Recherche des performances dans la mise en oeuvres des codes linéaires cycliques en ASIC à haut debit

Thierry Vallino

Thèse

présentée par

VALLINO Thierry

pour l'obtention du titre de

Docteur de l'Université de Metz
en Micro-électronique

Recherche des performances dans la mise en œuvres des codes linéaires cycliques en ASIC à haut débit.

Soutenue le 17 décembre 1999

Composition du jury :

Rapporteur :

M. CALVEZ Jean Paul :	Professeur, IRESTE-Université de Nantes
M. PIESTRAK Stanislaw :	Professeur, Université de Wroclaw

Directeur de thèse :

M. LEPLEY Bernard :	Professeur, Université de Metz

Examinateurs :

M. BRAUN Francis :	Professeur, Université de Strasbourg
M. DANDACHE Abbas :	Maître de Conférences, Université de Metz
M. DELAHAYE Jean Pierre :	Ingénieur TDF-C2R Metz
M. MONTEIRO Fabrice :	Maître de Conférences, Université de Metz

LICM / CLOES
Laboratoire Interfaces Composants et Microélectronique
Centre Lorrain d'Optique et d'Electronique des Solides
Université de Metz et SUPELEC

Thèse

présentée par

VALLINO Thierry

pour l'obtention du titre de

Docteur de l'Université de Metz
en Micro-électronique

Recherche des performances dans la mise en œuvres des codes linéaires cycliques en ASIC à haut débit.

Soutenue le 17 décembre 1999

Composition du jury :

Rapporteur :

M. CALVEZ Jean Paul :	Professeur, IRESTE-Université de Nantes
M. PIESTRAK Stanislaw :	Professeur, Université de Wroclaw

Directeur de thèse :

M. LEPLEY Bernard :	Professeur, Université de Metz

Examinateurs :

M. BRAUN Francis :	Professeur, Université de Strasbourg
M. DANDACHE Abbas :	Maître de Conférences, Université de Metz
M. DELAHAYE Jean Pierre :	Ingénieur TDF-C2R Metz
M. MONTEIRO Fabrice :	Maître de Conférences, Université de Metz

LICM / CLOES
Laboratoire Interfaces Composants et Microélectronique
Centre Lorrain d'Optique et d'Electronique des Solides
Université de Metz et SUPELEC

REMERCIEMENTS

Je tiens tout d'abord à remercier M. Bernard LEPLEY, directeur du LICM/CLOES pour m'avoir accepté dans son laboratoire et pour les conseils qu'il m'a fournis tout au cours de ce travail.

Je tiens aussi à remercier M. Abbas DANDACHE et M. Fabrice MONTEIRO pour avoir pris sur leur temps pour encadrer cette thèse et pour les conseils qu'ils m'ont apportés.

Je remercie tout particulièrement M. CALVEZ et M. PIESTRAK pour avoir bien voulu être les rapporteurs de mon travail.

Enfin, je voudrais remercier les autres membres du jury, M. BRAUN, M. DELAHAYE, pour avoir bien voulu examiner ce travail.

Je tiens tout particulièrement à remercier les thésard du laboratoire Angélique, Hervé, Jean Philippe, Serge et les membres du CESIUM Gaëlle, Jean-François et Rémy pour m'avoir « supporter » pendant ces années et pour l'ambiance conviviale qu'ils ont créé.

Enfin, je remercie également ma famille et tous mes amis pour leur soutien moral.

Résumé

L'essor des télécommunications est un des faits marquant de cette fin du siècle. Il s'accompagne d'un accroissement des applications dédiées (télévision numérique, Internet). Cependant, les voies de transmissions ne sont pas parfaites et peuvent corrompre les données émises. L'utilisation des codes correcteurs d'erreurs permets alors dans une certaine mesure d'y remédier. Notre travail s'est articulé autour de deux axes.

Le premier axe a consisté à trouver et développer l'architecture d'un codeur/décodeur à logique majoritaire fonctionnant à haut débit et pouvant traiter des flots continus de données. L'étude des différentes implantations possibles de diviseur polynomial (cœur du codeur/décodeur) a permis de montrer que l'architecture parallèle était la mieux adaptée à nos contraintes. Celle-ci a été décrite en VHDL synthétisable et simulée sous le logiciel Altera MaxPlus II. A partir de cette description, l'impact du degré de parallélisme sur la surface et le débit du codeur/décodeur a été étudié sur plusieurs codes à logique majoritaire (DSCC, DTI, EG). Il en ressort que l'augmentation du débit est nettement plus importante que celle de la surface (circuit EPF10K10LC84-3).

Le deuxième axe de recherche a consisté à une étude de faisabilité dans la mise en œuvre d'un code Reed-Solomon avec effacements. La méthode des effacements permets d'accroître la capacité de correction des codes Reed-Solomon d'origine au moyen d'une technique de marquage. Habituellement, l'implantation des codes RS avec effacements est réalisée mais sans mettre en œuvre la technique de marquage. Un tel marquage a été développé et simulé en coopération avec TDF-C2R. La simulation ayant permis de valider cette technique, un codeur/décodeur RS (127,121,7) a été décrit en VHDL synthétisable et simulé sous le logiciel Altera MaxPlus II. Les résultats ont permis de montrer que les performances d'un tel circuit sont comparables à celles des circuits usuels n'implantant pas le marquage.

Mot-clé : logique majoritaire, architecture parallèle, Reed-Solomon, effacements, codes linéaire cycliques

Abstract

Recent progress in telecommunication is the most important fact of this end of century. It is accompanied by a growth of dedicated applications (numerical TV, internet,...). However, media are not perfect and can corrupt the emitted data. The use of error correcting code permits to detect and correct the errors at the receipt.

The first part of this research was to find and develop an architecture of majority logic codec which can be used with high output throughput data stream. The study of the different implementation of the polynomial divider (core of the codec) allowed us to demonstrate the parallel architecture was the better choice compared with our constraints. This one has been described in VHDL and simulated with MaxplusII software of Altera. Using this description, the impact of the parallelism degree on both the area and the output throughput of the codec has been studied for different code (DTI, DSCC, EG).

The second part of this research was to implement a "Reed-Solomon with erasures" codec in ASIC. The use of erasure increases the error capability of the original Reed-Solomon code. An erasure processing (developed in co-operation with TDF) has been validated through the simulation based on several error models. After this validation, a (127,121,7) RS code with erasures has been described in VHDL and simulated with MaxplusII software of Altera in order to cope with the constraints of real time applications.

Keywords : majority logic, parallel architecture, Reed-Solomon, erasures, linear cyclic codes

Sommaire

Liste des figures

Bibliographie

[1] M.S. Hodgart, « Efficient coding and error monitoring for spacecraft digital memory »
Int . J. Electronics, 1992, Vol 73, n°1, pp 1-36

[2] Bruce Schneier,« Applied cryptography, protocols,algorithms and source code in C »
John Wiley & Sons, 1994

[3] Florent Chabaud, « Recherche de performance dans l'algorithmique des corps finis, application à la cryptographie », Thèse de l'école Polytechnique, Paris,1996

[4] M. Nicolaidis, Y.Zorian, « On line testing for VLSI- A compendium of approches », Journal of electronic testing- theory and application, vol 12n n°1/2, February-April 1998, pp 7-20

[5] » A. Paschalis & all, « Efficient design of totally self checking checker-decoders for all cyclic AN codes , IOLTW'99, Rhodes

[6] T.C. Dietterich, G. Bakiri, « Solving multiclass learning problems via error correcting output code », , Journal of artificial intelligence research, 1995, n°2, pp 263, 286

[7] Osamu Yamada & al,1 « Error correction system for a difference set cyclic code in a teletext system », brevet n° 4,675,868 ; juin 1987

[8] Osamu Yamada, « FM Multiplex Broadcasting »,
NHK laboratories Notes, vol 372, 1989

[9] Delahaye J.P, « A difference-set cyclic code(1057,813,34) ».
Annals of Telecommunications 51 n°1-2,1996

[10] T. Vallino, S. Piestrak, A. Dandache, F. Monteiro, B. Lepley, « Study of a new parallel architecture dedicated to the family of the DSCC codes »

5rd IEEE International On-Line Testing Workshop, Rhodes, Greece, July 5-7 1999

[11] A. Dandache, T. Vallino, F. Monteiro, J.P. Delahaye, « Code Reed Solomon (127,k,d) avec effacements : simulation et conception sur FPGA »
Traitement du Signal, Vol 16, n°4, 1999

[12] J.G. Remy, J. Cueugniet, C. Siben, « Système de radiocommunication avec les mobiles »,
Eyrolles, collection C.N.E.T-E.N.S.T, 1988

[13] Shu Lin, D. J. Costello Jr, « Error control coding : Fundamentals and applications »,
Prentice Hall 1983

[14] C. R. Hawthorne & all, « An Error-Correction Scheme for a helical-scan magnetic data storage system »
IEEE journal on selected areas in communications vol 10, n°1, 1992

[15] M. D. Rice and S. B. Wicker , « Modified majority logic decoding of cyclic codes in hybrid – ARQ systems »
IEEE Trans. Commmun, vol. 40, pp. 1413--1417, Sep. 1992

[16] G. Battail, « Théorie de l'information :Application aux techniques de communication »
Edition Masson, 1997

[17] A. Poli, Li. Huguet, « codes correcteurs, théorie et applications »
Collection Masson, 1989

[18] A Donnedu, Cours de mathématiques (volume 1 et 2) , édition Vuibert

[19] Alexandru Spataru, « Théorie de la transmission vol II : codes et décision »
Edition Masson, 1973

[21] G. Cohen, J.L. Dornstetter, P. Godlewski, « Codes correcteurs d'erreurs : une introduction au codage algébrique »,
collection C.N.E.T/ ENST ; Edition Masson 1992

[22] Leonard D. Baumert, « cyclic difference sets »
Lecture notes in Mathematics, vol 182, Springer-Verlag, 1971

[23] J. Singer, « A theorem in finite projective geometry and some applications to number theory », AMS Trans, 43, 1938.

[24] E.J. Weldon Jr, « Difference set cyclic codes », The Bell System Technical Journal, September 1966

[25] T. Kasami, L. Lin, W.W Peterson, « Some results on cyclic codes which are invariant under the affine group and their applications », Inf. control, 2, pp 475-496, 1968.

[26] C. R. P. Hartmann & all, « On the structure of generalized finite geometry code »,
IEEE trans on information theory, IT-20(2) Mars 1974.

[27] Michael Braun & all, « Parallel CRC computation in FPGA », Rapport de laboratoire, Université de Saarlands

[28] Beniamino Castagnolo, Maria Rizzi, « High Speed Error correction circuit based on iterative cells », Int. J. Electronics. 1993, Vol 14 n°4, pp 529-540

[29] Kobayashi & all, « New LSI Architecture for ultra High Speed Error correction an it's application to the majority decoding LSI for (1057,83) code », Nhk laboratories Notes, 1992

[30]. Kobayashi & all, « 50 Mhz CMOS pipelined majority logic decoder », IEICE trans. Fundamentals, vol E79-a, n°7, juillet 1993

[31] K.E. Batcher, « Sorting Networks and their applications », proc 1968, SJCC AFIPS vol 32, 1968, pp 307-314

[32] Stanislaw J. Piestrak, « The minimal test set for multioutput threshold circuits implemented as sorting networks », IEEE trans on computers vol 42 n°6 juin 1993

[33] Tong Bi Pey and Charles Zukowski, « High Speed Parallel CRC », IEEE trans on communication vol 40 n°4, 1992

[34] « CRC MegaCore function » Altera Datasheet, ver 2A-DS-CRC-02, April 1999

[35] R. Blahut, « Transform techniques for error control code », IBM J of Research and Development, 23, 1979, pp 299-315

[36] J.L. Dornstetter ,« On the equivalence between Berlekamp's and euclid's theorem » IEEE Trans on Inf Theory, vol IT-33 May 1987, pp428-431

[37] « Reed-Solomon decoder with erasures », Société Hammer Core (mars 1999)

[38] « AHA 4011. 10 Mbytes/sec Reed-Solomon Error correction device », Product Specification, Advanced Hardware Architectures

[39] K. Matsushima & all, « Parallel encoder and decoder architecture for cyclic codes », IEICE trans on fundamentals, vol E79-A, n°9, sept 1996

[40] C. Paar, « A New architecture for parallel finite field multiplier with low complexity based on composite field », IEEE trans on computers July 1996, vol 45, n°7, pp 1313-1323

Introduction

Ces dernières années ont vu l'émergence de réseaux de communications aussi divers que la téléphonie mobile, internet, intranet. Elle s'est accompagnée d'un accroissement des applications dédiées (visioconférence par exemple). Cependant ces diverses applications n'utilisent plus un seul support de transmission dédié, mais tout un panel de médias (voie hertzienne, câble, satellite, etc). Aussi, différents protocoles sont ou ont été à l'étude pour faciliter la transmission à travers ces supports (recommandation DAVIC, norme MNSC). Cependant ceux-ci doivent utiliser des fonctions de traitement des données de plus en plus complexes. Parallèlement, le volume des données à transmettre s'accroît. Par conséquent le risque d'erreur augmente. Il est alors nécessaire d'assurer la sûreté des données de plus en plus rapidement. Pour cela diverses techniques peuvent être utilisées :

- *implantation software* : le problème majeur consiste à implanter des algorithmes efficaces en temps de cycles de calculs (par exemple l'utilisation de masques est souvent nécessaire pour implanter un XOR entre deux bits sur un DSP). Cependant, cette technique est très souple et permet de reconfigurer « rapidement » son système.

- *implantation hardware* : elle permet de réaliser et d'implanter des fonctions complexes fonctionnant à des débits relativement élevés. En contrepartie cette méthode

n'offre pas la souplesse d'une architecture software (cependant, il existe des FPGA pouvant se reprogrammer in situ).

Une des fonctions complexes (telles que la compression de données, le modulateur, le démodulateur, le codage de canal, etc) à implanter pour la transmission de donnée est d'assurer la sûreté des messages envoyés. En effet, les médias utilisés pour transporter les données ne sont pas parfaits et sont le siège de perturbations pouvant corrompre les messages transmis. L'impact des erreurs va alors être aggravé par l'augmentation du débit et de la quantité de données. Pour faire une comparaison, une erreur de un bit a un impact plus important dans un fichier compressé que dans un fichier non compressé.

