alors :

- Erreurs détectables $e_D = 6$ (=7-1).
- Erreurs corrigeables $e_C = 3$ (6/2).
- Erreurs détectables non corrigeables $e_{DNC} = 3$ (6-3).

Exemple :

Le code à répétition C(1, 4), distance de Hamming de C = 4.

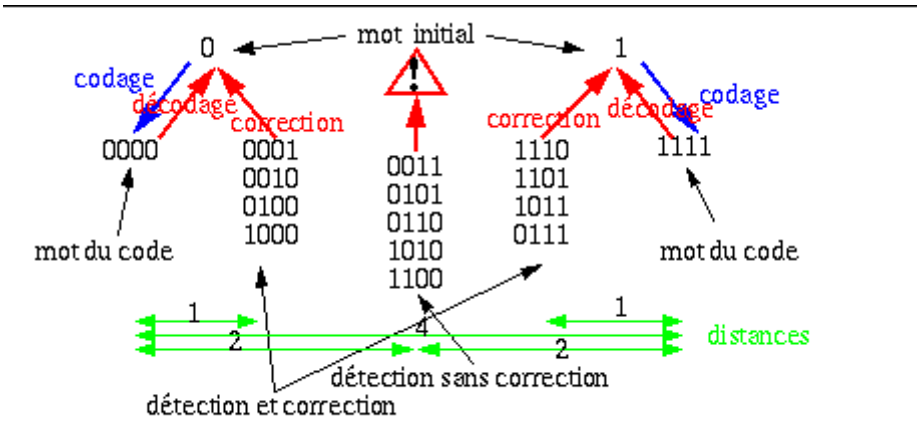


Figure2.6 : correction par proximité. [32]

e. Correction des erreurs, méthode matricielle

Il y a deux méthodes [19] de correction des erreurs :

1. détecter la forme des signaux reçus et attribuer le symbole : cela dépend des formes d'ondes et donc de la méthode de modulation choisie. C'est la méthode matérielle ("hard") de correction des erreurs.

2. interpréter la redondance d'un code pour lui attribuer un symbole : c'est la méthode dite logicielle ("soft") qui est celle que nous allons développer.

A la réception d'un mot de code, nous lui associons le message dont le mot de code est le plus proche au sens de la distance entre mots de code. C'est le maximum de vraisemblance dont l'illustration a été faite dans l'étude des erreurs corrigeables.
Pour un mot de code reçu le syndrome associé est :

$S_m = Y_m.H^t = E_m.H^t$…….……2.8

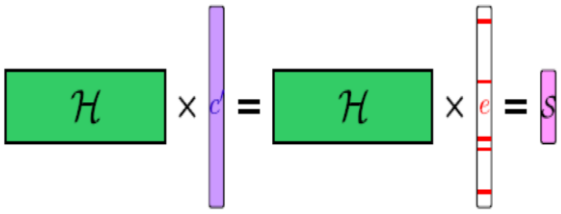


Figure 2.7 : calcule du syndrome.[6]

Chaque ligne de H^t est le syndrome d'une erreur simple. Pour un code (n , k) il y a nerreurs simples possibles et avec (n-k) bits de contrôle nous pouvons coder ($2^{n-k} – 1$) erreurs (-1 car"000…" n'est pas un code d'erreur) → il reste détectables (2^{n-k}- 1 - n) autres erreurs que des erreurssimples (erreurs doubles, triples, ….).

- le syndrome du mot reçu est identique à la colonne de la matrice de contrôle correspondant au bit à corriger.

- si l'on trie les colonnes de H suivant leur poids binaire croissant et que les poids de ses colonnes couvrent l'intervalle [1, 2^{m-1}] alors la valeur binaire du syndrome est égale au numéro de bit erroné.

Un code de Hamming est compact si les 2^{n-k} syndromes différents sont utilisés pour identifier les 2^{n-k-1} erreurs simples (+ syndrome nul !).

Donc pour corriger les erreurs les opérations s'ordonnent ainsi :

- Réception d'un mot de code Y_m.
- calcul du syndrome associé S_m.

- avec une table des erreurs, calcul de l'erreur la plus vraisemblable E_m' associée au syndrome.
- calcul du mot de code le plus vraisemblable : c_m' = Y_m E_m'(en binaire, le + est un ou exclusif).
- extraction du message X_m' de c_m'. **[26]**

❖ La méthode matricielle impose de travailler sur des mots de taille fixe, ce qui est un inconvénient majeur pour les réseaux informatiques où les messages sont de taille variable.

VI. Principes des codes polynomiaux

1. Présentation

Notation : tout vecteur peut être présenté sous une forme polynômiale **[14]** :

$$U = <u_0,u_1,u_2,...,u_n> <==> U(x) = u_0+u_1.x+u_2.x^2+...+u_n.x^n$$

- **Somme polynomiale :**
  - somme deux à deux de chaque coefficient (somme vectorielle).

  Exemple : $<x^3+1> + <x^4+x^2+1> = <x^4+x^3+x^2+1>$

- **Produit polynomiale :**
  - produit vectoriel.

  Exemple : $<x^3+1> .<x+1> = <x^4+x^3+x+1>$

- **Division polynomiale :**
  - idem

  Exemple : $<x^4+x^3+x^2+x+1> / <x+1> = <x^3+ x^2+1>$

2. Définition

Un code polynômial est un code linéaire systématique dont chacun des mots du code est un multiple du polynôme générateur (noté g(x))<==> les lignes de la matrice génératrice sont engendrées par le polynôme générateur.

Le degré du polynôme définit la longueur du champ de contrôle d'erreur.

Exemple

Soit le polynôme générateur : g(x) = x+1, intéressons-nous à un code C(2,3), donc les calculs se feront modulo x^3+1.

Première ligne : g(x). 1 = x+1

Deuxième : g(x).(x^2+1) mod (x^3+1) = (x^3+x^2+x+1) mod (x^3+1) = x^2+x.

Ce qui donne la matrice génératrice suivante : G(2,3) = $\begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

3. Capacité de détection d'erreurs

Un code polynômial (k, n) dont le polynôme générateur a plus d'un coefficient non-nul (donc il ne divise pas x_i, i<n) permet de détecter toutes les erreurs simples.

Si le polynôme générateur d'un code polynômial (k, n) a un facteur irréductible de trois termes (il ne divise ni x_i ni 1 + x_{j-i}, i<j<n), le code permet de détecter les erreurs doubles.

- par exemple : g(x) = x^3 + x + 1

Pour qu'un code polynomial détecte toutes les erreurs d'ordre impair, il suffit que son polynôme générateur ait (x+1) comme facteur.

- Par exemple : le code de polynôme générateur (x+1) qui est équivalent à la parité.

❖ **Capacité de détection des paquets d'erreurs :**

Un code polynômial (k, n) permet de détecter toutes les erreurs d'ordre l < = n-k (c'est-à-dire inférieur au degré du polynôme générateur), et la probabilité de ne pas détecter les erreurs d'ordre l>n-k est très faible.

4. Principe du codage

L'opération de codage consiste à ajouter les bits de contrôles à l'information i(x), et pour cela il ya plusieurs méthodes qui sont proposées, comme par exemple la méthode

de division euclidienne qui sert à obtenir un mot de code m(x) d'un code polynomial (k, n) de polynôme générateur g(x) associé au mot initial i(x) qui est défini par :

$c(x) = i(x).\ x^{n-k} + r(x)$, où $r(x)$ est le reste de la division de $i(x).x^{n-k}$ par le polynôme générateur g(x) (noté : $r(x) = (i(x).x^{n-k})\ mod\ g(x)$).

Toujours dans les opérations de codage, il y'a une autre méthode appelée « méthode matricielle » son principe est :Pour encoder un message m on le multiplie par une matrice génératrice G, on obtient un mot de code c,ensuite après passage par un canal bruité, une erreur e s'ajoute au mot de code c on obtient **c' = c + e**:

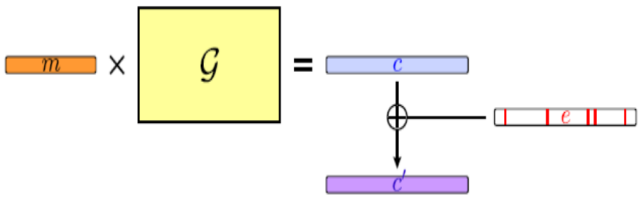


Figure2.8 : l'encodage.[6]

Remarque :

- Les r = n-k bits de r(x) (de degré < = n-k-1) forment les bits du champ de contrôle.
- Les bits de poids fort (de degré > n-k-1) forment le mot initial (code systématique).

5. Principe du décodage

A la réception, chaque mot reçu c'(x) est divisé par le polynôme générateur g(x).

Décoder dans un code C désigne l'action d'associer un mot de code (un élément du sous-espace vectoriel) à un mot de l'espace vectoriel. On cherche le plus souvent à décoder en associant à un mot le mot de code duquel il est le plus proche. Cependant il faut d'abord décider du sens que l'on veut donner à l'expression le plus proche.[5]

VII. Principes des codes cycliques

1. Définition

Un code cyclique (k, n) est un code linéaire (k, n) tel que toute permutation circulaire d'un mot du code est encore un mot du code.

Exemple

- Un code cyclique (1, 2) possède les mots de code suivants : {01, 10} ou {00, 11}, mais pas {01, 11}.
- Un code cyclique (1, 3) possède les mots de code suivants : {000, 111}.