Il va alors être nécessaire de contrôler les erreurs afin de permettre une restitution aussi fidèle que possible du message émis et à un débit de plus en plus important. Plusieurs méthodes ont été développées afin de détecter/corriger les erreurs [13] :

- *Transmission avec requête* (ARQ) : On rajoute des symboles de parité pour effectuer une détection d'erreur. Si une erreur est détectée, on demande une retransmission,

- *Transmission utilisant des codes correcteurs d'erreurs* : cette fois ci la redondance est calculée selon une loi qui permet la détection et la correction des erreurs,

- *Transmission hybride* : cette méthode utilise conjointement le système ARQ et les codes correcteurs d'erreurs

Les méthodes ARQ et ARQ-hybride peuvent être utilisées respectivement pour des applications à faible et moyen débit à cause de la retransmission [13]. Par contre, les codes correcteurs d'erreurs peuvent être utilisés pour des applications à haut débit [13]. Le dialogue entre la source et le destinataire n'existe plus, ce qui est nécessaire pour certaines applications (vérification des mémoires dans une navette spatiale par exemple [1]) [Il est à noter que les codes correcteurs ne sont pas limités à cet usage et sont aussi

utilisés dans d'autres domaines tels que la cryptographie [2], [3], le test en ligne [4],[5] ou l'intelligence artificielle [6]

De ces trois méthodes, la plus adaptée aux hauts débits consiste donc en l'utilisation des codes correcteurs. Les premières implantations des codes correcteurs ont été logicielle et arrivent à la limite de leur performance. Il est donc nécessaire de développer et d'implanter des architectures matérielles spécifiques aux codes correcteurs et dédiées à des applications à haut débit. Plusieurs architectures peuvent être mises en concurrence :

- *L'architecture série* : première architecture utilisée pour les codes linéaires cycliques, son avantage est qu'elle est simple à implanter (XOR et flip-flop uniquement) mais lente et ne permet pas de traiter des données en continu [17].

- *L'architecture systolique* : Les opérations se font toujours en série, mais en pipeline. Lorsque le débit augmente, cette architecture n'est pas capable de traiter un flot continu de données [17].

- *L'architecture parallèle* : C'est une généralisation de l'architecture série. Elle permet des débits plus importants que les deux précédentes et peut traiter un flot continu de données. On doit alors tenir compte d'une nouvelle contrainte : La surface du codeur/décodeur parallèle [10].

Notre travail porte essentiellement sur l'étude des performances des codes correcteurs d'erreurs notamment, les codes linéaires cycliques.

Le chapitre 2 présente l'étude de l'impact du parallélisme sur la surface pour un code particulier : le code DSCC (Difference Set Cyclic Code). Ce code est un code linéaire cyclique utilisé par la télévision japonaise NHK pour sécuriser le transport de données [7], [8]. Ce code a aussi été étudié à TDF-C2R [9] et intégré dans un simulateur de code correcteur de la même société. Les résultats de l'étude ayant été concluants [10], elle a

été étendue à la famille des codes cycliques à décodage à logique majoritaire à laquelle le code DSCC appartient.

Le troisième chapitre porte sur un autre problème des codes correcteurs qui consiste à réduire le coût de la transmission. Effectivement, plus le nombre d'erreurs s'accroît, plus le nombre de symboles de redondance à utiliser augmente, ce qui a pour conséquence une utilisation plus importante de la bande passante. Cependant, certains codes (Reed-Solomon) permettent, grâce à l'utilisation de nouveaux algorithmes (code de Reed-Solomon avec effacement), une augmentation de la capacité de correction (i.e. le nombre maximum d'erreurs pouvant être corrigé) sans accroître le coût de la transmission. Par contre, cette augmentation peut se faire au détriment de la surface du codeur utilisé et du débit. Dans la plupart des applications industrielles [37] [38] implantant le code de Reed-Solomon avec effacements, la détection des effacements (qui permet d'augmenter la capacité de correction) n'est pas réalisée. Le Reed-Solomon est (comme les codes à logiques majoritaires) un code linéaire cyclique. Mais, alors que les codes à logiques majoritaires sont des codes binaires (l'information est soit un 1 soit un 0), les codes de Reed-Solomon travaillent sur des symboles de plusieurs bits. Cet emploi de symboles au lieu d'un simple bit entraîne un système de correction complexe. L'étude que nous avons effectuée ne concerne alors pas l'amélioration de l'architecture du système de correction (amélioration de l'architecture) mais uniquement l'étude de l'implantation sur FPGA de la détection des effacements pour un code de Reed-Solomon. Une technique de marquage permettant d'implanter les effacements (méthode qui accroît la capacité de correction sans trop augmenter la redondance) a été développée en collaboration avec TDF-C2R, simulée puis synthétisée sur FPGA afin de valider le concept.

Chapitre I. Les codes correcteurs

Ce premier chapitre est consacré à une présentation générale des codes correcteurs. La partie I.1. pose le problème des transmissions et du stockage des données et présente différentes solutions possibles avec leurs avantages et inconvénients. Les parties I.2. et I.3. sont une description générale de deux familles de codes correcteurs d'erreurs : les codes linéaires et les codes linéaires cycliques.

I.1. Généralités

I.1.1. Position du problème

Dans la société actuelle, que ce soit dans la vie quotidienne ou professionnelle, l'information peut être considérée comme une richesse à deux points de vue :

- Au niveau d'une application, c'est l'information qui constitue réellement la richesse du système. Par exemple lors du transfert de données entre deux banques, c'est l'information qui est primordiale et non son transport (même si celui ci est important).
- A un deuxième niveau, les infrastructures et services offerts pour transporter cette information constitue eux aussi une richesse (boom des téléphones portables par exemple, télévision « pay per view »).

Cependant, d'un point de vue très général, toutes ces applications ont le même but : mettre en relation (conversation, échange de données) deux points quelconques

(personnes, application, matériel) via des supports de transmission très divers (voies hertzienne, câble, fibres optiques). L'augmentation des transferts d'informations (prises au sens large) peut alors être limitée par les caractéristiques physiques des supports de transmission. Par exemple dans le cas de la télévision « pay per view », on doit transmettre N programmes là où avant on ne transmettait qu'un seul programme (limitation du spectre en fréquence pour une application). Il sera alors nécessaire d'avoir un débit plus important pour transmettre les données.

Des techniques particulières ont été mises au point pour utiliser au mieux le spectre en fréquence disponible. On peut citer par exemple le multiplexage en fréquence ou le multiplexage temporel. Ces deux techniques se focalisent sur l'encombrement spectral et l'utilisation optimum des fréquences allouées à une application.

Parallèlement, on peut diminuer le volume des données à transmettre (optimisation du coût de transmission) grâce à l'emploi d'un codage adéquat (*codage de source*) [16].

Type de bruit	Remarques diverses
Galactique	Bruit prépondérant pour une fréquence comprise entre 20 MHz et 200MHz Négligeable au-delà de 300 MHz
Atmosphérique	Prépondérant jusqu'à 20 MHz Essentiellement dû aux éclairs lors des orages
Industriel	Lignes de transports d'énergies Dispositifs de chauffage à induction,...
Urbain	Enseignes lumineuses Dispositif d'allumage des véhicules
Brouillage	Diaphonie Intermodulation entre émetteurs

Tableau 1. Types de bruits pouvant intervenir dans une transmission.

Toutefois, la transmission entre un émetteur A et un récepteur B peut être considérée comme imparfaite (le tableau 1 donne un aperçu des différents types de bruit pouvant intervenir pendant une transmission de données [12]). Comme le débit des données à

transmettre augmente, le risque d'erreur devient de plus en plus important. Par exemple, dans le cas d'une application pour laquelle un codage de source est employé, une « petite » erreur pendant la transmission peut transformer une partie importante du message originel (ce qui pour un transfert entre banques est catastrophique). Le récepteur B doit donc posséder un moyen de vérifier le message reçu, c'est à dire pouvoir détecter/corriger les éventuelles erreurs entachant ce message. La méthode employée dépend alors de l'application visée et du type de bruit perturbant la transmission.

I.1.2. Solutions possibles pour la détection et la correction des erreurs

Une des techniques parmi tant d'autres pour minimiser l'impact des erreurs dues au canal de transmission consiste à rajouter de la redondance (symboles quelconques) au message à transmettre. Cependant si on minimise l'effet des erreurs éventuelles, le récepteur ne peut toujours pas les détecter. Aussi au lieu de rajouter des symboles quelconques, rajoute-t-on des symboles qui dépendent de l'information suivant une loi connue à la fois de l'émetteur et du récepteur. Celle ci dépend alors de l'application choisie et du type de bruit présent sur le canal de transmission. Suivant la loi utilisée, différentes stratégies sont possibles.

I.1.2.1.Systèmes avec ARQ (Automatic Repeat reQuest).

Les systèmes avec ARQ [13] utilise une loi ajoutant des symboles de parité. Cette loi permet de détecter la présence d'erreurs mais le récepteur ne peut ni localiser ni corriger les éventuelles erreurs. Dans le cas d'un message considéré comme erroné, le récepteur émet une requête demandant la réémission de ce message. On parle alors de codes détecteurs d'erreurs avec ARQ. Trois implantations de base du système ARQ existent :

- « Stop and Wait ARQ »
- « Go back N ARQ »

- « Selective repeat ARQ »

Dans le cas du système « Stop and Wait ARQ », l'émetteur attend que le récepteur lui envoie un signal pour continuer l'émission des données. Le récepteur ne peut envoyer que deux valeurs ACK ou NAK. La réception du signal ACK indique que la transmission s'est bien déroulée et l'émetteur peut envoyer le message suivant. Dans le cas contraire (NAK), des erreurs ont perturbé la transmission et l'émetteur doit de nouveau envoyer la donnée précédente. La figure I-1 montre un exemple de déroulement d'une session « Stop and Wait ARQ »

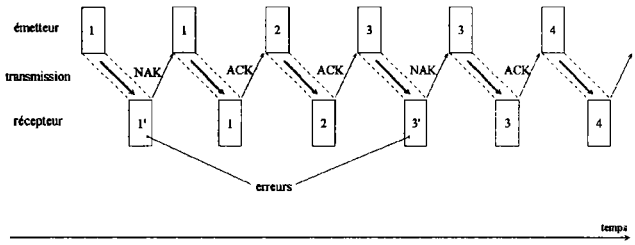


Fig I-1. Exemple de fonctionnement d'un système « Stop and Wait ARQ ».

Dans le cas du système « Go back N ARQ », l'émetteur envoie de façon continue les messages sans s'occuper du récepteur. Le premier signal d'accusé de réception arrive après N cycles de transmission (N dépendant de l'application). Si cet accusé indique qu'il n'y a aucune erreur, l'émetteur continue sa transmission. Dans le cas contraire, l'émetteur émet le mot erroné et les N mots suivants (voir figure I-2). Ce système est plus intéressant que le système « Stop and Wait ARQ » mais n'est pas valable pour des débits importants à cause du nombre important de messages non erronés qui sont retransmis dans le cas d'une erreur sur un seul message.

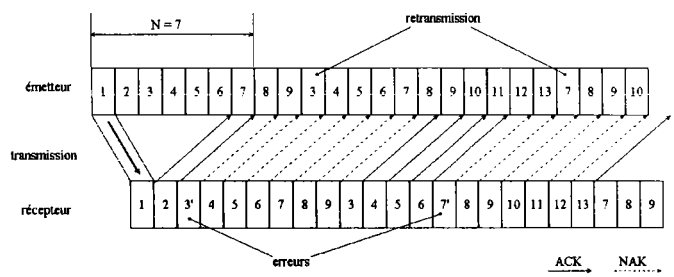


Fig I-2. Exemple d'un système Go back 7 ARQ

Pour le système « Selective repeat ARQ », la procédure utilisée est similaire à celle du système « Go back N ARQ » mais il n'y a retransmission que du message erroné (figure I-3). Cela implique l'utilisation d'une mémoire au niveau du récepteur afin de remettre en ordre les données reçues.

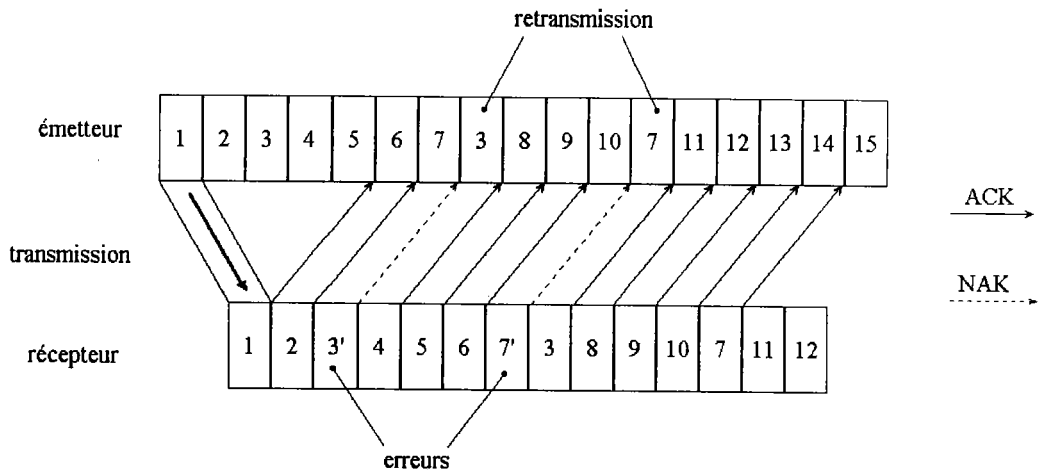


Fig I-3. Exemple de fonctionnement d'un système « selective repeat ARQ »

Les systèmes de décodage utilisés par ces techniques sont simples à implanter. De plus, ces systèmes sont adaptatifs dans le sens où il n'y a retransmission que dans le cas où une erreur se produit. Cependant, ils présentent deux inconvénients majeurs qui interdisent l'utilisation de l'ARQ pour des applications ayant des flots continus de données :

- Le débit utile décroît rapidement quand le nombre d'erreurs augmente.
- Il est nécessaire d'utiliser un canal bidirectionnel ou deux canaux de transmission pour assurer le dialogue émetteur/récepteur, chose qui n'est pas possible dans toutes les applications [14], [15].

La méthode décrite dans le paragraphe suivant permet de s'affranchir d'un dialogue entre l'émetteur et le récepteur et est utilisée pour des systèmes de communication à une voie de transmission. Ce sont les codes correcteurs d'erreurs (Forward Error Correction).

I.1.2.2.Codes correcteurs d'erreurs

Pour les codes correcteurs d'erreurs, la redondance suit une loi mathématique qui permet à la fois la détection, la localisation et la correction des erreurs (codage de canal). Le décodage demande alors plus d'opérations (localisation et correction) que dans les cas précédents. L'implantation du décodeur sera alors plus ou moins difficile selon le code utilisé (la figure I-4 donne le schéma général d'une transmission avec un code correcteur).

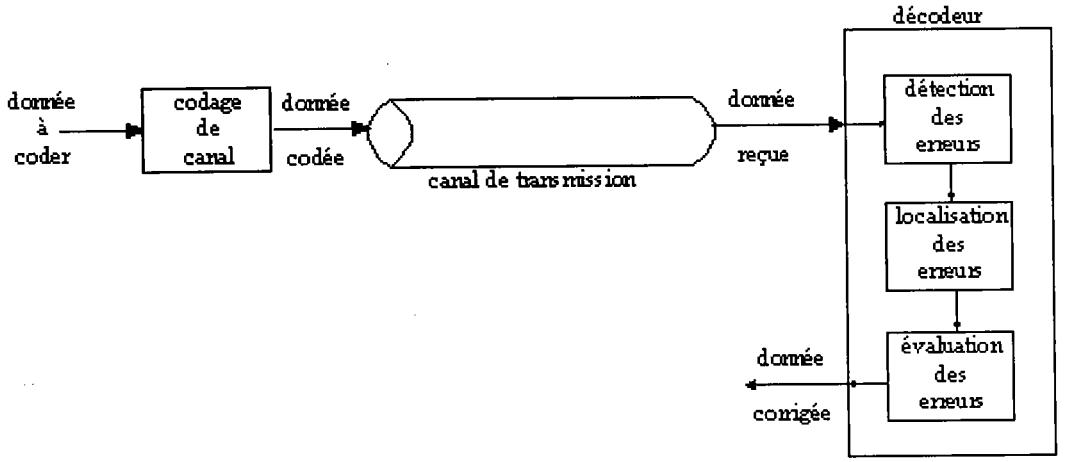


Fig I-4. Exemple d'utilisation d'un code correcteur d'erreur

Comme il ne peut y avoir de retransmission des données erronées, le code correcteur utilisé pour une application doit pouvoir détecter et corriger les erreurs les plus vraisemblables statistiquement (taux de couvertures). Il est alors nécessaire de connaître le type d'erreurs (isolée ou en paquet) et leur fréquence d'apparition. Plusieurs modèles de canaux de transmissions (modèles de Pareto, de Neymann-Pearson, de Markov [16]) ont été développés. Ils sont utiles pour faire des simulations nécessaires au choix d'un code approprié à une application [projet européen RACE]. Par exemple, le modèle de Pareto a été utilisé pour modéliser l'intervalle entre deux erreurs consécutives et des tests de comparaison ont été effectués en RFA par IBM et les PTT. Ces tests ont permis de montrer que ce modèle est valable pour des intervalles courts [17]

L'avantage des codes correcteurs est qu'ils permettent un débit utile constant (égal à k/n avec k et n respectivement la taille de la donnée avant codage et la taille après codage). Par contre, pour corriger un nombre important d'erreur, il est nécessaire d'utiliser des codes dont le paramètre *n* est grand avec des conséquences sur :

- le temps de traitement,
- le débit (pour un k donné, le débit utile chute avec n),
- la surface utilisée pour implanter le décodeur.

De plus si le nombre d'erreurs dans le mot reçu est supérieur à la capacité de correction du code utilisé, le mot en sortie du décodeur est mal décodé. La fausse correction devient alors une source d'erreurs.

Ce dernier inconvénient peut être supprimé si on utilise conjointement un code correcteur d'erreurs et un système ARQ. On parle alors « d'ARQ hybride ».

I.1.2.3. Systèmes hybrides

Deux types de système hybride existent. Dans ce paragraphe, nous nous intéresserons au système hybride de type I uniquement. Pour les systèmes de type II, on peut se référer à [13, p 479-497].

Le système hybride de type I consiste à utiliser un code correcteur d'erreur dans un système avec ARQ (qui peut être l'un de ceux du paragraphe A.I.2.1). L'emploi du code correcteur d'erreur permet la détection et la correction d'un certain nombre d'erreurs et diminue la fréquence de retransmission des messages. En effet, si le message reçu peut être corrigé par le code correcteur, il n'y a pas besoin de demander une retransmission. Certains codes correcteurs ont déjà été modifiés pour pouvoir être inclus dans un tel système [15].

Pour des canaux avec un taux d'erreur faible, le système ARQ est plus efficace que le système hybride à cause de son système adaptatif. Par contre, quand le nombre d'erreurs croît, le système hybride devient plus avantageux car il diminue le nombre de retransmission à effectuer [13]. Cependant, ce système ne peut toujours pas être utilisé pour des systèmes à une voie de transmission.

I.1.3. Conclusions

Le tableau 2 résume les avantages et les inconvénients de chaque méthode.

Méthode	Avantages	Inconvénients
ARQ	Simple à implanter	Débit faible Dialogue émetteur/récepteur
FEC (forward error correction)	Débit constant Pas de dialogue	Le mot décodé peut être faux Plus lourd à implanter
ARQ hybride	Moins de retransmission FEC moins puissant	Nécessite un dialogue

Tableau 2. Résumé des différentes méthodes de corrections des erreurs

Dans toute cette étude, nous ne nous intéresserons qu'à la deuxième stratégie, celle des codes correcteurs, c'est à dire que nous nous placerons dans le cas d'applications pour lesquelles le dialogue n'est pas autorisé et le débit de sortie doit être constant.

Essentiellement, on peut décomposer les codes correcteurs en deux sous familles principales : les codes convolutifs et les codes linéaires. Ces codes sont utilisés couramment dans l'industrie. Dans le cadre de cette étude, nous nous limiterons à l'étude des codes linéaires et plus particulièrement des codes linéaires cycliques. Pour les codes convolutifs et multivariables, on pourra consulter respectivement les ouvrages [13, pp 257-446] et [18, pp201-317].

I.2. Les codes linéaires.

I.2.1. Rappels mathématiques

Dans ce paragraphe, nous allons rappeler quelques définitions de base utiles pour l'étude des codes linéaires et cycliques. Pour de plus amples informations on pourra étudier [19, tome 1 et 2] ou tout autre livre de mathématiques sur les structures fondamentales

I.2.1.1. Structure de groupe

Un ensemble C muni d'une loi interne × (C, ×) possède une structure de groupe si :

- la loi × est associative : $\forall a,b,c \in C$, $a\times(b\times c) = (a\times b)\times c$.
- la loi interne possède un élément neutre **e** : $\forall a \in C$, $a\times e = e\times a = a$.
- tout élément de C admet un inverse a^{-1} : $a\times a^{-1} = a^{-1}\times a = \mathbf{e}$.

Lorsque la loi est commutative, on dit que C est un groupe commutatif. (On note alors sa loi additivement).

I.2.1.2. Structure de corps

Soit E un ensemble muni de deux lois de compositions interne (+ et x). On dit que E est un corps si

- (E,+) est un groupe commutatif,
- × est une loi associative,
- × est distributive pour l'addition +,
- × possède un élément neutre,
- Tout élément non nul est inversible.

I.2.1.3. Structure d'espace vectoriel

Soit E un ensemble quelconque et K un corps commutatif. On suppose que E est muni d'une loi de composition interne + et d'une loi de composition externe × sur K, c'est a dire :

$E \times E \rightarrow E$:	$K \times E \rightarrow E$:
$(a,b) \# a+b$	$(\alpha,a) \# \alpha \times a$

On a alors la définition suivante :

E est un espace vectoriel sur K si

- (E,+) est un groupe commutatif
- $\forall a,b \in E, \forall \alpha \in K \quad \alpha \times (a+b) = \alpha \times a + \alpha \times b$
- $\forall a \in E, \forall \alpha,\beta \in K \quad (\alpha+\beta) \times a = \alpha \times a + \beta \times a$
- $\forall a \in E, \forall \alpha,\beta \in K \quad \alpha \times (\beta \times a) = (\alpha \times \beta) \times a$
- $\forall a, \in E, \quad 1 \times a = a \times 1$

(Par exemple, pour tout corps commutatif K, l'ensemble K[x] des polynômes à une indéterminée x sur le corps K est un espace vectoriel).

De plus, si l'espace vectoriel est de dimension finie, il est possible de calculer une base de cet espace (c'est à dire un ensemble d'éléments linéairement indépendants). Tout élément de l'espace vectoriel peut alors s'écrire comme une combinaison linéaire des éléments de la base utilisée.

L'utilisation de cette structure et l'existence d'une base va permettre de définir un certain type de codes en blocs : les codes linéaires.

I.2.2. Les codes linéaires

I.2.2.1. Codes en blocs

D'après le paradigme de Shannon, tout système de communication peut être schématisé de la façon suivante [16] :

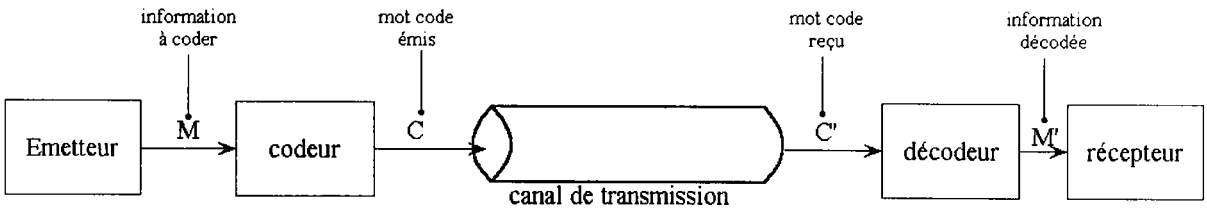


Fig I-5. Représentation générale du système de communication considéré

D'après ce qui a été écrit dans le chapitre précédent, l'émetteur rajoute des symboles à l'information à transmettre (redondance) suivant une loi ψ connue du récepteur. Le destinataire du message peut alors vérifier si la loi ψ est respectée. Si cette loi n'est pas respectée, le message reçu contient une ou plusieurs erreurs (détection des erreurs) et le récepteur va tenter de les localiser et les corriger (décodage). Dans le cas contraire, aucune erreur n'a été détectée (il y a toujours la possibilité que les erreurs aient transformé un mot de code en un autre mot de code).

De plus, il est nécessaire que la loi ψ fasse correspondre à une information un seul mot code. ψ doit donc être une bijection entre l'espace des mots d'informations et l'espace des mots codes.

Dans le cas particulier du codage binaire par bloc (les symboles appartiennent à l'ensemble $\{0,1\}$), les bits d'informations sont regroupés en blocs de taille fixe k (il y donc 2^k messages distincts possibles). Le codeur transforme (loi ψ) alors ces blocs en mot code constitués de n bits. Par conséquent, il y a 2^k mots de code distincts qui correspondent aux 2^k mots d'information (ψ est une bijection).

A la réception, en observant le mot, le destinataire peut décider s'il y a des erreurs ou non. Si le message n'appartient pas à l'ensemble des mots de codes il y a une erreur. Dans le cas contraire on considère qu'il n'y a pas d'erreurs de transmission (les éventuelles erreurs ne sont pas couvertes par le code utilisé). Pour déterminer si le mot

reçu est un mot de code, le récepteur doit utiliser un dictionnaire (mémoire de 2^k*n bits). Pour des valeurs de k et n assez grandes, cette manière de faire devient coûteuse.

La linéarité (i.e. ψ est une loi linéaire) devient alors une structure souhaitable pour un code en bloc car elle permet de réduire la complexité du codage et du décodage.

I.2.2.2. Code linéaire, Matrice génératrice, Matrice de contrôle

On définit un code linéaire C(n,k) par [17] :

> Soit $V = \{(v_1,v_2,..v_n)/\ v_i \in \{0,1\}\}$. Un code en bloc de longueur *n* constitué de 2^k mot code distincts C(n,k) est dit linéaire si et seulement si les 2^k mot de codes forment un sous espace vectoriel de V de dimension k

D'après le paragraphe I.2.1.3., il existe une base dans cet espace. Comme la dimension de l'espace vectoriel est k, il existe k vecteurs indépendants. On peut donc utiliser ces vecteurs pour former une base de l'espace vectoriel. On va noter g_i le $i^{ème}$ vecteur de la base.

D'après la définition d'une base d'un espace vectoriel de dimension finie, tout élément de C peut s'écrire comme une combinaison linéaire des vecteurs de la base considérée. Par conséquent, l'ensemble C peut s'écrire :

$$C = \{u \in V / u = a.G,\ a = (a_1,a_2,\ldots,a_k) \in \{0,1\}^k\} \quad (1.1)$$

où G est une matrice k×n dont les lignes sont les vecteurs g_i de la base de C. La matrice G est alors appelée *matrice génératrice de C*

Le circuit de codage peut être constitué d'un circuit mémorisant les k lignes de la matrice G. le codage consiste alors à additionner (modulo 2) les différentes lignes de la base selon le mot « u » à coder. La figure I-6 (b) est un exemple d'implantation d'un codeur dont la matrice génératrice est représentée à la figure I-6 (a).

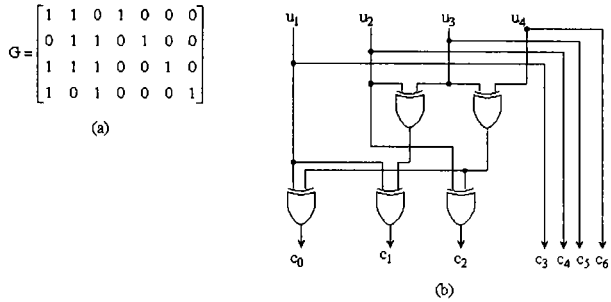


Fig I-6. Exemple d'implantation de codeur

Considérons maintenant le sous espace vectoriel orthogonal à C, c'est à dire

$$C^{\perp} = \{v \in V / <u,v> = 0 \ \forall \ u \in C\} \qquad (1.2)$$

Il est clair que cet espace est de dimension (n-k) [18 chapître 2]. On peut alors le considérer comme un code linéaire $C^{\perp}(n,n-k)$. Il s'ensuit qu'il possède une matrice génératrice (n-k)×n que l'on peut noter H. On peut alors écrire la relation suivante :

$$C = \{u \in V / u.H^t = 0\} \qquad (1.3)$$

La matrice H est appelée *matrice de contrôle* de C.

En utilisant la propriété précédente, le récepteur peut alors détecter la présence ou l'absence d'erreurs dans le mot reçu. Cette propriété peut aussi être utilisée pour corriger les erreurs. C'est ce que l'on va démontrer dans le paragraphe suivant.