2. Propriétés / autocorrection

- Un code cyclique (k, n) est un code polynomial dont le polynôme générateur divise $x_n + 1$.
- Les dispositifs de codage et de décodage sont identiques à ceux des codes polynomiaux.
- Les codes cycliques sont principalement utilisés pour leur capacité de correction.
- Les codes cycliques (k, n) dont le polynôme générateur est primitif et tel que $n=2^{n-k} + 1$, sont des codes de Hamming.[7]

VIII. Conclusion

Le chapitre deux était consacré sur une brève explication d'une seule branche des codes correcteurs d'erreurs appelé « les codes en blocs », et qui regroupent des différents types d'autre codes comme les codes linéaires, polynomiaux, cycliques.

Dans ce chapitre, nous avons éclairé aussi les relations qui existent entre les différents codes avec leurs principes et leurs méthodes de correction d'erreurs (codage et décodage).

Chapitre 3
Codes Reed Muller

I. Champs de Galois

1. Introduction

Le présent chapitre présentera au début une brève introduction aux mathématiques utilisées dans les codes correcteurs Reed – Muller. On introduira la notion de groupe, d'anneau et de champ dans les mathématiques abstraites. Par la suite, on se concentrera sur les « champs de Galois » utilisés pour ces types des codes correcteurs.

2. Groupe

Un groupe [31] G est un ensemble d'éléments défini par l'opération binaire•. Chaque paire d'éléments, a et b, a un seul élément c défini par l'opération binaire • dans G, $c = a \bullet b$.

Un groupe doit respecter les contraintes suivantes :

- Le groupe doit être fermé, pour n'importe quel élément a et b dans G, l'opération binaire $c = a \bullet b$ doit toujours retourner un élément dans le même groupe G.
- La loi associative doit être applicable ; $a\ (bc) = (ab)\ c$ pour tout a,b,c compris dans G.
- Pour n'importe quel élément a dans G, il doit exister un élément tel que $a \bullet e = e \bullet a = a$. L'élément e est appelé l'élément identité de G.
- Pour tout élément a dans G, il doit exister son élément inverse a' tel que $a \bullet a' = e$.

Un groupe est dit « abélien » si la loi de commutation est valable, $ab = ba$ pour tous les éléments compris dans G.

3. Anneaux

Un anneau [31] R est un ensemble d'éléments défini par deux opérations binaires appelées, addition et multiplication. Un anneau R est un groupe « abélien » selon l'opérateur d'addition. La multiplication doit respecter les contraintes suivantes:

- La multiplication doit être associative, a (bc)= (ab) cp our tout a,b,c compris dans R.
- La multiplication doit être distributive par rapport à l'addition, a (b + c)= ab + ac et(a +b) c= ac+ bc pour tout a,b,c compris dans R.

Un anneau est dit commutatif si sa multiplication est commutative, ab = ba pour tout a,b compris dans R.

Un anneau est appelé domaine d'intégrité s'il possède un élément unité pour la multiplication et qu'il n'admet pas de division par zéro.

- Il y a un élément unité pour la multiplication tel que $a1$ = $1a$ = a pour tout a compris dans R.
- Pour tout a,b en R,ab = 0, il y a alors la possibilité que a = 0 ou b = 0

4. Corps ou Champ

Un corps ou champ [11] C est un domaine d'intégrité dans lequel tous les éléments non nuls sont inversibles. Pour un a non nul dans C, il existe un élément a^{-1} dans C tel que aa^{-1} =1. Un corps possède toutes les propriétés définies au sous-titre 2 et 3.

5. Champs 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 La dénomination «champ de Galois » provient du mathématicien français Galois qui en a découvert les propriétés fondamentales.

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

5.1. Eléments du champ de Galois

Un « champ de Galois »[7] consiste en un ensemble de nombres, ces nombres sont constitués à l'aide de l'élément base α comme suit : $\mathbf{0, 1, \alpha, \alpha^2, \alpha^3, ..., \alpha^{N-1}}$

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 :

$$\alpha_{m-1}x_{m-1} + \alpha_{m-2}x_{m-2} + ... + \alpha x + \alpha \quad .....3.1$$

Avec : $\alpha_{m-1}...\alpha_0$: éléments bases du $GF(2)$(valeurs : 0,1)

Sur les « champs de Galois », on peut effectuer toutes les opérations de base. 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 (XOR). La soustraction effectuera la même opération qu'une addition, c'est-à dire, la fonction logique « XOR »[11].

La multiplication et la division seront des opérations modulo « grandeur du champ », donc $\mathrm{mod}(2^m - 1)$.

5.2. Addition dans les GF(2)

Dans ce point, on cherche à comprendre comment l'affirmation du titre (5.a) est possible. Considérons le tableau ci-dessous dans lequel on fait l'addition binaire entre les deux éléments *A* et *B* du $GF(2)$:

A	B	Reste	Résultat
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tableau 3.1 : addition de deux éléments dans GF(2).

En négligeant le reste dans le résultat final de l'addition, on constate que la somme entre deux éléments dans un *GF* (2) donne une addition modulo 2, c'est-à-dire, une fonction logique «XOR». Comme *GF* (2) est le champ de base, cette relation sera valable pour tous les champs dérivés, c'est-à-dire, pour *GF* (2^m).

5.3. La soustraction dans GF(2)

Ici, on cherche à comprendre comment une soustraction dans *GF*(2) correspond à faire une addition dans le même champ. Considérons le tableau suivante dans lequel on fait la soustraction binaire entre les deux éléments *A* et *B* :

A	B	Emprunte	Résultat
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Tableau 3.2 : soustraction de deux éléments dans GF(2).

On constate que la soustraction dans *GF* (2) effectue la même opération que l'addition **dans le même champ, c'est-à-dire une opération logique « XOR ».**

- **Polynôme primitif :**

Ce polynôme permet de construire le « champ de Galois » souhaité. Tous les éléments non nuls du champ peuvent être construits en utilisant l'élément α comme racine du polynôme primitif. Chaque *m* a peut être plusieurs polynômes primitifs *p*(*x*)**,** mais dans le tableau ci-dessous, on mentionne seulement les polynômes ayant le moins d'éléments.

Les polynômes primitifs pour les principaux « champs de Galois » sont les suivants:

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	$1+x+x^3+x^{12}+x^{16}$
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		21	
11		22	

12 13	$1+x^3+x^{10}$ $1+x^2+x^{11}$ $1+x+x^4+x^6+x^{12}$ $1+x+x^3+x^4+x^{13}$	23 24	$1+x^2+x^{21}$ $1+x+x^{22}$ $1+x^5+x^{23}$ $1+x+x^2+x^7+x^{24}$

Tableau 3.3 : polynômes primitifs dans *GF* (2^m)[23].

- **Dérivation formelle d'un polynôme dans un Champ de Galois :**

Un polynôme d'ordre *m* est défini comme:

$f(x) = f_m x^m + f_{m-1}x^{m-1} + \ldots + f_1 x + f_0$ **……3.2**

La dérivée formelle[18] du polynôme f(x) dans un « champ de Galois » *GF* (2^m)est définie avec les puissances paires de f(x) :

$f'(x) = f_1(x) + 2f_2(x) + 3f_3(x^2) + \ldots + mf_m(x^{m-1})$**……..3.3**

$f'(x) = f_1 + 3f_3x^2 + 5f_5x^4 + \ldots$

La dérivée est définie seulement par les puissances paires car les puissances impaires sont précédées de coefficients pairs qui annulent les termes :

2 = 1 + 1 = 0

4 = 2 + 2 =1 + 1 + 1 + 1 = 0

...

La dérivation doit respecter les règles suivantes :

- Si $f^2(x)$ divise $a(x)$ alors $f(x)$ divise $a'(x)$.
- La dérivation est une opération linéaire, elle doit respecter la règle de linéarité suivante :

$$[f(x)a(x)]' = f'(x)a(x) + f(x)a'(x) \text{……..} \quad \textbf{3.4}$$

6. Monômes et vecteurs sur F_2

Nous commençons par introduire le concept d'espaces vectoriels [4] sur F_2 et leur correspondance à certains anneaux de polynômes.

Notation :

- Nous utilisons la notation Z_n pour référer l'ensemble {0, 1, . . ., n - 1}.

- Nous utilisons une chaîne de longueur n avec des éléments de F_2 pour écrire un vecteur dans l' espace vectoriel $F(2)^n$. Par exemple, si nous avons le vecteur $\sigma = (1, 0, 1, 1, 0, 1, 0) \in F(2)^7$, Nous écrivons simplement $\sigma = 1011010$.
- Supposons que x et y sont des vecteurs dans l'espace vectoriel $F(2)^n$ avec : $x = (x_1, x_2, \ldots, x_n)$ et $y = (y_1, y_2, \ldots, y_n)$.

Soit α un élément de F_2, où nous appelons scalaire α. Ensuite, nous avons les opérations standards de l'addition et la multiplication d'un scalaire et vecteurs. Toute fois, nous avons également étendre l'ensemble des opérations de la manière suivante :

- addition d'un scalaire : $\alpha + x = (\alpha + x_1, \alpha + x_2, \ldots., \alpha + x_n)$
- complément d'un vecteur : $\overline{x} = 1 + x$
- Multiplication des vecteurs : $x*y = (x_1*y_1, x_2*y_2, \ldots, x_n*y_n)$

Notez que $(F(2)^n, +, *)$ est un anneau commutatif.

Supposons que nous avons donné un espace vectoriel $F(2)^{2m}$, Ensuite, nous considérons l'anneau $R_m = F_2[x_0, x_1, \ldots, x_{m-1}]$. Nous verrons bientôt qu'il existe une bijection entre les éléments de R_m et $F(2)^{2m}$ (En fait, un isomorphisme d'anneaux entre $(R_m, +, *)$ et $(F(2)^{2m}, +, *)$.