I.2.2.3. Décodage des codes linéaires

I.2.2.3.1. Décodage par tableau standard

Soit un code linéaire C(n,k). Il y a 2^k mots de code possibles $\{c_1, c_2, \ldots, c_2{}^k\}$. Lors de la transmission sur un canal bruité, le mot reçu appartient à l'ensemble $\{0,1\}^n$. Il y a donc 2^n mots possibles. Lors du décodage, le récepteur doit partitionner ces 2^n mots possibles en 2^k ensembles disjoints E_i. Chaque ensemble E_i contient un seul mot de code C_i et tous mots devant être décodés comme Ci (c'est à dire en tenant compte de des erreurs). Connaissant l'ensemble dans lequel se trouve le vecteur reçu, celui ci peut être facilement décodé (figure I-7)

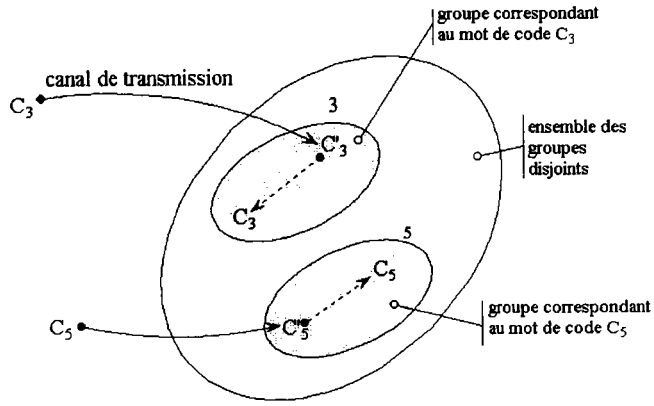


Fig I-7. Principe de décodage des codes linéaires

On peut utiliser la propriété de linéarité des codes pour réaliser une telle partition. On considère l'ensemble des 2^k mots de codes que l'on range dans la première ligne d'un tableau (le vecteur nul tête). Des 2^n-2^k mots restants, on choisit un vecteur e_1. La seconde ligne est constituée en additionnant chaque mot de code de la première ligne avec le vecteur e_1. La troisième ligne est réalisée de la même façon mais avec un vecteur e différent. Le processus continue jusqu'à ce que les 2^n vecteurs aient rempli le tableau. Ce tableau est appelé *tableau standard* (figure I-8). Ce tableau possède plusieurs propriétés :

- Chaque vecteur apparaît une seule fois et dans une seule ligne.
- La somme de deux vecteurs dans une ligne est un mot de code.

Par construction, le tableau standard est composé de 2^k colonnes disjointes. Soit E_j la $j^{ème}$ colonne du tableau standard. E_j peut aussi s'écrire :

$$E_j = \{c_j, c_j+e_1, c_j+e_2 \ldots c_j+ e_2^{n-k}{}_{-1}\} \qquad (1.4)$$

Les colonnes E_j peuvent alors être utilisées dans la méthode de décodage décrite précédemment (d'après la construction du tableau standard, il faut rechercher la position du vecteur reçu C' pour connaître le mot code qui se trouve dans la même colonne).

0	c_1	c_2	...	c_j	...	c_M
e_1	e_1+c_1	e_1+c_2	...	e_1+c_j	...	e_1+c_M
⋮	⋮	⋮	⋮		⋮	⋮
e_i	e_i+c_1	e_i+c_2	...	e_i+c_j	...	e_i+c_M
⋮	⋮	⋮	⋮		⋮	⋮
e_P	e_P+c_1	e_P+c_2	...	e_P+c_j	...	e_P+c_M

Fig I-8. Tableau standard pour un code linéaire C(n,k) [$p = 2^{n-k}-1$, $M = 2^k-1$]

Par exemple, on considère le code généré par la matrice G de la figure I-9a. Ce code transforme un mot de longueur 4 en un mot code de longueur 7. Il y a alors 16 mots de code possible (figI-9b).

En appliquant la méthode précédente, le tableau standard est un tableau de 8 lignes et de 16 colonnes, soit un total de 128 mots de 7 bits (fig I-10).

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

(a)

mot à coder	mot code
0000	0000000
0001	1010001
0010	1110010
0011	0100011
0100	0110100
0101	1100101
0110	1000110
0111	0010111
1000	1101000
1001	0111001
1010	0011010
1011	1001011
1100	1011100
1101	0001101
1110	0101110
1111	1111111

(b)

Fig I-9. Caractéristiques du code (7,4)

Par exemple, supposons que le mot envoyé et le mot reçu soient respectivement les suites « 1100101 » et « 1101101 ». En parcourant le tableau, on retrouve le mot reçu à la position (6,5) (i.e. $6^{ème}$ colonne, $5^{ème}$ ligne). Le mot en tête de la ligne donne l'erreur affectant le mot reçu (ici la suite «0001000 »). Par contre le mot en tête de la colonne indique par quel mot-code le mot reçu doit être décodé (« 1100101 »).

mots de code

erreur															
0000000	1010001	1110010	0100011	0110100	1100101	1000110	0010111	1101000	0111001	0011010	1001011	1011100	0001101	0101110	1111111
1000000	0010001	0110010	1100011	1110100	0100101	0000110	1010111	0101000	1111001	1011010	0001011	0011100	1001101	1101110	0111111
0100000	1110001	1010010	0000011	0010100	1000101	1100110	0110111	1001000	0011001	0111010	1101011	1111100	0101101	0001110	1011111
0010000	1000001	1100010	0110011	0100100	1110101	1010110	0000111	1111000	0101001	0001010	1011011	1001100	0011101	0111110	1101111
0001000	1011001	1111010	0101011	0111100	1101101	1001110	0011111	1100000	0111001	0010010	1000011	1010100	0000101	0100110	1110111
0000100	1010101	1110110	0100111	0110000	1100001	1000010	0010011	1101100	0111101	0011110	1001111	1011000	0001001	0101010	1111011
0000010	1010011	1110000	0100001	0110110	1100111	1000100	0010101	1101010	0111011	0011000	1001001	1011110	0001111	0101100	1111101
0000001	1010000	1110011	0100010	0110101	1100100	1000111	0010110	1101001	0111000	0011011	1001010	1011101	0001100	0101111	1111110

FigI-10. Tableau Standard utilisé pour le décodage du code (7,4)

Pour des codes linéaires avec des paramètres grands, le problème devient difficile à résoudre tant au niveau du temps de calcul que de la mémoire qui doit être utilisée pour stocker le tableau. On utilise alors le plus souvent le décodage par syndrôme.

I.2.2.3.2. Décodage par syndrôme

Considérons une ligne quelconque L_i égale à l'ensemble $\{e_i, e_i+c_1,\ldots,e_i+c_M\}$. On peut alors remarquer :

$$\forall\, u \in \{1,..,M\}\ (e_i+c_u)H^t = e_iH^t \text{ (car } c_u \text{ est un mot de code)} \qquad (1.5)$$

$$\forall\, i,j \in \{1,..,M\}^2\ e_iH^t \neq e_jH^t \qquad (1.6)$$

Le résultat e_iH^t est appelé *syndrôme.*

D'après la formule 1.5, tous les vecteurs d'une même ligne ont le même syndrôme. La formule 1.6 indique qu'il y a correspondance une à une entre le syndrôme et le premier terme des lignes du tableau standard. En utilisant, ces deux propriétés, il est possible de calculer un nouveau tableau, plus simple que le tableau standard, et nous permettant de décoder le vecteur reçu. Si on reprend l'exemple précédent, le tableau utilisé pour le décodage est celui de la figure I-11 a).

L'algorithme de décodage nécessite alors trois opérations (figure I-11 b) :

- Le calcul du syndrôme S à partir du vecteur reçu C'.
- Le calcul de e_i à partir du syndrome.
- Le calcul du mot de code initial C : $C = C' \oplus e_i$

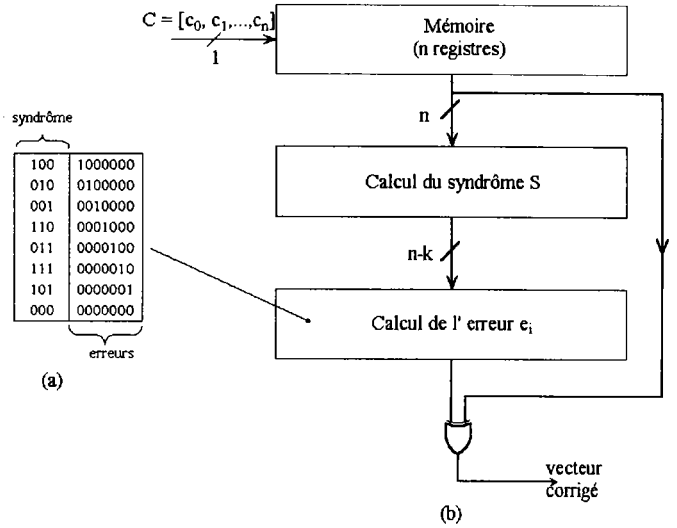


Fig I-11. Implantation générale d'un décodeur pour un code linéaire

I.3. Les codes cycliques

I.3.1. Définition

Soit $C = [c_0,c_1,\ldots,c_{n-1}]$ un mot de code quelconque. En effectuant une permutation circulaire de i rangs vers la droite, on obtient un nouveau vecteur $V=[c_{n-i},..,c_{n-1},c_0,..c_{n-i-1}]$

Deux cas peuvent se produire :

- le vecteur V appartient à $C^{\perp}$ le code dual de C,
- le vecteur V appartient à C.

Il existe des codes pour lequel la deuxième possibilité est toujours vérifiée : les codes cycliques. D'où la définition

> Un code linéaire C(n,k) est dit cyclique si toute permutation d'un mot de code donne un autre mot de code.

I.3.2. Polynôme générateur.

Il est commode pour les codes cycliques d'utiliser une représentation polynomiale, c'est à dire qu'à tout vecteur $C = [c_0,c_1,\ldots,c_{n-1}]$ on va associer le polynôme $\sum_{i=0}^{i=n-1} c_i x^i$.

Travailler sur les mots de codes revient alors à effectuer des calculs dans l'anneau des polynômes modulo (x^n-1) : $K[x]/(x^n-1)$. On peut alors démontrer les théorèmes [13] :

- Pour un code cyclique C(n,k), il existe un unique polynôme g(x) de degré (n-k) s'écrivant : $g(x) = 1 + g_1x + ...+g_{n-k-1}x^{n-k-1} + x^{n-k}$
- Tout mot de code est un multiple de g(x) appelé *polynôme générateur du code C(n,k).*
- Tout polynôme de degré inférieur à (n-k) et multiple de g(x) est un mot de code.
- g(x) est un polynôme générateur de C(n,k) $\Rightarrow$ g(x) divise x^n-1

D'après ce qui précède tout mot de code C peut s'écrire comme le produit de deux polynômes u(x) et g(x) respectivement de degré (k) et (n-k). Il s'ensuit que si les coefficients de u(x) sont les données à coder, une multiplication de ce polynôme avec g(x) permet d'obtenir le mot code.

I.3.3. Processus d'encodage.

I.3.3.1. Codage systématique.

On vient de voir que tous les mots de codes sont des multiples du polynôme générateur. L'algorithme le plus simple pour calculer le mot code est alors d'effectuer la multiplication entre la donnée à coder et g(x). Dans ce cas, la donnée n'apparaît pas en clair dans le mot de code ce qui constitue déjà une protection. Le codage systématique est une autre forme de codage qui permet que la donnée à coder soit visible dans le mot de code (figure I-12). C'est ce que nous allons voir avec l'algorithme de codage par division.

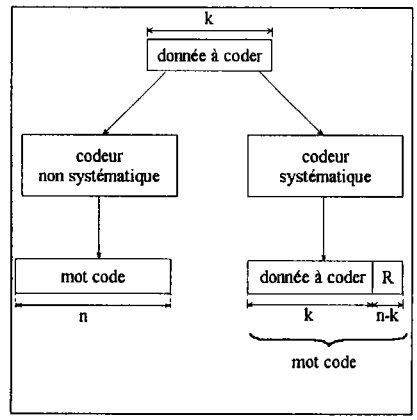


Fig I-12. Principe d'un codage systématique.

I.3.3.2. Algorithme de codage par division.

On considère une donnée à coder M de taille k notée $[m_0,m_1,\ldots,m_{k-1}]$. On va lui associer le polynôme M(x) égal à $\sum_{i=0}^{i=k-1} m_i x^i$. D'après le théorème d'Euclide, il existe deux polynômes p(x) et r(x) tels que : $x^{n-k}M(x) = p(x)g(x) + r(x)$ avec deg r(x) < n-k. Comme nous travaillons avec des codes binaires, la soustraction est équivalente à l'addition (en arithmétique modulo 2, -1 ≡1 [2]). Par conséquent,

$$[x^{n-k}M(x) + r(x)] \text{ est un multiple de } g(x) \quad (1.6)$$

D'après le paragraphe I.3.2 on peut en conclure que l'expression (1.6) est un mot de code dont la représentation est la suivante : $[m_0,m_1,\ldots,m_{k-1},r_0,..r_{n-k-1}]$. On a donc bien réalisé un codage systématique pour lequel trois étapes de calculs sont nécessaires.

1- Prémultiplication de la donnée M(x) par x^{n-k} (ce qui revient à rajouter n-k zéros).

2- Calcul du reste r(x) de la division entre $x^{n-k}M(x)$ et le polynôme générateur.

3- Combinaison entre le reste calculé à l'étape 2 et la donnée à coder pour former le mot code (XOR bit à bit entre $x^{n-k}M(x)$ et r(x)).

Les étapes 1 et 3 sont facilement réalisables. L'opération la plus coûteuse en temps de calcul est l'évaluation du reste de la division entre $x^{n-k}M(x)$ et g(x). Nous allons maintenant donner l'exemple d'un diviseur qui calcule le reste de la division entre $(x^{n-k}M(x))$ et g(x) en n cycles.

I.3.3.3. Exemple d'implantation d'un codeur par division.

I.3.3.3.1. Préliminaire- conception d'un multiplicateur

Soit D une donnée (taille k) à multiplier par un polynôme g de degré r. Le produit P de ces deux polynômes s'écrit comme :

$$D(x)g(x) = \sum_{i=0}^{i=k-1}\sum_{j=0}^{j=r-1} d_i g_j x^{i+j} = d_{k-1}g_{r-1}x^{k+j-2} + .. + \left(\sum_{i,j}^{i+j=k+r-m} d_i g_j\right) x^{k+r-m} + .. (1.7)$$

Intéressons nous par exemple aux deuxième et troisièmes termes du produit. On peut écrire les égalités :

$$\begin{cases} p_{k+r-3} = d_{k-2}\, g_{r-1} + d_{k-1}g_{r-2} \\ p_{k+r-4} = d_{k-3}g_{r-1} + d_{k-2}g_{r-2} + d_{k-1}g_{r-2} \end{cases} \quad (1.8)$$

On peut remarquer que les données (d_{k-2}, d_{k-1}) semblent avancer relativement aux coefficients de g(x) comme le montre la figure I-13.

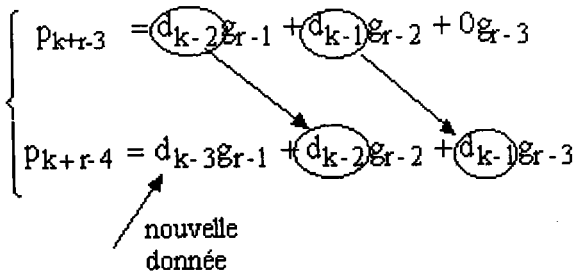


Fig I-13. Principe du multiplieur

En utilisant, cette propriété [20], il est possible de concevoir un circuit (constitué uniquement de portes XOR et de flip-flop) qui calcule de proche en proche les différents coefficients du polynôme P (figure I-14a). En observant ce circuit, on s'aperçoit que les portes XOR en série vont limiter le débit de sortie de ce circuit. En réorganisant les équations de la formule (1.7), on peut concevoir sur le même principe un circuit permettant de s'affranchir de ces limitations (figure I-14b). Sur un FPGA de la famille des Flex10k (EPF10K0LC84 d'Altera), les deux versions d'un multiplieur polynomial

par $(1+x+x^3+x^4+x^5)$ peuvent fonctionner jusqu'à des fréquences maximales respectives de 115 Mhz et 125 Mhz.

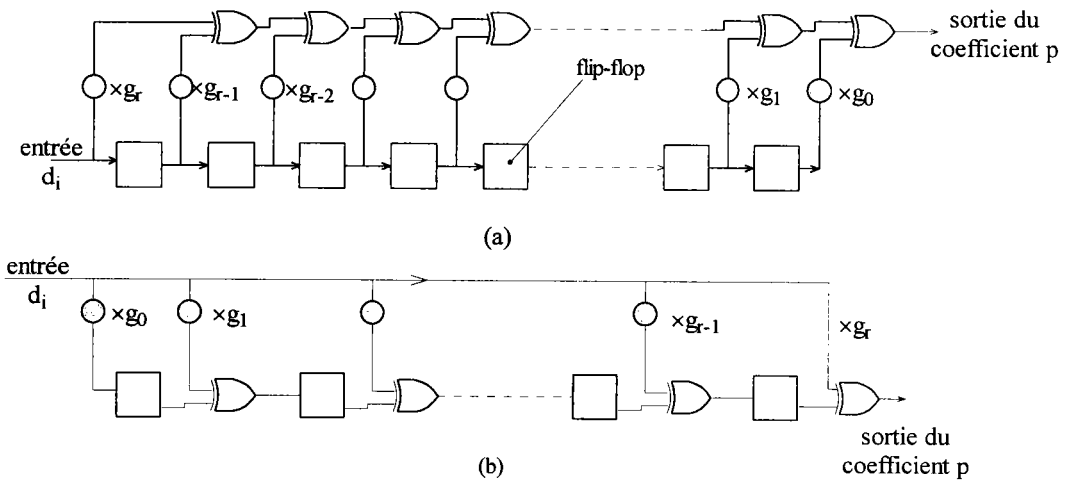


Fig I-14. Circuit de multiplication de deux polynômes

I.3.3.3.2. Circuit de division.

En utilisant un circuit similaire à celui de la figure I-14, il est possible de calculer le reste de la division entre un polynôme D de degré *k* et un polynôme g de degré *r*.

On considère les deux polynômes $R_{(i)}$ et $Q_{(i)}$ respectivement le reste et le plus petit terme du quotient de la division de D par g(x). Ces deux polynômes vérifient les équations (I.9) et (I.10).

$$\begin{cases} R_{(i+1)}(x) = Q_{(i)}g(x) - R_{(i)}(x) & (1.9) \\ Q_{(i)} = \dfrac{r_{\deg(R_{(i)})}}{g_r} x^{\deg(R_{(i)})-r} & (1.10) \end{cases}$$

Pour un code linéaire cyclique, g_r est toujours égal à un. L'équation (I.9) peut alors s'écrire :

$$R_{(i+1)}(x) = r_{\deg R_{(i)}} x^{\deg(R_{(i)})-r} \times g(x) - R_{(i)}(x) \qquad (1.11)$$

En posant $H(x)+1 = x^{-r}g(x)$, la relation (I.11) peut s'écrire :

$$R_{(i+1)}(x) = r_{\deg R_{(i)}} x^{\deg(R_{(i)})} H(x) - \sum_{j=0}^{j=\deg R(i)-1} r_{ij} x^j \qquad (1.12)$$

La division par g(x) revient alors à utiliser conjointement un circuit de multiplication par H(x) et un rétro-bouclage (deuxième terme de l'équation I.13) pour pouvoir calculer le reste. En utilisant des graphes de fluence [21], on peut montrer que l'équation peut être implantée par le circuit de la figure I-15.a. Pour les mêmes raisons que précédemment, le débit de sortie d'un tel circuit est limité. En général, on lui préfère le circuit de la figure I-15.b.

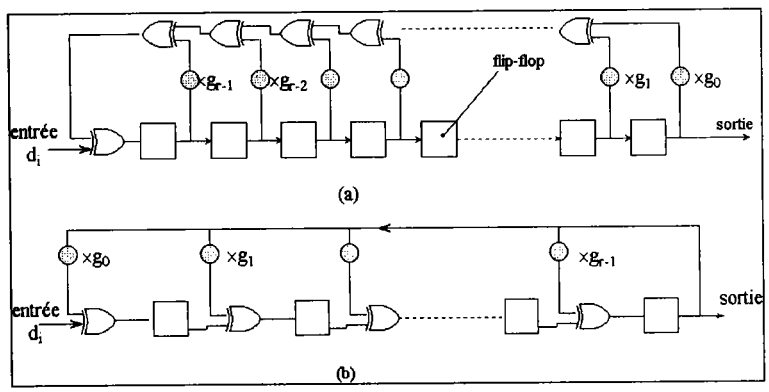


Fig I-15. Exemple de circuit de division polynomiale

I.3.3.3.3. Propriétés.

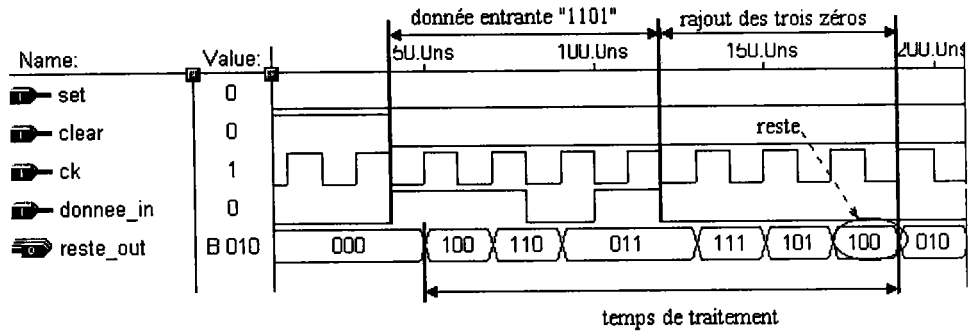


Fig I-16. Exemple de simulation d'un diviseur ($g(x) = 1+x+x^3$)

Comme pour le multiplicateur, *n* cycles d'horloges sont nécessaires afin d'obtenir le reste de la division d'un polynôme de degré *n* par un polynôme de degré *r*. La figure I-16 donne un exemple de simulation du diviseur pour un code cyclique (7,4) dont le polynôme générateur est $1+x+x^3$.

Pour un codage systématique, on doit d'abord effectuer une prémultiplication par x^r. En modifiant la position d'entrée des données (fig I.17), il est possible de s'affranchir de

la multiplication [13]. Le calcul du reste ne nécessite plus alors que k cycles d'horloges, ce qui est avantageux. La figure I-18 montre la simulation d'un diviseur de ce type pour le code (7,4) précédent.

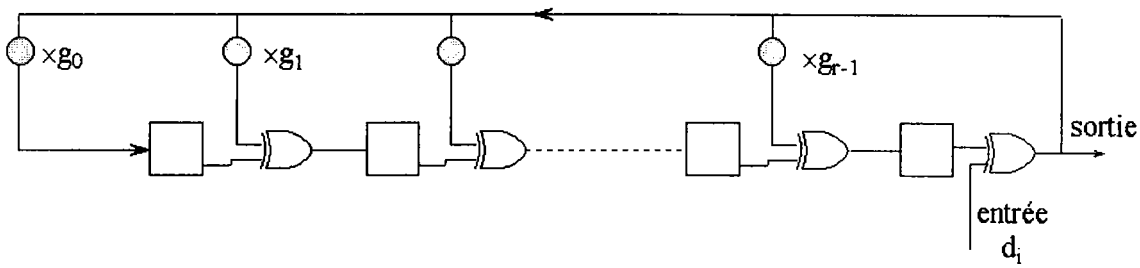


Fig I-17. Circuit modifié pour le calcul du reste

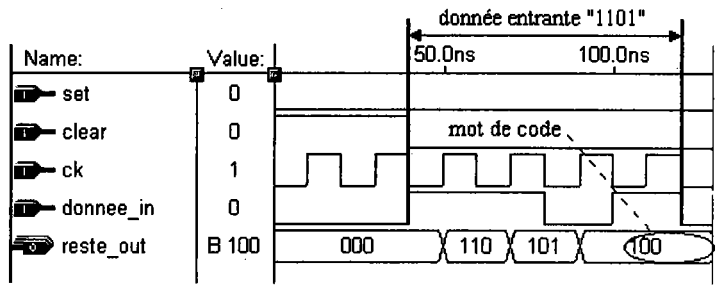


Fig I-18. Exemple de simulation d'un diviseur du second type ($g(x) = 1+x+x^3$)

I.3.4. Processus de décodage

I.3.4.1. Détection des erreurs

Le calcul du syndrôme ($S = C'H^t$) exposé dans le paragraphe I.2.2.3.2 reste valable pour les codes cycliques. Cependant, dans le cas d'un codage systématique, le calcul de S est simplifié.

La donnée reçue C' est assimilé à un polynôme de degré n. Par construction, un mot de code est un multiple du polynôme générateur. L'examen du reste de la division de C' par le polynôme générateur permet de décider si le message reçu contient des erreurs :

- si le reste est différent du polynôme nul, le mot reçu n'est pas un multiple du polynôme générateur. Par conséquent, ce n'est pas un mot de code (paragraphe I.3.2.).
- si le reste est égal à zéro, la donnée reçue est un multiple de g(x) et est donc un mot de code. On ne peut pas cependant exclure le cas où les erreurs transforment un mot de code en un autre mot de code (cas non détectable).

Le calcul du syndrôme passe alors d'un calcul matriciel au calcul d'un reste de division. Pour l'effectuer, on peut utiliser le circuit utilisé pour l'encodage. Le calcul du syndrôme étant effectué, l'algorithme de décodage nécessite alors deux autres étapes :

- Evaluation de l'erreur à partir du syndrôme en utilisant soit une RAM soit une logique câblée
- La correction des erreurs en faisant l'addition modulo 2 de l'erreur et du vecteur reçu. Pour un code de longueur *n*, *n* portes XOR sont alors nécessaires.

Une des limites de cette approche est que la complexité du circuit de décodage augmente fortement avec la taille du mot code. Le décodeur de Meggitt, en utilisant judicieusement les propriétés du syndrôme, diminue la complexité du décodage.

I.3.4.2. Correction des erreurs : le décodeur de Meggitt

I.3.4.2.1. Propriété du syndrôme

Soit S(x) le syndrôme du vecteur $C = [c_0, c_1, \ldots, c_{n-1}]$.

On appelle $C'^{(1)}$ le vecteur $[c_{n-1}, c_0, .., c_{n-2}]$ (permutation circulaire d'un rang vers la droite). Alors le reste de la division de xS(x) par g(x) est égal au syndrôme de $C'^{(1)}$.

I.3.4.2.2. Le décodeur de Meggitt.

En utilisant la structure cyclique du code, il est possible de décoder en série les données reçues. En effet, dans l'étape 2 du décodage, la table utilisée recherche à partir du syndrôme la valeur du vecteur d'erreur (de taille n). On peut limiter cette table à l'ensemble des valeurs de syndrôme correspondant à une erreur en position (n-1). Deux cas peuvent alors se produire :

- Le syndrôme permet de conclure qu'il n'y a pas d'erreur en position (n-1). On effectue alors un décalage à droite du syndrôme. En utilisant la propriété du I.3.4.2.1, le

syndrôme obtenu est égal au syndrôme du vecteur $[c'_{n-1},c'_0,\ldots,c'_{n-2}]$. On peut alors tester si le symbole c'_{n-2} est erroné.

- Le syndrôme indique qu'il y a une erreur en position (n-1). Le nouveau vecteur corrigé est alors : $C_{corr} = c'_0 + c'_1 x + .. + c'_{n-2} x^{n-2} + (1 + c'_{n-1}) x^{n-1}$. Pour tester le symbole c'_{n-2}, on effectue un décalage circulaire. D'où le nouveau vecteur :

$$C^{(1)}_{corr} = (1 + c'_{n-1}) + c'_0 x + c'_1 x^2 + .. + c'_{n-2} x^{n-1}$$

Sachant que le polynôme générateur divise x^n+1, et en appelant $S^{(1)}_{corr}$ le syndrôme de $C^{(1)}_{corr}$, on peut démontrer la relation :

$$S^{(1)}_{corr} = S^{(1)} + 1 \quad (1.13)$$

Par conséquent un '1'est ajouté au syndrôme (bit de poids le plus faible) pendant que celui ci est décalé d'un bit vers la droite.

Un exemple d'implantation d'un décodeur (décodeur de Meggitt) [19] pour code cyclique est représenté à la figure I-19. Il est constitué trois unités : le calcul du syndrôme, un bloc d'évaluation de l'erreur et d'un registre servant à conserver le mot reçu. Le processus de décodage se fait alors en cinq étapes.