Définition 1 : Un monôme booléen **[2], [20]** est un élément $p \in R_m$ de la forme suivante :

$$P = x_0^{r_0} x_1^{r_1} \ldots x_{m-1}^{r_{m-1}}$$

Ou $r_i \in N$ et $i \in Z_m$. Un polynôme booléen est tout simplement, comme prévu, une combinaison linéaire (à coefficients dans F_2) Des monômes booléens; tout élément de R_m peut être considéré comme un Polynôme booléen.

Définition 2 **[8]** : Étant donné une valeur booléenne d'un monôme $p \in R_m$, On dit que p est sous forme réduite si elle est carré. Pour tout monôme booléen $q \in R_m$, Il est trivial de trouver la forme réduite de q par l'application:

$x_i x_j = x_j x_i$; comme R_m est un anneau commutatif

$x_i^2 = x_i$; comme 0*0 = 0, et 1*1 = 1

Un polynôme booléen sous forme réduite est tout simplement une combinaison linéaire de forme réduite des monômes booléennes (à coefficients dans F_2).

Exemple : Supposons que nous avons le polynôme Booléen $p = 1 + x_1 + x_0^5 x_2^2 + x_0 x_1^4 x_2^{101} \in R_3$. Ensuite, en appliquant les règles ci-dessus, pour que nous pouvons obtenir sa forme réduite p' tel que :

$P' = 1 + x_1 + x_0 x_2 + x_0 x_1 x_2$

Considérons la cartographie **[15]** $\psi: R_m \rightarrow (F_2)^{2m}$, Défini comme suit:

$$
\begin{aligned}
\psi(0) &= \underbrace{00\ldots0}_{2^m} \\
\psi(1) &= \underbrace{11\ldots1}_{2^m} \\
\psi(x_0) &= \underbrace{11\ldots1}_{2^{m-1}}\underbrace{00\ldots0}_{2^{m-1}} \\
\psi(x_1) &= \underbrace{11\ldots1}_{2^{m-2}}\underbrace{00\ldots0}_{2^{m-2}}\underbrace{11\ldots1}_{2^{m-2}}\underbrace{00\ldots0}_{2^{m-2}} \\
\psi(x_2) &= \underbrace{11\ldots1}_{2^{m-3}}\underbrace{00\ldots0}_{2^{m-3}}\underbrace{11\ldots1}_{2^{m-3}}\underbrace{00\ldots0}_{2^{m-3}}\underbrace{11\ldots1}_{2^{m-3}}\underbrace{00\ldots0}_{2^{m-3}}\underbrace{11\ldots1}_{2^{m-3}}\underbrace{00\ldots0}_{2^{m-3}} \\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
\psi(x_i) &= \underbrace{11\ldots1}_{2^{m-i}}\underbrace{00\ldots0}_{2^{m-i}}\ldots \\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots
\end{aligned}
$$

Pour tout monôme $p \in R_m$, Pour calculer $\psi(p)$, nous trouvons la forme réduite :

$$P' = x_{i1} x_{i2} \ldots x_{ir}$$

Ou $i_j \in Z_m$, $i_j = i_k \rightarrow j = k$, et $0 \leq r \leq m$. Ensuite :

$$\psi(p) = \psi(x_{i1}) * \psi(x_{i2}) * \ldots * \psi(x_{ir})$$

Pour tout polynôme $q \in R_m$, On peut écrire q comme:

$$q = m_1 + m_2 + \ldots + m_r$$

(Où m_i est un monôme de R_m, $m_i = m_j \rightarrow i = j$ et $0 \leq r \leq 2^m$). Ensuite :

$$\psi(q) = \psi(m_1) * \psi(m_2) * \ldots * \psi(m_r)$$

II. Principe Pratique de Reed – Muller

1. Historique

Les codes Reed Muller sont des codes correcteurs qui ont été découverts en 1954 par Muller [10] furent la première classe non triviale des codes capables de corriger des erreurs multiples. Reed [17] en a donné un algorithme de décodage trés performant par vote majoritaire qui permet de les décoder jusqu'à la moitié de leur distance minimale sur le canal de transmission.

Depuis, cette famille a été largement étudiée et généralisée aux corps finis de plus de 2 éléments. Historiquement, un code Reed-Muller d'ordre 1 en 5 variables, qui a 64 mots de longueur 32 et corrige 7 erreurs, a été utilisé par les sondes *Mariner* lancées par la NASA entre 1969 et 1973 pour assurer une transmission (numérique) correcte des photos de Mars[17].

Un code de cette famille est identifié à l'aide de deux paramètres, usuellement notés r et m, appelés respectivement ordre et nombre de variables. Ces paramètres interviennent dans la description utilisant les fonctions booléennes : le code binaire de Reed-Muller d'ordre r en m, que l'on note RM(r,m), est l'ensemble des tables de vérité des fonctions booléennes en m variables dont la forme algébrique normale (ANF) est de degré au plus r. lorsque l'alphabet est le corps fini à q éléments, il suffit de considérer les fonctions q-aires [1].

Les codes Reed Muller ont de nombreuses propriétés intéressantes, ils forment une famille infinie des codes, et les grands codes Reed Muller peuvent être construit à partir des codes plus petits. Cette observation, nous montrons que les codes de Reed Muller peuvent être définis d'une manière récursive [28].

2. Introduction

Les codes de Reed – Muller sont des codes de détection et de correction des erreurs. Ce sont des codes linéaires non cycliques contrairement aux codes BCH et les codes Reed Solomon.

Les messages sont divisés en blocs dont des informations redondantes vont être ajoutées à chaque bloc en utilisant la matrice génératrice, permettant ainsi de diminuer la possibilité de retransmission. La longueur des blocs dépend de la capacité du codeur. Malheureusement, les codes de Reed Muller deviennent plus faibles lorsque leurs longueurs augmentent.

Le décodeur traite chaque bloc et corrige les éventuelles erreurs en utilisant une forme d'un processus connu sous le nom vote de la majorité. A la fin de ce traitement, les données originales seront restaurées.

Typiquement, la transmission des données dans un canal avec cette méthode est effectuée ainsi :

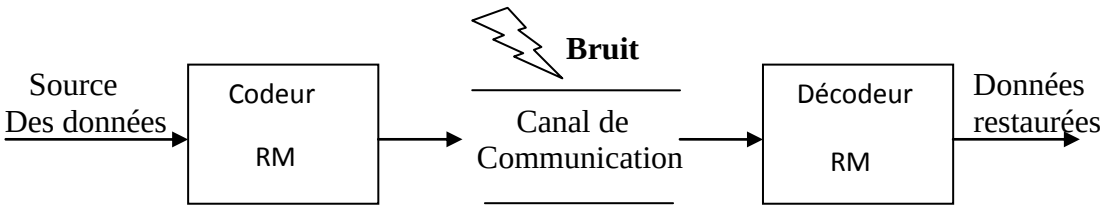


Figure 3.1 : Schéma général des Codes Correcteurs.

3. Vue simple des codes Reed Muller

3.1. Définition et première propriété

Le code Reed-Muller d'ordre r sur m variables est l'espace vectoriel des vecteurs de valeurs des fonctions booléennes sur m variables et de degré au plus r **[1]**.

Pour abréger, on notera un tel code RM(r,m). Si l'on se rappelle du point précédent, on voit qu'il s'agit d'un code de longueur 2^m, longueur que nous noterons n dans ce chapitre.

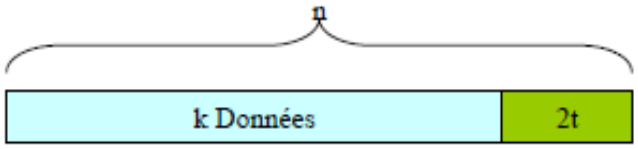


Figure 3.2: mot - code de Reed – Muller

La longueur maximale d'un code de Reed – Muller est définie comme :

$$n = k + 2t = 2^s - 1 \quad \text{…… 3.5}$$

Avec :

- K : nombre de symboles d'information
- $2t$: nombre de symboles de contrôle (la distance minimale)
- S : nombre de bits par symbole (s = m, le nombre de variables)

Nous supposerons ici que les points de $(F_2)^m$ sont énumérés par l'ordre usuel défini dans la première section du chapitre. La dimension de RM(r,m) est donc la même que celle des fonctions de degré au plus r sur m variables.

Une première propriété particulièrement importante des codes de Reed- Muller est leur structure récursive qui est connue sous le nom de décomposition (u, u + v).

3.1.1. Décomposition (u, u+v)

Les codes de Reed-Muller vérifient la relation de récurrence **[28]** suivante :

$$RM(r + 1, m + 1) = \{ u \mid u + v, \text{ avec } u \in RM(r + 1, m) \text{ et } v \in RM(r,m) \}.$$

Le symbole | désigne la concaténation de séquences binaires.

La preuve provient directement de la représentation à l'aide de l'ANF (Forme Algébrique Normale) d'une fonction booléenne. Pour une fonction f de RM(r+1,m+1) on peut ainsi écrire :

$F(X_1, \ldots, X_{m+1}) = U(X_1, \ldots, X_m) + X_{m+1}V(X_1, \ldots, X_m)$

Où U et V sont des polynômes à m variables. Il suffit pour cela de regrouper les termes de l'ANF qui font intervenir X_{m+1}. On sait de plus que le degré de U est d'au plus r+1 et que celui de V est d'au plus r. Aux deux polynômes U et V correspondent bien des fonctions booléennes de m variables qui appartiennent aux codes de la proposition. De plus, le vecteur des valeurs de f s'écrit bien u|u + v avec l'ordre choisi sur les monômes.