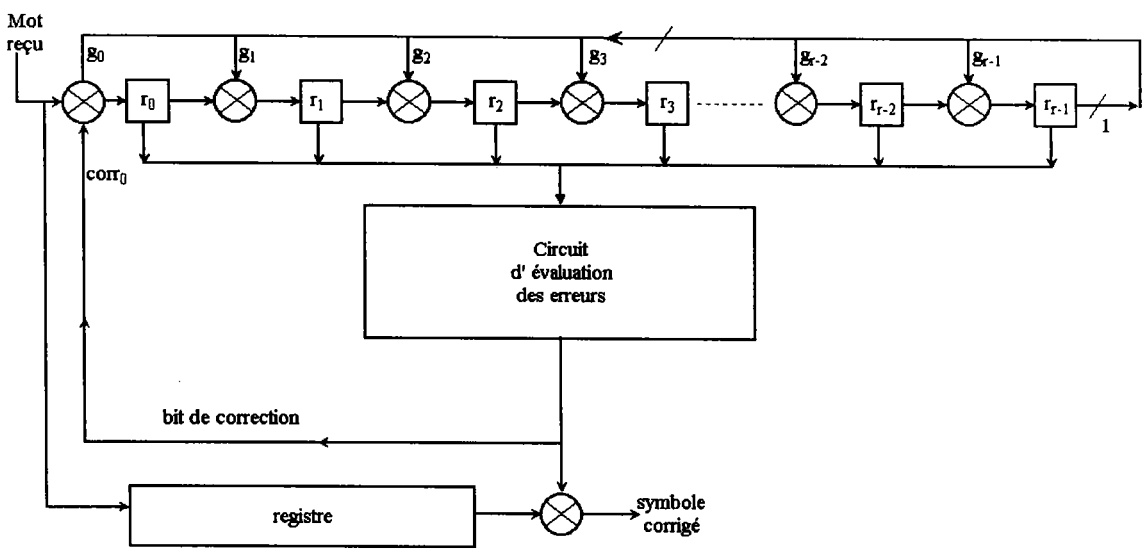


Fig I-19. Décodeur de Meggitt

• *Première étape* : Le syndrôme est calculé à l'aide du circuit de division. Cette opération nécessite *n* cycles d'horloges.

• *Deuxième étape* : Le circuit d'évaluation utilise le syndrôme pour déterminer si le bit en position (n-1) est erroné. Si la sortie de ce circuit est à 1, le bit situé à la position n-1 est considéré comme erroné et une correction est nécessaire. Dans le cas contraire (sortie à 0), la correction n'est pas nécessaire.

• *Troisième étape* : Le registre conservant le mot reçu est décalé d'un bit vers la droite afin de permettre la correction du bit se situant à la position (n-1). Pour cela, on effectue une somme modulo 2 (XOR) entre le bit à la position (n-1) et la sortie du circuit d'évaluation des erreurs.

Pendant le même temps, le contenu du syndrôme est décalé d'un bit vers la droite avec le rajout de la correction conformément à la formule 1.13.

• *Quatrième étape* : Le nouveau syndrôme est utilisé pour évaluer l'erreur du bit se situant à la position (n-2) dans le mot reçu et le corriger dans le cas d'une éventuelle erreur (étape 2 et 3).

• *Cinquième étape* : Le circuit décode bit par bit le mot reçu. Lorsque la dernière correction a été effectuée, le syndrôme doit être un polynôme nul. Dans le cas contraire, le nombre d'erreurs dans le mot reçu était supérieur à la capacité de correction du code (nombre maximum d'erreurs pouvant être corrigées). Le mot décodé est alors inexact.

Ce décodeur connu sous le nom de décodeur de Meggitt est valable pour tous les codes cycliques. Son implantation ne dépend que de la complexité du circuit d'évaluation des erreurs. Pour certains codes, ces circuits sont simples. On peut citer par exemple le décodage par piégeage d'erreurs, le décodage pour les codes BCH, le décodage par logique majoritaire.

I.4. Conclusions

Au vu de ce qui précède, on ne peut pas conclure qu'un code ou qu'une méthode de correction soit plus valable qu'une autre. Le choix de la méthode utilisée dépend uniquement de l'application choisie, de ces contraintes (types de canal utilisé, débit demandée à l'application) et du type d'erreurs pouvant survenir sur le canal de transmission (erreurs isolées ou en paquet, fréquence d'apparition des erreurs). Sur le tableau suivant, on a donné quelques exemples d'applications avec la méthode de correction utilisée.

Famille de code	Type de code	Applications caractéristiques
BCH	code linéaire cyclique	European Video_conference Experiment (EVE) Domaine satellitaire
DSCC	code linéaire cyclique	Système teletext japonais transmission de donnée en FM
Reed-Solomon	code linéaire cyclique	Compact disc, internet télévision, norme DAVIC et MCNSE
Hamming	code linéaire	Test
Bit de parité	code détecteur	Transmission série
Code de Goppa	code linéaire	Cryptographie

Fig I-20. Exemple d'applications de codes correcteurs d'erreurs

Dans notre cas, nous nous sommes focalisés sur deux familles de codes mais avec des objectifs différents.

Pour la première famille (les codes à logiques majoritaires), nous allons étudier une technique permettant d'améliorer la vitesse de traitement de ces circuits afin qu'ils puissent être utilisés dans des applications à haut débit (chapitre II). Par contre pour la deuxième famille (code Reed Solomon), nous allons surtout étudier l'implantation d'une technique permettant d'augmenter la capacité de correction du code donc de diminuer le coût de transmission (chapitre III).

Chapitre II. Les codes à logiques majoritaires

Après l'étude générale des codes correcteurs, nous allons nous intéresser en premier lieu au codage et au décodage d'une famille particulière de code : les codes à logique majoritaire. Dans une première partie, nous allons présenter les méthodes de codage et de décodage de cette famille ainsi que les principaux codes qui la compose (DSCC, DTI, EG). Dans la deuxième et troisième partie, nous étudierons les problèmes posés par l'implantation en ASIC des codes à logique majoritaire. Enfin, dans la dernière partie, nous donnerons les résultats (débit et surface) sur l'implantation choisie.

II.1. Le décodage à logique majoritaire

II.1.1. Principe de décodage [13]

Soit C' le message reçu de taille *n*. D'après ce qui a été dit précédemment, ce vecteur peut s'écrire comme la somme des deux vecteurs C et E qui sont respectivement le mot code envoyé et un vecteur de taille *n* représentant les erreurs ayant entaché le message d'origine. Pour tout vecteur W appartenant au code dual de C ($C^{\perp}$), on peut écrire la relation suivante :

$$W.C' = W.(C+E) = W.E = w_0e_0 + w_1e_1+\ldots+w_{n-1}e_{n-1}$$

Supposons qu'il existe J vecteurs W dans $C^{\perp}$ tels que :

- $\forall j \in \{1,\ldots,J\}\ w_{j,n-1} = 1$.
- Pour tout i différent de n-1, il existe au plus un vecteur dont la $i^{éme}$ composante $w_{j,i}$ est égale à 1.

Ces J vecteurs sont alors appelés *vecteurs orthogonaux à la position n-1*. On obtient les J équations suivantes :

$$\begin{aligned} W_1 &= w_{1,0}\, e_0 + w_{1,1} e_1 + \ldots\ w_{1,n-2} e_{n-2} + e_{n-1} \\ W_2 &= w_{2,0}\, e_0 + w_{2,1} e_1 + \ldots\ w_{2,n-2} e_{n-2} + e_{n-1} \\ &\vdots \\ W_J &= w_{J,0}\, e_0 + w_{J,1} e_1 + \ldots\ w_{J,n-2} e_{n-2} + e_{n-1} \end{aligned} \qquad (2.1)$$

Ces J équations peuvent être utilisées pour déterminer s'il y a une erreur en position n-1. En effet, supposons qu'il y ait au plus $\lfloor J/2 \rfloor$ erreurs dans le message reçu (limite de correction du code).

S'il y a une erreur en position n-1, les erreurs restantes se répartissent dans au plus $\lfloor J/2 \rfloor$-1 équations orthogonales. Par conséquent, $J-\lfloor J/2 \rfloor+1$ équations au moins sont égales à 1 soit une majorité des équations.

S'il n'y a aucune erreur en position n-1, les erreurs se répartissent dans au plus $\lfloor J/2 \rfloor$ équations. Par conséquent, il y a au moins $J-\lfloor J/2 \rfloor$ équations qui sont égales à zéro.

L'étude de la sortie des J équations permet alors de décider si le bit à la position (n-1) est erroné. Cependant, le récepteur n'a accès qu'à l'information C' et ne connaît ni C ni E. En conséquence la méthode précédente doit être modifiée. On peut alors démontrer qu'il y a une relation entre les coefficients $w_{i,j}$ et les bits S_i du reste de la division de R(x) par g(x). Le calcul de la matrice W revient alors à la somme linéaire des bits du reste R avec les (n-k) premiers coefficients des vecteurs orthogonaux W_j

L'algorithme de correction nécessite alors 4 étapes :

• Etape 1 : La donnée reçue C' rentre dans le circuit de correction et le syndrôme S est calculé.

• Etape 2 : Les J équations orthogonales sont formées par combinaison des symboles du syndrôme.

• Etape 3 : Ces J équations constituent les entrées d'une porte à logique majoritaire dont la sortie est égale à la valeur qui est prise par la majorité des entrées.

• Etape 4 : le symbole de correction est réintroduit dans le calcul du reste et l'algorithme recommence à partir de l'étape 2 pour la correction du deuxième symbole.

n cycles de calculs sont nécessaires pour obtenir le message décodé.

II.1.2. Le code Difference Set Cyclic Code (DSCC) [24]

Le code DSCC appartient à la famille des codes linéaires cycliques à décodage à logique majoritaire. Sa théorie est basée sur la construction d'une série parfaite de différence finie [22].

II.1.2.1. Série parfaite de différence finie

Soit P un ensemble de L+1 entiers non nuls tels que : $0 \leq q_0 \leq .. \leq q_L \leq L(L+1)$. A partir de cet ensemble, il est possible de former L(L+1) différences ordonnées D définies par $D = \{q_j - q_i / j \neq i\}$. De façon évidente, la moitié des différences ordonnées est négative.

On dit alors que l'ensemble P est une *série parfaite de différence finie d'ordre L* si et seulement si :

1- Toutes les différences ordonnées négatives sont distinctes.

2- Toutes les différences ordonnées positives sont distinctes.

3- Si $q_j - q_i$ est une différence négative alors $L(L+1)+1+ q_j - q_i$ n'est égal à aucune différence positive

Par exemple, si on considère l'ensemble P = {0, 2, 7, 8, 11} l'ensemble D des différences est égal à {2,7,8,11,5,6,9,1,4,3,-2,-7,-8,-11,-5,-6,-9,-1,-4,-3}. Les propriétés 1 et 2 sont respectées. L'ensemble V = { $L(L+1)+1+ q_j-q_i$ / $j \neq i$} est égal à {19,4,13,10,16,15,12,20,17,18}. La propriété 3 est vérifiée. L'ensemble V est une série parfaite de différence finie

Singer [23] a trouvé une méthode pour calculer les séries parfaites de différences finies quand L est égal à p^s (p premier et s un entier positif). Nous nous placerons dans le cas où p est égal à 2 (corps de Galois).

II.1.2.2. Construction du polynôme générateur [24]

Soit P = $\{0,q_1,..,q_{2^s}\}$ une série parfaite de différence finie d'ordre 2^s. On définit le polynôme Z(x) par :

$$Z(x) = \sum_{i=0}^{i=2^s} x^{q_i} \qquad (2.2)$$

On pose $H = \sum_{i=0}^{i=k-1} h_i x^i$ le plus grand commun diviseur de Z(x) et de x^n+1 (avec n égal à $2^s(2^s+1)+1$).

Alors le polynôme G(x) définit par la formule (2.3) génère un code cyclique de longueur n appelé code DSCC.

$$G(x) = \frac{x^n+1}{H(x)} = \sum_{i=0}^{i=n-k} g_i x^i \qquad (2.3)$$

Il s'ensuit que les paramètres du code sont :

Taille du mot code	$n = 2^s(2^s+1)+1$
Taille de la redondance	$n-k = 3^s+1$ [22]
Taille du mot à coder	$k = 2^s(2^s+1)- 3^s+1$

Le tableau 3 donne les polynômes générateurs des 5 premiers codes DSCC.

s	Polynôme générateur *	difference set *
1	0, 2, 3, 4	3, 2, 0
2	0, 2, 4, 6, 7, 10	11, 8, 7, 2, 0
3	0, 2, 4, 6, 8, 12, 16, 22, 25, 28	45, 42, 36, 29, 25, 24, 10, 2, 0
4	0, 4, 10, 18, 22, 24, 34, 36, 40, 48, 52, 56, 66, 67, 71, 76, 77, 82	201, 196, 186, 167, 166, 159, 128, 126, 115, 112, 103, 67, 50, 46, 24, 18, 0
5	0, 1, 3, 4, 5, 11, 14, 17, 18, 22, 23, 26, 27, 28, 32, 33, 35, 37, 39, 41, 43, 45, 47, 48, 51, 52, 55, 59, 62, 68, 70, 71, 72, 74, 75, 76, 79, 81, 83, 88, 95, 96, 98, 101, 103, 105, 106, 108, 111, 114, 115, 116, 120, 121, 122, 123, 124, 126, 129, 131, 132, 135, 137, 138, 141, 142, 146, 147, 149, 150, 151, 153, 154, 155, 158, 160, 161, 164, 165, 166, 167, 169, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 186, 188, 189, 191, 193, 194, 195, 198, 199, 200, 201, 202, 203, 208, 209, 210, 211, 212, 214, 216, 222, 224, 226, 228, 232, 234, 236, 242, 244	1023, 990, 924, 905, 879, 792, 754, 702, 697, 677, 597, 555, 528, 511, 452, 439, 348, 338, 298, 277,255, 219, 138, 127, 109, 63, 54, 31, 15, 7, 3, 1, 0

* chaque polynôme est représenté par les exposants de ces termes non nuls

Tableau 3. Liste des premiers codes DSCC

II.1.2.3. Construction de la matrice W [9]

Soit H*(x) le polynôme réciproque de H défini par la relation $H^*(x) = x^{q_{2^s}} H(x^{-1})$. On peut démontrer que ce vecteur génère un code orthogonal au code DSCC considéré (espace orthogonal du code DSCC) [13] [17]. Comme le polynôme Z est divisible par H, Z* est aussi divisible par H* et appartient à l'espace orthogonal.

Soit l'ensemble des équations $\{W_i / i \in \{0,1,\ldots,2^s\}\}$ définies par :

$$W_i = x^{n-1-q_{2^s}+q_i} Z^*(x) \quad (2.4)$$

Chaque polynôme W_i appartient à l'espace orthogonal. De plus comme $\{0, q_1, .., q_{2^s}\}$ est une série parfaite de différence simple, les polynômes W_i et W_j ($i \neq j$) n'ont en commun que le terme x^{n-1}. Par conséquent, $\{W_i / i \in \{0,1,\ldots,2^s\}\}$ est un ensemble de 2^s+1 polynômes orthogonaux à la position (n-1).

On peut alors prouver que ce code est capable de corriger 2^{s-1} erreurs [23].

II.1.2.4. Exemple des éléments d'un code DSCC

Si on considère la série parfaite de différence finie P = {0, 2, 7, 8, 11}. H(x) doit être le plus grand diviseur commun de $(X^{21}+1)$ et de $(1+x^2+x^7+x^8+x^{11})$. On trouve alors que H est égal à $1+x^2+x^7+x^8+x^{11}$.