Cette décomposition récursive est particulièrement utile pour démontrer des résultats sur les codes de Reed-Muller ou même pour les décoder comme dans l'algorithme de Dumer que l'on mentionnera plus loin

Une illustration de l'utilisation de cette décomposition récursive est donnée dans la démonstration du lemme suivant qui nous sera utile plus loin. Remarquons d'abord que RM(m,m) contient tous les vecteurs de longueur 2^m et que RM(0,m) est le code à répétition de longueur 2^m.[28]

3.2. matrice génératrice

La matrice génératrice [8] du code RM(r,m) que nous noterons G_{RM} est la matrice dont les lignes correspondent à des vecteurs des valeurs des fonctions monômes de degré au plus r. La matrice génératrice est définit comme suit :

$$G_{\mathcal{RM}(r,m)} = \begin{bmatrix} \psi(1) \\ \psi(x_0) \\ \psi(x_1) \\ \vdots \\ \psi(x_{m-1}) \\ \psi(x_0x_1) \\ \psi(x_0x_2) \\ \vdots \\ \psi(x_{m-2}x_{m-1}) \\ \psi(x_0x_1x_2) \\ \vdots \\ \psi(x_{m-r}x_{m-r+1}\ldots x_{m-1}) \end{bmatrix}$$

Exemple :

- la matrice génératrice de RM(2, 3) est :

$$G_{\mathcal{RM}(2,3)} = \begin{bmatrix} \psi(1) \\ \psi(x_0) \\ \psi(x_1) \\ \psi(x_2) \\ \psi(x_0x_1) \\ \psi(x_0x_2) \\ \psi(x_1x_2) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

3.3. distance minimale

La distance minimale[12] du code RM(r, m) est $\mathbf{d = 2^{m-r}}$.

Le résultat peut se montrer par récurrence sur m en utilisant la décomposition (u, u+v). On verra également une démonstration à la section suivante grâce aux équations qui vont nous servir au décodage majoritaire.

3.4. Autres propriétés

Après la longueur, la dimension et la distance minimale, la propriété la plus importante d'un code est peut être sa distribution de poids, c'est-à-dire le nombre de mot de codes pour tous les poids de Hamming possibles. Malheureusement, pour les codes de Reed-Muller cette distribution est inconnue dans le cas général et n'est en fait connue que pour trés peu de paramètre [29].

4. Codage

4.1. Introduction

Nous présentons maintenant la méthode choisi pour encoder efficacement k bits d'information en un mot de code de taille n>k.

4.2. Théorie du codage

Rappelons que pour encoder k bits d'information d'un vecteur X en un mot de code de longueur $n = 2^m-1$ il suffit de calculer le produit $X * G_{RM}$. Ici, la forte structure de la matrice génératrice facilite l'encodage et le décodage de RM(r, m) comme on le verra plus loin.

Encoder un mot de RM(r, m) (c'est-à-dire passer d'un message de k bits à un mot de code de n bits) revient à calculer le vecteur de valeurs d'une fonction booléenne de degré au plus r donnée comme la liste de ses k coefficients non nuls.

$$C : (F_2)^{m+1} \rightarrow F$$

$$(u_0, \ldots, u_m) \rightarrow u_0 \cdot U + u_1 \cdot X_1 + \cdots + u_m X_m$$

Cette fonction est clairement injective, et comme $(F_2)^{m+1}$ est de dimension m+1, C est donc un code de dimension m + 1 et de longueur 2^m **[2]**.

- **Exemple** : Prenons m = 3. Une base de F est :

$$\delta_0 = \delta_{(0,0,0)}$$
$$\delta_1 = \delta_{(1,0,0)}$$
$$\delta_2 = \delta_{(0,1,0)}$$
$$\delta_3 = \delta_{(1,1,0)}$$
$$\delta_4 = \delta_{(0,0,1)}$$
$$\delta_5 = \delta_{(1,0,1)}$$
$$\delta_6 = \delta_{(0,1,1)}$$
$$\delta_7 = \delta_{(1,1,1)}$$

On notera qu'il est trés commode, lorsque l'on désigne δ_u d'identifier le vecteur binaire u et l'entier dont il est la décomposition binaire (par exemple, δ_3 et $\delta_{(1,1,0)}$)

Nous allons maintenant coder le message :

$$x = (0, 1, 1, 0) \in (F_2)^4$$

Le message codé correspondant à ce message est donc $C_3(x) = X_1 + X_2$. Calculons les coordonnées de $C_3(x)$ dans la base (δ_i) i€{0...7}. On a :

$$X_1 = \delta_1 + \delta_3 + \delta_5 + \delta_7$$

et

$$X_2 = \delta_2 + \delta_3 + \delta_6 + \delta_7$$

Par conséquent :

$$X_1 + X_2 = \delta_1 + \delta_2 + \delta_5 + \delta_6$$

Les coordonnées de $C_3(x)$ dans la base (δ_i) i €{0...7} sont donc :

(0, 1, 1, 0, 0, 1, 1, 0).

De même, on calcule la matrice génératrice de C :

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

(La première ligne correspond aux coordonnées de U dans la base (δ_i)i€{0...7}, la seconde aux coordonnées de X_1, la troisième aux coordonnées de X_2 et la dernière à celles de X_3). On vérifie que l'on a bien :

$$(0,1,1,0)\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} = (0,1,1,0,0,1,1,0)$$

5. Décodage

5.1. Introduction

Il existe différentes techniques disponibles pour le décodage des codes de Reed-Muller, mais l'un des plus courantes et les plus facilement réalisables est le décodage de vote majoritaire. Nous allons enquêter sur une forme de cette technique dans ce document.

5.2. Décodage selon l'algorithme de vote majoritaire

Avant de commencer, nous allons accorder une mention de capacité à la correction d'erreur des codes Reed-Muller. Donnons le code RM (r, m), la distance entre deux mots de code est 2^{m-r} **[25]**.

Pour corriger les ε erreurs, nous devons avoir une distance strictement supérieure à 2ε. Ainsi, dans le cas des codes Reed-Muller, puisque nous avons une distance de 2^{m-r}, Nous pouvons corriger max (0, 2^{m-r-1} -1) erreurs **[35]**. Nous allons voir maintenant l'algorithme de base pour le décodage en utilisant le vote majoritaire :

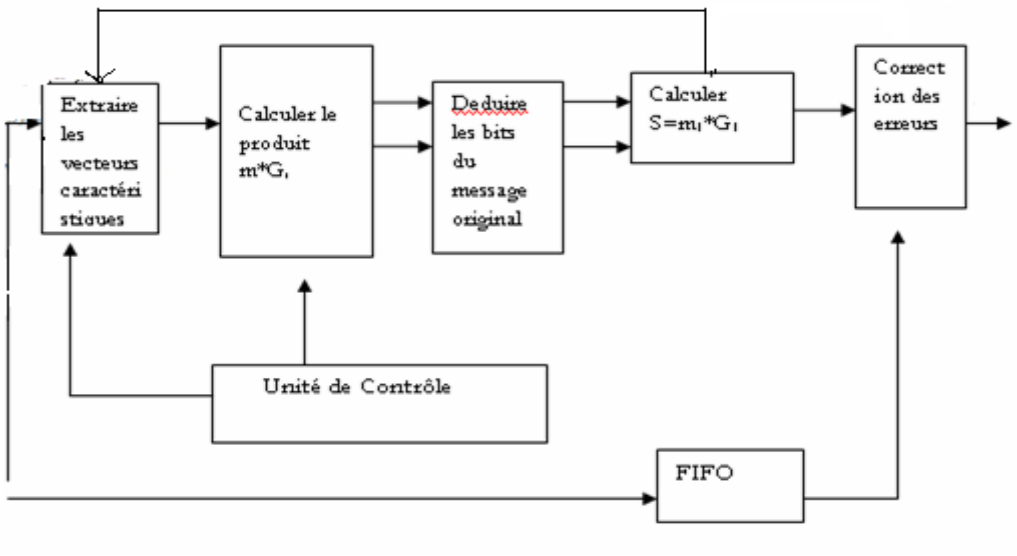


Figure3.3 : schéma du décodage

Avec :

m : le message reçu ;

G_i : les lignes de la matrice génératrice correspondantes aux vecteurs caractéristiques et leurs compléments ;

m_i : c'est la partie du message construit par la troisième étape ;

G_i : la partie de la matrice génératrice correspondante aux monômes de degré élevé traités ;

L'idée essentielle **[14]** derrière cette technique c'est que :

Pour chaque ligne de la matrice génératrice, nous tentons de déterminer par un vote majoritaire si cette ligne a été utilisée dans la formation de mot de passe correspondant à notre message ou pas.

Définition [5] : Soit P tout monôme de degré r en R_m avec P' la forme réduite. Ensuite, nous formons l'ensemble J des variables qui ne sont pas dans P' et leurs compléments. Les vecteurs caractéristiques de p sont tous les vecteurs correspondant aux monômes de degré m-r sur les variables de J. Tout monôme contenant une variable et son complément correspond au vecteur 0 ($\psi(x_i\overline{x_i}) = \psi(x_i) * \overline{\psi(x_i)} = 0$), ce qui implique que tout monôme contenant à la fois une variable et son complément est équivalent au

monôme de degré 0. Ainsi, sans perte de généralité, on ne considère que les monômes où les variables qui sont distincts (c - à – d, il n'apparaît pas des variables et leurs compléments).

Exemple : Si nous travaillons sur RM (3, 4), les vecteurs caractéristiques de $x_0x_1x_3$ sont les vecteurs correspondant aux monômes $\{x_2, \overline{x_2}\}$, les vecteurs caractéristiques de x_0x_2 sont les vecteurs correspondant aux monômes $\{x_1x_3, \overline{x_1}x_3, x_1\overline{x_3}, \overline{x_1x_3}\}$.

Notre algorithme de décodage commence par examiner les lignes correspondantes aux monômes de degré r. Nous commençons par le calcul de 2^{m-r} vecteurs caractéristiques de chaque ligne de la matrice génératrice, puis nous prenons le produit scalaire de chacun de ces vecteurs caractéristiques avec notre message reçu. Si la majorité des produits scalaires sont des 1, nous supposons que cette ligne a été utilisée dans la construction de notre mot de passe, et nous avons établi la position de notre vecteur message d'origine correspondant à cette ligne à 1. Si la majorité des produits scalaires sont des 0, alors nous supposons que cette ligne n'a pas été utilisée dans la formation de notre message, et donc, nous avons mis l'entrée correspondante dans le vecteur message original à 0.

Après, nous allons mener cette technique pour toutes les lignes correspondantes aux monômes de degré r, nous prenons le vecteur de longueur $\binom{m}{r}$ Correspondant à la partie du message que nous venons de calculer, et on le multiplie par $\binom{m}{r}$ Lignes de notre matrice génératrice qu'on a la prise en considération.

Cela nous donne un vecteur S de longueur n. Nous ajoutons S à notre message reçu, et procéder de manière récursive sur les lignes correspondantes aux monômes de degré r-1.

Exemple : Nous décodons le message U = 00110110 dans RM (2, 3), la matrice génératrice de ce code est définit comme suite :

$$G_{\mathcal{RM}(2,3)} = \begin{bmatrix} \psi(1) \\ \psi(x_0) \\ \psi(x_1) \\ \psi(x_2) \\ \psi(x_0x_1) \\ \psi(x_0x_2) \\ \psi(x_1x_2) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Selon l'algorithme, nous commençons par le bas de la matrice, et tout d'abord par les lignes correspondantes aux monômes de degré r=2.

- pour la ligne correspondante au monôme $\psi(x_1x_2)$, les vecteurs caractéristiques sont x_0, $\overline{x_0}$.

  U * $\psi(x_0) = 0$

  U * $\overline{\psi(x_0)} = 0$

  ➔ m = - - - - - - - - - - **0**

- pour la ligne correspondante au monôme $\psi(x_0x_2)$, les vecteurs caractéristiques sont x_1, $\overline{x_1}$.

  U * $\psi(x_1) = 1$

  U * $\overline{\psi(x_1)} = 1$

  ➔ m = - - - - - - - - - **10**

- pour la ligne correspondante au monôme $\psi(x_0x_1)$, les vecteurs caractéristiques sont x_2, $\overline{x_2}$.

  U * $\psi(x_2) = 0$

  U * $\overline{\psi(x_2)} = 0$

  ➔ m = - - - - - - - - **010**

Nous avons terminé le traitement des lignes correspondantes aux monômes de degré r = 2, nous calculons s :

$$s = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Nous ajoutons S à U on obtient 10010110, et on procède à traiter les lignes correspondantes à des monômes de degré r - 1 = 1.

Pour la ligne correspondante au monôme $\psi(x_2)$, les vecteurs caractéristiques sont x_0x_1, $\overline{x_0}x_1$, $x_0\overline{x_1}$, $\overline{x_0x_1}$

$U * \psi(x_0) * \psi(x_1) = 1$

$U * \overline{\psi(x_0)} * \psi(x_1) = 1$

$U * \psi(x_0) * \overline{\psi(x_1)} = 1$

$U * \overline{\psi(x_0)} * \overline{\psi(x_1)} = 1$

➔ m = - - - - - - - -**1 010**

Nous continuons à procéder de cette façon, et découvrir que le message original est représenté par le vecteur V = 0111010 (nous pouvons trivialement vérifier que c'est correct en vérifiant que $V \cdot G_{RM(2,3)} = U$).

6. Conclusion

Les codes de Reed-Muller peuvent être vus comme étant un polynôme représentant l'information à transmettre qui a subi à un traitement, ce qui crée l'information de redondance requise pour la correction d'erreurs. Ainsi, par le traitement effectuer sur l'information et les redondances ajoutées, on a un surplus d'informations qui permettra de corriger les portions erronées.

Chapitre 4 Implémentation logicielle

1. Objectif du projet :

Le but de ce projet informatique est de mettre en pratique un code correcteur d'erreur, à savoir le code Reed Muller et de voir comment se comporte ce type de code en traitant des blocs d'information de taille fixe, ainsi de faire une synthèse comparative entre ces types des codes et les codes Reed Solomon qu'on a implémenter dans un autre projet.
Plutôt que de travailler sur des fichiers sonores ou textes dont les erreurs ne sont pas apparentes, et afin de mettre en exergue la correction, nous avons travaillé sur des images.
Notre projet donc consiste à programmer les modules suivants :

- Un lecteur de fichiers au format BMP
- Un codeur RM
- Un décodeur RM, avec correction d'erreurs
- Un générateur d'erreurs aléatoires pour simuler des erreurs de lectures de bits
- Une interface graphique pour bien visualiser le phénomène, avec affichage de l'image originale, de l'image erronée ou brouillée et de l'image décodée avec correction

Notre choix s'est fait sur une image de taille 240×240 pixels, au format BMP pour leur simplicité. Donc notre application sert à afficher trois images de même dimension côte à côte sur l'écran, la première représente l'image originale (sur laquelle nous avons appliqué le code Reed-Muller), la deuxième c'est l'image brouillée et puis l'image corrigée.

2. Outils Utilisés :

2.1. Langage de Programmation C++ :

C'est un langage de programmation de haut niveau développé en 1983 Bjarnes Stroustrup aux Laboratoires Bell. Il s'agit d'une version orientée objet de son prédécesseur C. C'est un langage très utilisé pour développer des applications graphiques (comme celles qui tournent sur les environnements Windows et Macintosh).

On a été plus tenté par ce langage pour ses fonctionnalités remarquables à savoir :

- Opérateurs de gestions de bits;
- les opérateurs new et delete pour la gestion d'allocation mémoire ;
- les références .
- etc…

2.2. Environnement de développement Code Blocks :

Code**::**Blocks est un IDE (environnement de développement intégré) gratuit pour le développement en C/C++. Il est facilement configurable à l'aide d'extensions et est mis à jour quasi-quotidiennement.

Code::Blocks est un IDE libre et gratuit, disponible pour Windows, Mac et Linux.

C'est un environnement de développement intégré, libre et multiplate-forme. Il est écrit en C++ grâce à la bibliothèque wxWidgets.

- **Brève description**

Code::Blocks se veut simple d'utilisation, mais peut se révéler très complet si nous allons fouiller un peu dans les options. Il est très personnalisable, et extensible, grâce à son architecture de plug-ins, dont la plupart sont inclus dans l'archive et l'installateur. Vous n'aurez donc pas à les installer un à un.

2.3. Simple DirectMedia Layer :

Simple DirectMedia Layer (SDL) est une bibliothèque très utilisée dans le monde de la création d'applications multimédias en deux dimensions comme les jeux vidéo, les démos graphiques, les émulateurs, etc. Sa simplicité, sa flexibilité, sa portabilité et surtout sa licence GNU LGPL contribuent à son grand succès.

Cette bibliothèque se retrouve sous de nombreux systèmes d'exploitation, comme Linux, Windows, Mac OS, Mac OS X, ….

Elle est écrite en C, elle peut cependant être utilisée par de nombreux langages de programmation, comme l'Ada, C\C++, C#, Ch, D, Eiffel, Erlang, Euphoria, Guile, Haskell, OCaml, Java, Lisp, Lua, ML, Objective C, Pascal, Perl, PHP, Pike, Pliant, Python (grâce à Pygame), Ruby et Smalltalk.

En outre, la SDL gère :

- l'affichage vidéo ;
- les évènements ;
- l'audio numérique ;
- la gestion des périphériques communs comme le clavier et la souris et aussi le joystick ;
- la lecture de CD-Audio ;
- le multithreading ;
- l'utilisation de timers précis ...

3. Réalisation du prototype

Comme nous avons mentionné plus haut, nous avons utilisé le langage C++ qui couvre plusieurs librairies parmi elles, la librairie SDL qui est libre et gratuite, cette librairie offre la possibilité de dessiner pixel par pixel ou encore d'afficher des images.

Rappelons que notre travail est effectué sur une image bitmap de 240×240 pixels, et chaque pixel est déclaré en Uint32. D'une autre façon, chaque pixel est composé de quatre composantes : R, G, B, A (rouge, vert, bleu, et l'élément de transparence alpha), chacune de ces composantes est déclarée en Uint8, qui est donc une valeur entière allant de 0 à 255.

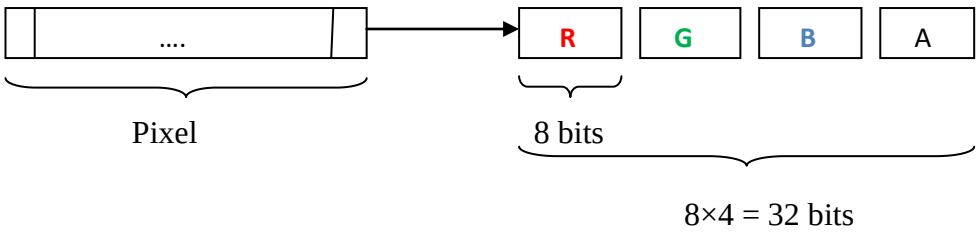


Figure 4 .1 : Structure d'un pixel

Pour appliquer les codes Reed – Muller il faut d'abord préciser ces paramètres. Dans notre cas, nous avons traité une image de 240×240 = 57 600 pixels = 1843 200 bits = 230 400 symboles.