Par conséquent, le polynôme générateur est égal à $(X^{21}+1)/H(x)$ soit :

$G(x) = 1+x^2+x^4+x^6+x^7+x^{10}$

Le code DSCC est donc un (21,11) code cyclique.

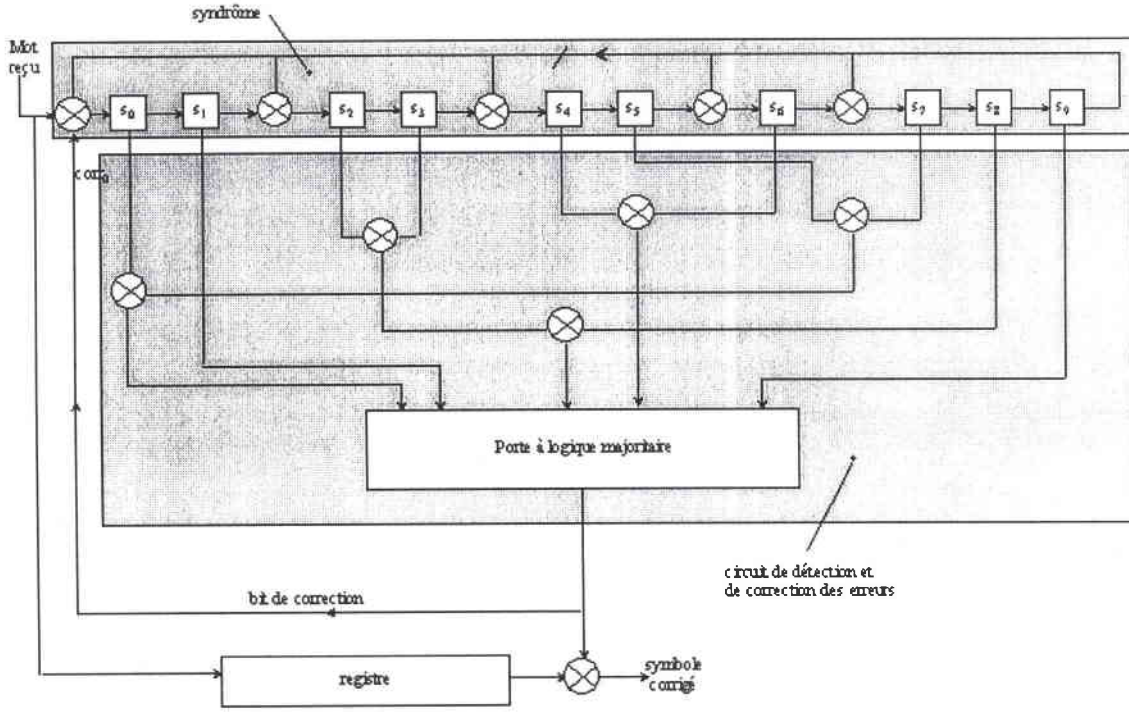


Fig II-1. Exemple d'implantation du décodeur DSCC (21,11)

Le polynôme Z*, quant à lui, est égal à $x^{11}Z(x^{-1}) = 1+x+x^4+x^9+x^{11}$. On en déduit les équations W_i :

$$W_0 = x^9 + x^{12} + x^{13} + x^{18} + x^{20}$$

$$W_1 = x + x^{11} + x^{14} + x^{15} + x^{20}$$

$$W_2 = x^4+x^6+x^{16}+x^{19}+x^{20}$$

$$W_3 = 1+x^5+x^7+x^{17}+x^{20}$$

$$W_4 = x^2+x^3+x^8+x^{10}+x^{20}$$

On peut alors former les équations utiles pour le décodage (voir figure II-1).

$A_0 = s_9 = e_9+e_{12}+e_{13}+e_{18}+e_{20}$

$A_1 = s_1 = e_1+e_{11}+e_{14}+e_{15}+e_{20}$

$A_2 = s_4+s_6 = e_4+e_6+e_{16}+e_{19}+e_{20}$

$A_3 = s_0+s_5+s_7 = e_0+e_5+e_7+e_{17}+e_{20}$

$A_4 = s_2+s_3+s_8 = e_2+e_3+e_8+e_{10}+e_{20}$

Les codes DSCC sont aussi puissants que les meilleurs codes cycliques mais dans cette classe de code, il y a peu de codes avec des paramètres utilisables.

II.1.3. Code Doubly Transitive Invariant (DTI)

II.1.3.1. Théorie mathématique

II.1.3.1.1. Permutation affine

Soit C(n,k) un code linéaire cyclique de taille $n = 2^m-1$. Chaque mot de code peut s'écrire $(c_0, c_1,\ldots, c_{n-1})$. On peut alors étendre chaque mot de code par un bit de parité c_p. Le nouveau vecteur est alors de taille 2^m et s'écrit $(c_p, c_0, c_1,\ldots, c_{n-1})$. L'ensemble des mots de code forme alors un code linéaire $C_e(n+1,k)$ appelé *extension de C.*

On considère alors α un élément primitif du corps de Galois $GF(2^m)$ (annexe A). On peut faire une correspondance entre les éléments de $GF(2^m)$ et les positions des symboles dans le mot code, c'est à dire :

- le symbole c_p situé à la position 0 est repéré par α^∞.
- les symboles c_i situé à la position i est repéré par α^i , i appartenant à $\{1,..,2^m-2)$,

On définit la permutation affine par l'opération qui déplace le symbole de la position Y à la position Z = aY+b avec a et b deux éléments quelconques de $GF(2^m)$.

II.1.3.1.2. Propriété d'invariance

II.1.3.1.2.1. Notion de descendance

Soit h un entier non nul inférieur à 2^m. Cet entier peut s'écrire sous une forme binaire ($h = \sum_{i=0}^{2^m-1} \chi_i 2^i$). Considérons $h'(= \sum_{i=0}^{2^m-1} \chi'_i 2^i)$ un entier non nul inférieur à 2^m et différent de h.

h' est dit descendant de h si χ'_i est inférieur ou égal à χ_i pour i appartenant à l'ensemble{1,..,m-1}. On note $\Delta(h)$ l'ensemble des descendants non nuls de h.

II.1.3.1.2.2. Invariance

On va utiliser la propriété de descendance pour avoir une condition nécessaire et suffisante pour qu'un code cyclique étendu C_e soit invariant par permutation affine. Dans [25], on démontre le théorème suivant (Doubly Transitive Invariant property) :

Soit C un code cyclique généré par le polynôme g(x). On définit le code étendu C_e à partir de C en rajoutant un bit de parité. Soit α un élément primitif du corps de Galois $GF(2^m)$. Le code C_e est invariant pour les permutations affines si et seulement si :

- pour toute racine α^h du polynôme g(x) et pour tout h' appartenant à $\Delta(h)$, $\alpha^{h'}$ est une racine de g(x),
- $\alpha^0 = 1$ n'est pas une racine de g(x).

Considérons alors le code C_e invariant pour les permutations affines. On applique à ce code la permutation $Z = \alpha Y$. Cela revient à laisser inchanger le symbole à la position 0 et faire un décalage cyclique à droite de tous les autres symboles. On peut alors en déduire que le code C construit en supprimant le symbole à la position 0 est cyclique. On va utiliser cette propriété pour calculer une nouvelle classe de code.

II.1.3.2. Code DTI de type 0

II.1.3.2.1. Construction du polynôme générateur

Soient J et L deux entiers tels que le produit de J et de L soit égal à n (= 2^m-1). Le polynôme x^n+1 peut alors s'exprimer comme :

$$x^n+1=(1+x^J)\left(\sum_{i=0}^{i=(L-1)J} x^i\right)=(1+x^J)\delta(x) \quad (2.5)$$

Sachant que $(\alpha^L)^J$ est égal à 1 (α est un élément primitif de GF(2^m)), l'ensemble des racines du polynôme $(1+x^J)$ est égal à $\{0, \alpha^L, \alpha^{2L},.., \alpha^{(J-1)L}\}$. L'exposant i des racines α^i de $\delta(x)$ n'est alors pas un multiple de L.

On va alors utiliser les racines de $\delta(x)$ pour construire un nouveau polynôme H(x) de telle manière que α^i est racine de H(x) si et seulement si

1- α^i est une racine de $\delta(x)$.

2- $\forall j \in \Delta(i)$ α^j est aussi une racine de $\delta(x)$.

H est alors le plus petit commun multiple des polynômes minimaux (voir annexe A) des racines α^i de H. D'après le paragraphe II.1.3.1.2.2, le code cyclique engendré par H vérifie la propriété d'invariance. Par conséquent, le code étendu H' est invariant pour les permutations affines. Si on considère le code dual du code engendré par H, ce code est aussi un code cyclique. Le polynôme générateur de ce code vérifie alors la relation suivante :

$$g(x)=x^{n-degH}\times G(x^{-1}) \quad (2.6)$$

II.1.3.2.2. Construction de la matrice W

Par construction, $\delta(x)$ est un multiple de H. Son degré étant inférieur à n, δ est un mot de code généré par H. Il s'ensuit que les polynômes $\{x^p\delta(x) / p\in \{1,..,J-1\}\}$ sont aussi des mots de code par définition des codes cycliques.

Or pour p et q différents, les polynômes $x^p\delta(x)$ et $x^q\delta(x)$ n'ont aucun terme en commun (par construction de δ). En ajoutant à ces mots de code un bit de parité, on obtient des mots de code de longueur 2^m et qui appartiennent à l'ensemble généré par H'. Tous ces mots de code $\{h'_0,\ldots,h'_{j-1}\}$ vérifient alors les propriétés suivantes :

- Ils ont en commun le terme α^∞ (le bit de parité est égal à 1 car J est impair)
- Le terme x^p ($p\in\{1,..n\}$) n'apparaît que dans un polynôme et un seul.

On en conclut que tous ces polynômes sont orthogonaux à la position 0 (α^∞). Comme les polynômes $\{h'_0,\ldots,h'_{j-1}\}$ appartiennent à l'ensemble généré par H', ils sont invariants pour les permutation affines. Par conséquent, les polynômes $\{p'_0,\ldots,p'_{j-1}\}$ obtenu en appliquant la permutation $\alpha Y+\alpha^{n-1}$ appartiennent à l'ensemble généré par H'. De plus, tout ces vecteurs sont orthogonaux à la position n-1 (le bit de parité situé à la position 0 est « transporté » à la position n-1).

En supprimant le bit se situant à la position 0 (α^∞), on obtient alors J équations w_i orthogonales à la position n-1. On en conclut que le code généré par le polynôme $g(x) = x^{n-degH} \times G(x^{-1})$ est code cyclique à décodage à logique majoritaire capable de corriger E(J/2) erreurs. Ce code est appelé code DTI de type 0.

II.1.3.2.3. Exemple

On considère n égal à 15 (m = 4). On prend alors J et L égaux respectivement à 5 et 3. D'où le polynôme δ est égal à $1+x^5+x^{10}$. On considère α un élément primitif de $GF(2^4)$.

Par construction, l'ensemble des racines de $(1+x^5)$ est égal à $\{(\alpha^3)^i / 0< i \leq 4\}$ c'est à dire $\{\alpha^3, \alpha^6, \alpha^9, \alpha^{12}\}$.

L'ensemble des racines de δ est alors : $\{\alpha^i / 0< i \leq 14 \text{ et } i\neq \{3,6,9,12\}\}$

D'après les conditions sur les racines de H (voir tableau IV), le polynôme H a comme racine : $\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^5, \alpha^{10}$

Racine i (α^i)	Δ(i)	Racine retenue
1	$\varnothing$	Oui
2	$\varnothing$	Oui
4	$\varnothing$	Oui
5	{1, 4}	Oui
7	{1, 2, 3, 4, 5, 6}	Non
8	$\varnothing$	Oui
10	{2, 8}	Oui
11	{1, 2, 3, 8, 9, 10}	Non
13	{1, 4, 5, 8, 9, 12}	Non
14	{2, 4, 6, 8, 10, 12}	Non

Tableau 4. Liste des racines de δ et de H

Comme ($\alpha, \alpha^2, \alpha^4, \alpha^8$) sont des conjugués, ils ont le même polynôme minimum (voir annexe A). Il en est de même pour (α^5, α^{10}). En s'aidant des tables des polynômes minimaux [13 pp 579-582], H(x) est égal à $1+x^3+x^4+x^5+x^6$.

Par conséquent, $g(x) = x^{n-degH} \times \frac{x^{-n-1}}{H(x^{-1})} = 1 + x + x^4 + x^5 + x^6 + x^9$.

Les polynômes $\delta(x)$, $x\delta(x)$, $x^2\delta(x)$, $x^3\delta(x)$ et $x^4\delta(x)$ sont représentés dans le tableau (a) de la figure II-2. A la position 0 (α^∞) , on rajoute le bit de parité.

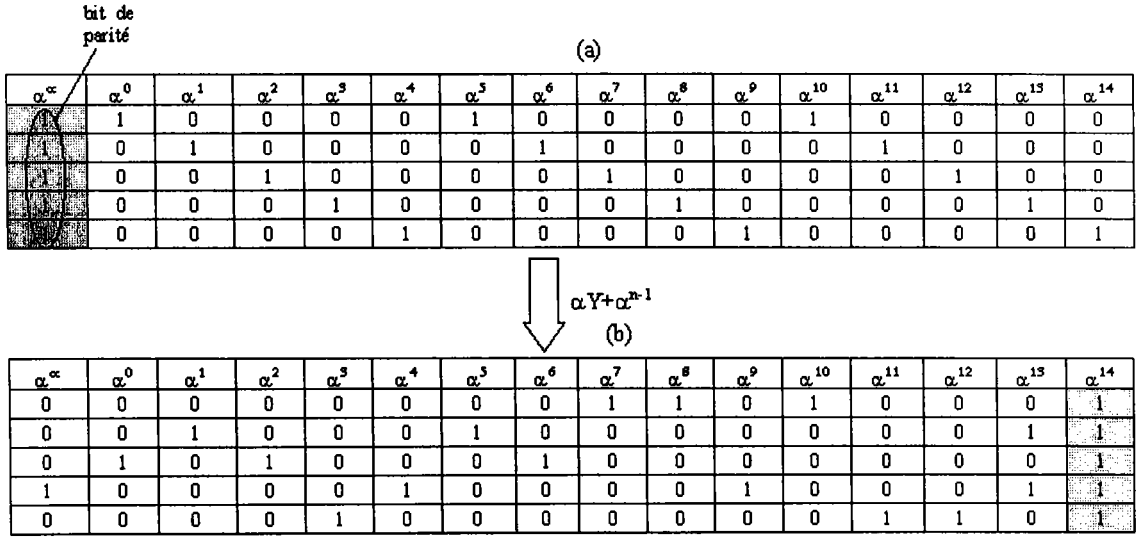


Fig II-2. Construction de la matrice orthogonale à la position α^{14}

En appliquant la permutation $\alpha Y+\alpha^{14}$, on obtient une matrice orthogonale à la position α^{14}.(Tableau b de la figure II.2). Si on supprime le bit situé à la position 0, on trouve la matrice W de décodage du code DTI (5,3) (figure II.3).