Les codes correcteurs Reed–Muller appartiennent à l'ensemble des codes en blocs, et suivant leurs principes, il faut répartir l'information (image) en plusieurs blocs.

Nous avons scindé notre image en 1033 blocs, chaque bloc se compose de 255 symboles ou les 223 premiers sont réservés pour l'information pure, et les 32 derniers symboles pour la redondance, donc notre code est défini comme suit : RM (3 , 8), avec :
3 : c'est le degré maximum des monômes

8 : c'est le nombre des variables avec $n = 2^m - 1$ ($255 = 2^8-1$).

$d_{min} = 2^{m-r} = 32$ ce qui implique $d_{min}/2 = 16$ c'est à dire que notre code correcteur RM (3 , 8) permet de détecter 32 erreurs et corriger 16 erreurs au maximum.

Par ailleurs, rappelons que les codes Reed – Muller sont basés sur les champs de Galois, et puisque chaque symbole est codé sur s = 8 bits ($n = 2^S-1$) donc nous travaillons dans un GF(2^8).

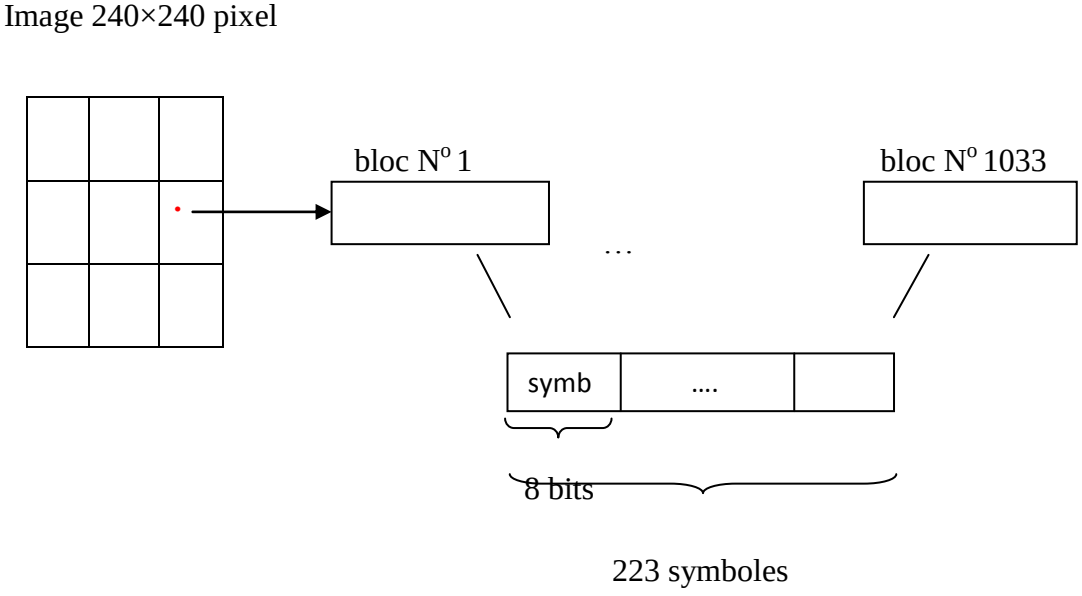


Figure 4.2 : Répartition de l'image

Notre mot de code étant de 255 symboles, nous aurons suivant le principe du code les 256 éléments du corps alpha suivant :

alpha = {"01", "02", "04", "08", "10", "20", "40", "80", "1D", "3A", "74", "E8", "CD", "87","13","26","4C","98","2D","5A","B4","75","EA","C9","8F","03","06","0C","18","30","60","C0","9D","27","4E","9C","25","4A","94","35","6A","D4","B5","77","EE","C1","9F","23","46","8C","05","0A","14","28","50","A0","5D","BA","69","D2","B9","6F","DE","A1","5F","BE","61","C2","99","2F","5E","BC","65","CA","89","0F","1E","3C","78","F0","FD","E7","D3","BB","6B","D6","B1","7F","FE","E1","DF","A3","5B","B6","71","E2","D9","AF","43","86","11","22","44","88","0D","1A","34","68","D0","BD","67","CE","81","1F","3E","7C","F8","ED","C7","93","3B","76","EC","C5","97","33","66","CC","85","17","2E","5C","B8","6D","DA","A9","4F","9E","21","42","84","15","2A","54","A8","4D","9A","29","52","A4","55","AA","49","92","39","72","E4","D5",B7","73","E6","D1","BF","63","C6","91","3F","7E","FC","E5","D7","B3","7B","F6","F1","FF","E3","DB","AB","4B","96","31","62","C4","95","37","6E","DC","A5","57","AE","41","82","19","32","64","C8","8D","07","0E","1C","38","70","E0","DD","A7","53","A6","51","A2","59","B2","79","F2","F9","EF","C3","9B","2B","56","AC","45","8A","09","12","24","48","90","3D","7A","F4","F5","F7","F3","FB","EB","CB","8B","0B","16","2C","58","B0","7D","FA","E9","CF","83","1B","36","6C","D8","AD","47","8E","01"}.

Remarque :

- Les éléments de Galois sont déterminés selon le principe suivant : 0, 1, α, α^2, …, et l'indice de chaque « alpha » représente sa puissance.
- Remarquons ainsi que ces valeurs sont en hexadécimal mais entre deux guillemets ; qui selon le langage de programmation C++ indique que ce sont des chaînes des caractères.

D'un autre côté, dans notre programme nous avons effectué les opérations d'addition, multiplication, division, XOR en binaire, donc pour éviter toute erreur au moment de l'exécution : à chaque utilisation d'un élément de ce champ, il doit être convertit en une suite de « 0 » et de « 1 » pour représenter l'équivalent binaire.

Par la suite, l'opération inverse s'avère indispensable c'est-à-dire une conversion de binaire à l'hexadécimale.

Les éléments du champ de Galois étant déclarés et les conversions étant faites, nous devons pour toute opération effectuée respecter les propriétés des champs de Galois comme par exemple :

- un XOR, c'est l'équivalent de l'addition et la soustraction.

Jusqu'à maintenant, nous n'avons pas approfondi notre programme, nous avons juste énoncé les éléments de Galois et quelques fonctions qui sont nécessaire dans nos calculs ultérieurs.

Rappelons que notre application sert à afficher trois images, chaque caractère occupant 1 octet en place mémoire ; notons qu'une seule des trois images affichées est composée de 57 600 pixels et que chaque pixel sur 32 bits (soit quatre symboles de 8 caractères chacun).En conséquence notre programme consomme énormément d'espace mémoire.

Pour une meilleure gestion de celle-ci, nous avons divisé notre application en cinq programmes (enregistrement de l'image, codage, brouillage, décodage et enfin l'affichage) en enregistrant à chaque fois les résultats obtenus dans des fichiers en mode écriture ou en mode ajout selon la nature de nos algorithmes, ainsi l'utilisation des tableaux dynamiques dont les programmes sont nécessaires.

Les étapes de notre travail consistent donc en ce qui suit :

3.1. Enregistrement de l'image :

Pour enregistrer l'image bitmap de rés0olution 240.240 pixels (sur laquelle nous avons appliqué le codage), nous avons utilisé des fonctions propres à la SDL, nous citons :

- SDL_Init(SDL_INIT_VIDEO), pour initialiser la SDL ;
- SDL_SetVideoMode, pour initialiser l'écran ;
- SDL_LoadBMP, qui sert à charger l'image ;

Nous avons utilisé une fonction obtenir pixel qui nous permet d'avoir la valeur du pixel qui est sur 32 bits.

Les composantes de chaque pixel vont être extraites à l'aide de l'instruction :

SDL_GetRGBA (pixel, originale->format, &r, &g, &b, &a);Et l'ensemble de ces composantes va être enregistrée dans un fichier.

3.2. Codage de l'information :

Pour comprendre comment la partie d'implémentation du codage fonctionne, nous devons interpréter la définition mathématique du codage :

$$c(x) = G_{RM} * i(x) \quad \text{.....4.1}$$

Avec :

$i(x)$: vecteur d'information

G_{RM} : matrice génératrice

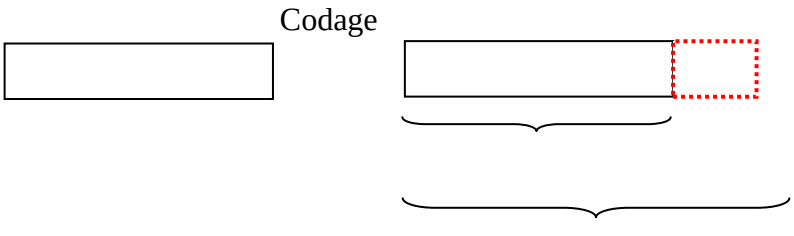
Il y a plusieurs méthodes utilisées pour la construction de la matrice génératrice, dans notre application nous avons choisi la manière définit dans le chapitre précédent.

Le codage est systématique, nous devons avoir les informations dans le degré élevé du mot de code de sortie puis les symboles redondants, et pour ceci généralement la matrice génératrice va être transformer à une autre matrice équivalente qui se compose par la matrice diagonale puis l'information en utilisant la méthode de Gauss par exemple.

Notre matrice génératrice est caractérisée par les monômes $\psi(x_i...x_j)$ qui représentent les lignes de cette matrice ou les 8 premiers corresponds aux fonctions de 8 variables choisi $\{x_0, x_1, \ldots, x_7\}$, ensuite on fait la concaténation des variables ou le degré maximum égale à 3, donc notre G_{RM} est défini comme suite :

$G_{RM}(3, 8) = \{ \psi(1), \psi(x_0), \psi(x_1), \psi(x_2), \psi(x_3), \psi(x_4), \psi(x_5), \psi(x_6), \psi(x_7), \psi(x_0x_1), \psi(x_0x_2), \psi(x_0x_3), \ldots\ldots, \psi(x_5x_6x_7)\}$.

Le codage consiste à ajouter 32 symboles de redondance à chaque 55 pixel (223 symboles) de l'image.



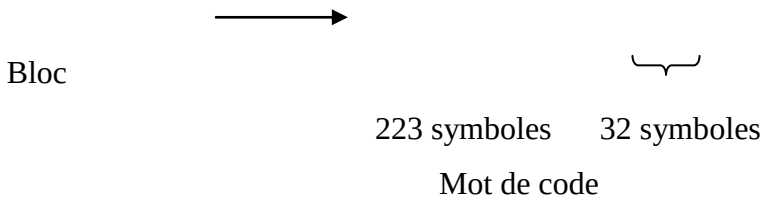


Figure 4.3 : l'encodage

3.3. Brouillage de l'information codée :

Les codes correcteurs Reed –Muller servent à détecter et corriger les erreurs survenues au cours de la transmission des données, mais dans notre travail nous avons opté pour la simulation d'un brouillage aléatoire afin de remplacer le bruit généré dans le canal de transmission.

Nous avons considéré le bruit comme étant une valeur de 8 bits de type caractère.

Nous avons fixé un tableau de 15 valeurs :

Bruit = {"00000001", "01010101", "10101010", "10000000" ,"00100100", "01000000", "00100000", "01000101", "00010000", "00001000", "00000100", "00000010", "01001001", "00010001", "10000100"}.

Avant de brouiller l'information, il faut d'abord récupérer le fichier « codage.txt ».

A l'aide de la fonction prédéfinie « rand », nous générons pour chaque symbole un nombre aléatoire entre 0 et 1 que nous devons le comparer avec un seuil.

Ce nombre est évalué selon :

K = rand ( ) * 1.0 / RAND_MAX ……..4.4

Pour savoir si le symbole doit être changé ou non, nous devons comparer le nombre K avec un seuil que l'utilisateur doit le définir.

Si le nombre généré K est supérieur au seuil, alors le symbole ne va pas être changé.

Et s'il est inférieur au seuil alors ce symbole devra être transformé par une nouvelle valeur qui est calculé en effectuant l'opération suivante :

Chaine XOR bruit[b] ……..4.5

Avec :

- Chaine : est le symbole du fichier qui doit être brouillé
- Bruit[b] : est une valeur du tableau bruit que nous avons défini plus haut.

Où b est calculé selon l'équation :

b = rand ()%14 ……4.6

Et ceci va être enregistré dans un nouveau fichier que nous avons appelé « brouillage ».

Nous tenons à signaler que le degré du brouillage est corrélé avec le seuil, plus le seuil est élevé, plus de bruit seront injecté et par conséquent l'image sera plus perturbée.

Cette étape s'achève par l'affichage d'une image brouillée en effectuant une opération inverse du codage, c'est-à-dire afficher l'image en omettant la redondance ajoutée lors du codage.

3.4. Décodage de l'information brouillée :

La phase de décodage est la plus importante dans les codes correcteurs d'erreurs, elle sert à corriger les erreurs ou il y a eu le bruit que nous avons généré.

Généralement, la manière de décodage (au niveau du récepteur) repose sur la manière de l'encodage qui se fait au niveau du l'émetteur, et puisque ce dernier (le codage) nécessite la matrice génératrice ($C(x) = G_{RM} * i(x)$) effectivement le décodage se base aussi sur cette matrice.

Donc dans la réalisation du décodage, il est nécessaire de récupérer les fichiers « brouillage » et « matrice génératrice ».

Pour décoder un mot de code passé par un canal de transmission bruité selon l'algorithme du vote de la majorité plusieurs étapes sont nécessaires :

1. traiter les lignes de la matrice génératrice correspondantes aux monômes de degré r = 3 (évidemment commençant par le bas de cette matrice), et extraire les vecteurs caractéristiques et leurs compléments. Pour ceci nous avons fixé un ensemble J qui couvre les variables du monôme, ensuite on déduit les variables

qui n'appartiennent pas à cet ensemble avec leurs compléments et on les met dans un autre ensemble J' pour la formation des vecteurs caractéristiques.

2. calculer le produit entre le message reçu et les lignes correspondantes aux vecteurs caractéristiques (ainsi leurs compléments) pour savoir si cette ligne a été utilisée dans la construction du mot de passe ou pas, pour ceci si la majorité des produits égale à 1 alors cette ligne est utilisable sinon elle n'est pas utilisée dans la formation du mot de passe.
3. déduire le message original bit par bit selon les résultats des produits calculer dans l'étape précédente. A chaque fois on rempli un bit dans le message originale, soit 0 soit 1 selon le résultat du produit.
4. calculer le vecteur $S = m_1 * G_{1RM}$ avec :

   m_1 : la partie du message formuler à partir des monômes de degré r = 3

   G_{1RM} : la matrice génératrice qui couvre juste les lignes correspondantes aux monômes de degré r = 3

5. comparer S avec le message reçu, s'il est le même on conclu que S c'est le message original, sinon on passe à l'étape suivante.
6. lorsque le traitement des lignes correspondantes aux monômes de degré r = 3 est terminé, on passe au degré r-1 et refaire les mêmes étapes. Le critère d'arrêt lorsque r = 1 et le message originale est formulé.

3.5. Affichage des images :

Cette étape regroupe tous les résultats des programmes précédents c'est-à-dire elle affiche les trois images : originale, brouillée, et puis corrigée dans une petite interface crée par l'instruction suivante :

ecran = SDL_SetVideoMode(800,300, 32, SDL_SWSURFACE);

L'affichage de chaque image consiste à effectuer l'opération inverse du premier sous-programme (l'enregistrement de l'image). Pour se faire nous devons d'abord récupérer le contenu des deux fichiers brouillage.txt et decodage.txt

Dans cette étape nous effectuons les opérations suivantes :

- Initialiser la SDL : SDL_Init(SDL_INIT_VIDEO);
- initialiser l'écran : SDL_SetVideoMode;

- déterminer le mode d'exécution : SDL_SWSURFACE;
- charger l'image originale : SDL_LoadBMP;
- Créer des surfaces pour les deux autres images (brouillée et corrigée) : SDL_CreateRGBSurface;
- Le brouillage et décodage se font sur une image codée, donc il faut éliminer tous les symboles redondants de chaque bloc.
- Reconstituer chaque pixel à partir de quatre symboles par l'instruction : SDL_MapRGBA;
- Utiliser la fonction « definirpixel », qui nous permet de coller le pixel sur la surface;
- Copier les trois images à l'écran par : SDL_BlitSurface;
- Coller les trois images à l'écran : SDL_Flip;
- Libérer l'espace mémoire occupé par les trois images;
- Quitter la SDL par : SDL_Quit;

Nous avons jugé utile d'exposer une démonstration des résultats de notre code correcteur, avec différentes images et en modifiant couramment la valeur du seuil afin de tester la capacité de correction ainsi que les limites de décodage.

N.B : Notons que notre application a été faite sur un simple PC de 2 GO de RAM, CPU Dual-Core 2.60 GHz et sur un système d'exploitation Windows 7.

Concernant les résultats de notre application nous avons les figures suivantes :

- Avec un seuil $\alpha = 0.02$:

❖ Avec un seuil α = 0.125 :

❖ Avec un seuil α = 0.2 :

Nous avons appliqué notre code correcteur sur une autre image de même résolution que la première et avec la même structure du pixel.

- Avec un seuil $\alpha = 0,1$

- Avec un seuil $\alpha = 0,3$

- Avec un seuil $\alpha = 0,5$

Discussion sur les résultats :

- L'image a été convenablement codée et brouillée;
- On constate une correction des images brouillées, avec une nette amélioration par rapport à l'image reçue, notons aussi que lors de la présence de bruit très rapproché, le décodeur peine à donner de bon résultats, aussi le temps d'exécution est très important (1h 45 à peu près) pour une image 240×240.
- La correction des erreurs est limitée, rappelons que notre code permet de détecter 32 erreurs et corriger 16 au maximum.
- plus le seuil de brouillage augmente, moins la correction est bonne.

Comparaison entre les codes Reed Solomon et les codes Reed Muller :

Nous avons les résultats d'implémentation des codes correcteurs Reed Solomon réaliser dans un ancien projet amélioré car les codes reed solomon implémenté dans le projet précédent est de paramètre (255,240) ou il peut détecter 16 erreurs et corriger juste 7 erreurs au maximum, par contre dans ce projet nous avons implémenté les codes reed muller de paramètre (255,223) c'est-à-dire il peut détecter 32 et corriger 16 erreurs, et pour pouvoir faire une synthèse comparative entre ces codes et les codes reed solomon afin de savoir quel type de ces deux codes correcteurs est plus performant que l'autre, nous avons amélioré ces codes dont les quels ils ont les même paramètres et la même capacité de correction que les codes reed muller.

Pour comparer les deux codes correcteurs reed solomon et reed muller, nous avons appliqué ces deux codes sur la même image avec même seuil de brouillage et les mêmes

paramètres c'est-à-dire les deux codes permettent de détecter et corriger le même nombre d'erreurs. Pour ceci voici les résultats obtenu, ou la première représente le résultat de l'implémentation des codes Reed Muller et la deuxième pour les codes Reed Solomon :

- Avec un seuil $\alpha = 0.125$ nous avons eu les résultats suivants

- Avec un seuil $\alpha = 0,5$

Commentaires et discussions :

- Les images ont été convenablement codées, brouillées et corrigées par les deux codes correcteurs Reed Solomon et Reed Muller.
- L'algorithme de décodage des codes Reed Solomon est plus efficace que l'algorithme des codes Reed Muller, car remarquons que les images corrigées par les codes Reed Solomon sont plus lisible par rapport aux images décodées par les codes Reed muller.
- Les deux codes correcteurs nécessitent un temps d'exécution très important pour une image de 240*240, ainsi une RAM de taille 2GO au minimun.
- Les deux codes correcteurs peinnent de corriger les erreurs lorsque le seuil de brouillage est très élevé.

Conclusion Générale

CONCLUSION ET PERSPECTIVES

La théorie des codes correcteurs d'erreurs est présentée dans l'ensemble des transmissions numériques de nos jours. Il est indispensable d'avoir des algorithmes de décodages efficaces, et les recherches actuellement vont vers une accélération et une amélioration des algorithmes existants.

Nous avons présenté ici des notions importantes de la théorie des codes correcteurs d'erreurs et précisément, les codes Reed - Muller qui sont des codes correcteurs basés sur le découpage de l'information en blocs linéaires.

Les codes de Reed – Muller permettent de corriger des erreurs et des effacements grâce à des symboles de contrôle ajoutés après l'information. Ce document traite l'aspect mathématique du codage et du décodage de ces codes.

Dans les codes Reed – Muller, le décodage représente la tâche la plus complexe, La complexité du codeur/décodeur dépend du type de codage choisi et des algorithmes choisis pour le décodage.

❖ *Perspectives*

La performance des codes correcteurs d'erreurs repose sur les algorithmes de codage et décodage établis, ce qui nous implique qu'il y a toujours des recherches et des améliorations dans ce domaine. Les futures améliorations pour notre projet sont :

- Ajouter une étape d'entrelacement qui consiste à disperser ou distancer les erreurs afin d'avoir une meilleurs correction.
- Rechercher d'autres algorithmes plus performants de codage/décodage, et tester diverses architectures.

Liste Des Références

BIBLIOGRAPHIE & WEBOGRAPHIE

[1] : Assmus. Les codes Reed Muller. DMATH : Mathématique discrète, 107, 1992.

[2] : Assmus, Jr., EF et Key, dessins et modèles JD et leurs codes. Syndicat de la Presse de l'Université de Cambridge, Cambridge, 1992.

[3] : Badrikian Josèphe mathématique pour téléinformatique, codes correcteurs, principes et exemples

[4] : Cameron, PJ et Van Lint, dessins et modèles JH, graphiques, codes, et leurs liens. Cambridge University Press, Cambridge, 1991.

[5] : Cedric Tavernier. Testeurs, problèmes de reconstruction uni variés et multi variés, et application à la cryptanalyse du DES. PhD thesis, Ecole doctorale de l'école polytechnique, Janvier 2004.

[6] : Christophe Ritzenthaler, codes correcteurs d'erreurs.

[7] : C. K. P Clarke, Reed – Solomon error correction, British Broadcast Corporation

[8] : Cooke, Ben. Reed-Muller Codes correcteurs d'erreurs. MIT Journal de premier cycle de Math- matiques Volume 1, MIT Département de Mathématiques, 1999.

[9] : Cory S.Mondlin, Error control coding in DSL system, CRC Press LCC, 2004

[10] : D. E. Muller. Application of Boolean algebra to switching circuit design and to error detecting. IRE Trans. Electronic Computers, EC-3, 1954.

[11] : D.J. Costello et S. Lin, Error Control Coding, Second Edition, Prentice Hall, 2004

[12] : F.J. MacWilliams and N.J.A. Sloane. The theory of error- correcting codes, volume 16. North holland mathematical libray, 1977.

[13] : G. Kabatiansky and C. Tavernier. List decoding with Reed- Muller codes of order one. nine International Workshop On Al- gebraic and Combinatorial Coding Theory, Proceedings ACCT- 9, pages 230–236, June 2004.

[14] : G. Kabatiansky and C. Tavernier. List decoding of first order Reed-Muller codes ii. tenth International Workshop on Alge- braic and Combinatorial Coding Theory, Proceedings ACCT-10, September 2006.

[15] : Hankerson, DR et al. Théorie du codage et cryptographie: Les Essentiels. Marcel Dekker, New York, 1991.

[16] : Hocquenghem, A., Codes correcteurs d'erreurs, Chiffres (Paris) 2 (1959), 147–156.

[17] : I. S. Reed. A class of multiple-error-correcting codes and the decoding scheme. IRE Trans. Inform. Th., IT-4, 1954.

[18]: John Gill, PGZ Decoder, Stanford University.

[19] : L.H. Charles LEE, Error-Control Block Codes, Artech House Publishers

[20] : MacWilliams, FJ et Sloane, NJA La théorie de la correction des codes d'erreur. Nord- Mathématique Volume Bibliothèque Hollande 16, Elsevier Science BV, Amsterdam, 1977.

[21]: Massey, J. L., Step-by-step decoding of the Bose–Chaudhuri Hocquenghemcodes,IEEE Trans. Inf. Theory, vol. IT-11, pp. 580–585, October 1965.

[22] : M. Demazure, Cours d'algèbre, Cassini 1997

[23] : Modin, Errror control coding.

[24] : O. Goldreich and L. A. Levin. A hard-core predicate for all onewayfunctions. In STOC '89 : Proceedings of the twenty-first annual ACM symposium on Theory of computing, pages 25–32, New York, NY, USA, 1989. ACM.

[25] : P. Chose, A. Joux, and M. Mitton. Fast correlation attacks : an algorithmic point of view. In Advances in Cryptology - EU- ROCRYPT 2002, volume 2332 of Lecture Notes in Computer Science, pages 209–221. Springer-Verlag, 2002.

[26] : Pierre CsillagEd.Ellipses, Introduction aux codes correcteurs

[27]: SHANNON C. E., « A Mathematical Theory of Communication », The Bell System Technical Journal, vol. 27, pp. 379-423, 623-656, juillet-octobre 1948.

[28] : T. Kasami, N. Tokura, and S. Asumi. On the weight enumeration of weights less than 2.5d of Reed-Muller codes. Information and control, 30(4) :380–395, 1974.

[29] : T. Kasami and N. Tokura. On the weight structure of Reed- Muller codes. IEEE Trans. on Inform. Theory, 16(6) :752–759, 1970.

[30] : Victor K. Wei. Generalized Hamming weights for linear codes. IEEE Transaction on Information Theory, 37 :1412–1418, September 1991.

[31] : William Stallings, Cryptography and Network Security, third edition, prentice Hall, 2003

[32] : www.greyc.ensicaen.fr/gbinetT_info/COURST_info_b_lineaires/.pdf.

[33] : www.irisa.fr / armor / lesmembres / cousin / Enseignement / Reseaux-generalites / Cours / 3.html

[34] : www.sciences.ch/html.fr/info/theorique/info/codecorrecteurs/01.php

[35] : Yi Lu and Serge Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In CRYPTO, pages 407–425, 2004.

Liste des figures

Liste Des Tableaux

Chapitre 3 : Code Reed-Solomon

ABSTRACTION

L e principe des codes correcteurs d'erreurs est de rajouter une information supplémentaire redondante de manière à détecter et éventuellement corriger de possibles erreurs de transmission. La théorie des codes correcteurs ne se limite pas qu'aux communications classiques (satellite, radio, télévision, fibre optique, DSL etc.) mais également aux supports pour le stockage comme les disques compacts, la mémoire RAM et d'autres applications où l'intégrité des données est importante.

La genèse de ces codes consiste à effectuer deux opérations à savoir le codage et le décodage. La première consiste en l'adjonction des bits de redondance, tandis que la deuxième essayera de détecter et corriger les erreurs en passant par plusieurs étapes.

Mots Clés : Théorie de l'information, Codes Correcteurs d'erreurs, Codes en Blocs, Codes Reed Muller.

ABSTRACT

T he principle of error-correcting codes is to add additional redundant information to detect and correct any possible transmission errors. The coding theory is not confined only to traditional communications (satellite, radio, television, fiber optics, DSL etc...) but also for storage media like compact disks, RAM and other applications where Data integrity is important.
The genesis of these codes is to perform two operations: the encoding and decoding. The first is the addition of redundant bits, while the second will attempt to detect and correct errors through several stages.

Keywords: Information theory, error correcting codes, block codes, Reed-Muller Codes..

ملخص

برنامج مصحح الأخطاء يرتكز على إضافة معلومات الى المعلومة الأصلية المراد إرسالها بهدف تعيين مواقع الأخطاء و تصحيحها. مثل هذه البرامج لا يقتصر استعمالها على الاتصالات الكلاسيكية (مذياع, تلفاز...) بل أيضا على أجهزة التخزين مثل أقراص التخزين, الذاكرة رام و غيرها من الأجهزة.

تطبيق هذه البرامج يتضمن القيام بعمليتين أساسيتين الأولى الكوداج أي إضافة معلومات مراقبة في آخر المعلومة الأصلية, و الثانية تسمى بالديكوداج و التي تمكننا من اكتشاف الأخطاء و تصحيحها مرورا بعدة مراحل.

المفاتيح : نظرية المعلومات, مصحح الأخطاء, مفتاح التقسيم, برنامج ربد ميلار.