α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
0	0	0	0	0	0	0	1	1	0	1	0	0	0	1
0	1	0	0	0	1	0	0	0	0	0	0	0	1	1
1	0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	1	0	0	0	1	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0	1

Fig II-3. Matrice de décodage du code DTI (5,3).

Finalement, le code DTI (6,3) est un code cyclique C(15,9) généré par le polynôme $1+x+x^4+x^5+x^6+x^9$ et capable de corriger 2 erreurs ou moins.

II.1.3.3.. Code DTI de type I

En utilisant les différents polynômes du paragraphe précédent, on considère le nouveau polynôme $H_1(x)$ défini à partir de H par la formule :

$$H_1(x) = (x+1)H(x) \quad (2.7)$$

Le code cyclique généré par H_1 est un sous code de H. En réalisant la même opération que dans le paragraphe II.1.3.2, on définit un polynôme G_1 par la formule

$$g_1(x) = g(x)/(x+1) \quad (2.8)$$

Le polynôme g1(x) = g(x)/(x+1) est le polynôme générateur du code DTI de type I. Comme le degré de $g_1(x)$ est inférieur à celui de g(x), ce code permet de coder des données de taille plus importante (1 symbole de plus).

De plus, l'ensemble des mots de code de H_1 est composé uniquement des mots-code de H de poids pair (nombre de '1' dans le mot code est pair) [13]. Or, après la transformation linéaire, tous les vecteurs sont orthogonaux à la position α^{n-1}. Par conséquent, il existe un unique polynôme qui possède un 1 à la position α^{∞}. Ce vecteur

étant supprimé pour la construction de la matrice W, il existe un unique polynôme de poids impair. Les (J-1) vecteur restants appartiennent au code généré par H_1

La matrice W du code DTI de type 1 est alors calculée à partir de la matrice W du code DTI de type 0 en supprimant l'équation de poids impair. On prouve alors que ce code peut corriger au maximum [J-1]/2 erreurs [13].

Le code DTI type 1 corrige donc le même nombre d'erreurs que le code DTI type 0. Cependant, le rendement est plus avantageux pour le code DTI type I ((k+1)/n) que pour le code DTI type 0 (k/n).

Pour des tailles petites, les codes DTI sont comparables aux codes BCH. Par exemple, le code BCH (63,39) permet de corriger 4 erreurs au maximum. Le code DTI équivalent est code (63,37) qui ne permet que de coder des informations de 37 bits soit deux bits d'information en moins. Cependant, le circuit de décodage du code DTI est beaucoup moins complexe que celui utilisé pour le code BCH [13]

II.1.4. Code Euclidean Geometry (EG)

II.1.4.1. Bref rappel sur la géométrie euclidienne

II.1.4.1.1. Définition de l'espace EG

On considère l'espace E des m-uplets $(a_0,\ldots,a_{m-1})$ dont les coefficients a_i appartiennent au corps de Galois $GF(2^s)$. On munit cet espace de deux lois + et × définies par :

$(a+b) = (a_0,\ldots,a_{m-1}) + (b_0,\ldots,b_{m-1}) = (a_0+b_0,\ldots,a_{m-1}+b_{m-1})$

$\beta\times a = \beta\times (a_0,\ldots,a_{m-1}) = (\beta\times a_0,\ldots,\beta\times a_{m-1})$

$E(+,\times)$ est un espace de dimension m appelé *Géométrie Euclidienne* sur $GF(2^s)$. On le note EG $(m,2^s)$.

II.1.4.1.2. Ligne dans EG(m,2^s).

Chaque m-uplets dans l'espace EG(m,2^s) est appelé point. L'origine est le m-uplets nul (0,…..,0).

Considérons un point A différent de l'origine. L'ensemble {βA / β∈ GF(2^s)} est une ligne dans EG(m,2^s). Cette ligne contient 2^s points. Par construction cette ligne passe par l'origine.

Si on considère maintenant deux points A et B linéairement indépendants alors l'ensemble {B+βA / β∈ GF(2^s)} constitue une ligne passant par B. Par définition les lignes {βA / β∈ GF(2^s)} et {B+βA / β∈ GF(2^s)} n'ont aucun point commun. On dit alors que les lignes sont parallèles dans EG(m,2^s). Il y ($2^{(m-1)s}$) –1 lignes parallèles à une ligne passant par l'origine.

De même si deux points A_0 et A_1 sont linéairement indépendants, alors les ensembles {B+β A_0 / β∈ GF(2^s)} et{B+β A_1 / β∈ GF(2^s)} n'ont que le point B en commun. On dit alors que B est le point d'intersection des deux lignes {B+β A_0 / β∈ GF(2^s)} et {B+β A_1 / β∈ GF(2^s)}. On peut alors démontrer qu'il y a (2^{ms}-1)/(2^s+1) lignes qui se coupent en un point B.

On va utiliser cette dernière propriété pour générer un code à logique majoritaire noté code EG d'ordre s.

II.1.4.2. Définition du code EG d'ordre s.

II.1.4.2.1. Définition du vecteur d'incidence.

Soit un entier *n* égal 2^{ms}-1 où m et s sont des entiers quelconques. Soit V un n-uplets sur GF(2) noté (v_0,…,v_{n-1}). Comme pour les codes DTI, il est possible de faire correspondre la position du coefficient v_i avec la $i^{ème}$ puissance de α un élément primitif de GF(2^{ms}).

On peut aussi considérer que $GF(2^{ms})$ est un espace $EG(m,2^s)$. En prenant une ligne L ne passant pas par l'origine, il est possible de créer un vecteur V dans $GF(2^{ms})$ tel que :

- la $i^{ème}$ composante v_i est à « 1 » si α^i appartient à la ligne L,
- la $i^{ème}$ composante v_i est à « 0 » si α^i n'appartient pas à la ligne L.

Le vecteur V est appelé alors *vecteur d'incidence* de L.

II.1.4.2.2. Exemple

On considère le corps de Galois $GF(2^4)$ avec m égal à 2. α est un élément primitif de $GF(2^4)$. Posons $\beta = \alpha^5$. β^3 est égal à α^{15}. On peut alors facilement vérifier que l'ensemble $\{0, \beta^0 = 1, \beta^1 = \alpha^5, \beta^2 = \alpha^{10}\}$ forme un espace de 4 éléments $GF(2^2)$. Par conséquent, $GF(2^2)$ est un sous espace de $GF(2^4)$. La figure II-4 montre que tous les éléments de $GF(2^4)$ peuvent s'exprimer comme une combinaison des éléments de $GF(2^2)$, c'est à dire que $GF(2^4) = \{a+b.\alpha / (a,b) \in GF(2^2)^2\}$.

corps de Galois généré par le polynôme $1+x+x^4$

representation puissance	representation polynômiale	representation 4-uplets	representation EG(2,4)
0	0	0 0 0 0	0 0
1	1	1 0 0 0	1 0
α^1	α	0 1 0 0	0 1
α^2	α^2	0 0 1 0	β 1
α^3	α^3	0 0 0 1	β β^2
α^4	1 α	1 1 0 0	1 1
α^5	α α^2	0 1 1 0	β 0
α^6	α^2 α^3	0 0 1 1	0 β
α^7	1 α α^3	1 1 0 1	β^2 β
α^8	α^2	0 0 1 0	β^2 1
α^9	α α^3	0 1 0 0	β β
α^{10}	1 α α^2	1 1 1 0	β^2 0
α^{11}	α α^2 α^3	0 1 1 1	0 β^2
α^{12}	1 α α^2 α^3	1 1 1 1	1 β^2
α^{13}	1 α^2 α^3	1 0 1 1	1 β
α^{14}	1 α^3	1 0 0 1	β^2 β^2

Fig II-4. Représentation de $GF(2^4)$ en fonction des éléments de $GF(2^2)$

On peut alors en déduire que les points $\{\alpha^{14}+0.\alpha, \alpha^{14}+1.\alpha, \alpha^{14}+\beta.\alpha, \alpha^{14}+\beta^2.\alpha\}$ c'est à dire l'ensemble $\{\alpha^7,\alpha^8,\alpha^{10},\alpha^{14}\}$ forme une ligne dans EG(2,4).

Les autres lignes passant par α^{14} sont : $\{0,\alpha^4,\alpha^9,\alpha^{14}\}$,$\{\alpha^1,\alpha^5,\alpha^{13},\alpha^{14}\}$, $\{\alpha^0,\alpha^2,\alpha^6,\alpha^{14}\}$, $\{\alpha^3,\alpha^{11},\alpha^{12},\alpha^{14}\}$. Les vecteurs incidents des lignes ne passant pas par l'origine sont regroupés à la figure II-5.

puissance de α^i

	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
V_{l1}	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1
V_{l2}	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1
V_{l3}	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1
V_{l4}	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1

Fig II-5. Vecteur d'incidence des lignes passant par α^{14} dans EG (2,4)

On va utiliser ces vecteurs d'incidences pour donner une définition des codes EG d'ordre s.

II.1.4.2.3. Les codes EG

Le code EG d'ordre s et de longueur 2^{ms}-1 est le code cyclique dont l'espace dual contient tout les vecteurs incidents des lignes de EG(m,2^s) ne passant pas par l'origine.

On va maintenant donner une méthode pratique du calcul du polynôme générateur d'un code EG d'ordre s.

II.1.4.3. Construction du polynôme générateur

II.1.4.3.1. Définition de la fonction W_2^s

Soit η un entier inférieur à 2^{ms}. Cet entier peut s'exprimer comme une combinaison des puissances de (2^s) (i.e. $\eta = \sum_{i=0}^{m-1} \eta_i \left(2^s\right)^i$). Le poids de η en base 2^s, noté W_2^s, est défini par la relation : $W_2^s(\eta) = \sum_{i=0}^{i=m-1} \eta_i$ (2.9).

II.1.4.3.2. Construction du code générateur

La fonction de poids $W_2{}^s$ est utilisée dans le théorème suivant qui donne les conditions pour le calcul des racines du polynôme générateur d'un code EG d'ordre s :

Soient α et η respectivement un élément primitif de $GF(2^{ms})$ et un entier inférieur à $(2^{ms}-1)$. (α^{η}) est racine du polynôme générateur du code EG d'ordre s si et seulement si :

$$0 < \max_{0 \le L < s} W_{2^s}(\eta^{(L)}) \le (m-1)(2^s-1) \quad (2.10)$$

avec $\eta^{(L)}$ le reste de la division de $(2^L)\eta$ par $2^{ms}-1$.

On définit alors le polynôme générateur comme le plus petit commun multiple des polynômes minimaux des racines trouvées.

II.1.4.3.3. Exemple.

On pose m et s égaux à 2. Le corps de Galois $GF(2^4)$ peut alors être considéré comme une géométrie euclidienne EG(2,4) sur $GF(2^2)$. Soit α un élément primitif de $GF(2^4)$. D'après le théorème précédent, les racines α^{η} du polynôme générateur du code EG d'ordre 2 doivent vérifier la relation suivante :

$$0 < \max_{0 \le L < 2} W_{2^2}(\eta^{(L)}) \le 3$$

Le tableau de la figure II-6 donne les trois poids $W_{2^2}(\eta)$, $W_{2^2}(\eta^{(1)})$, $W_{2^2}(\eta^{(2)})$.

Les seuls entiers qui vérifient la relation précédente sont 1, 2, 3, 4, 6, 8, 9, 12. Par conséquent α, α^2, α^3, α^4, α^6, α^8, α^9, α^{12} sont les racines du polynôme générateur du code EG d'ordre 2. Comme α, α^2, α^4, α^8 sont conjugués, ils ont le même polynôme

minimal ϕ_1. Pour une raison identique, α^3, α^6, α^9, α^{12} ont le même polynôme minimal. Le polynôme générateur g(x) est alors égal à $\phi_1\phi_2$. D'où :

$$g(x) = \left(1 + x + x^4\right)\left(1 + x + x^2 + x^3 + x^4\right) = 1 + x^4 + x^6 + x^7 + x^8$$

Le code EG d'ordre 2 est un code cyclique C(15,7).

η	$h^{(0)}$	$h^{(1)}$	$h^{(2)}$	$W_2(h^{(0)})$	$W_2(h^{(1)})$	$W_2(h^{(2)})$	Max
0	0	0	0	0	0	0	0
1	1	2	4	1	2	1	2
2	2	4	8	2	1	2	2
3	3	6	12	3	3	3	3
4	4	8	1	1	2	1	2
5	5	10	5	2	4	2	4
6	6	12	9	3	3	3	3
7	7	14	13	4	5	4	5
8	8	1	2	2	1	2	2
9	9	3	6	3	3	3	3
10	10	5	10	4	2	4	4
11	11	7	14	5	4	5	5
12	12	9	3	3	3	3	3
13	13	11	7	4	5	4	5
14	14	13	11	5	4	5	5

Fig II-6. Calcul des poids W_2 des nombres de 1 à 15

II.1.4.4. Construction de la matrice W

D'après la définition du polynôme générateur, l'espace dual d'un code EG d'ordre s contient tous les vecteurs incidents des lignes ne passant pas par l'origine. Si on ne considère une ligne L passant par le point $\alpha^{2^{ms}-2}$. Il y a exactement $(2^{ms}-1)/(2^s-1)$ lignes passant par ce point. Tous les vecteurs incidents des lignes ne passant pas par l'origine sont orthogonaux à la position $\alpha^{2^{ms}-2}$. On en conclut que ces J vecteurs forment une matrice W.

On peut alors démontrer que ce code est capable de corriger $E\left[\frac{2^{ms}-1}{2(2^{s}-1)}-\frac{1}{2}\right]$ erreurs (E désignant la partie entière d'un nombre) [26].

Pour l'exemple précédent, le code peut corriger 2 erreurs au maximum. La matrice W est donnée à la figure II-5 du paragraphe II.1.4.2.2.

II.1.5. Conclusions

Tous les codes étudiés (DSCC, DTI, EG) sont des codes à logique majoritaire. Ils se caractérisent par un polynôme générateur G et une matrice W purement combinatoire. Les schémas de principes des circuits de codage (fig I-17) et de décodage (fig II-1) sont identiques pour tous ces codes. On a cependant vu dans la première partie que ces implantations avaient leurs limites en termes de débit, car la fonction de division polynomiale est une fonction complexe à implanter. Nous allons donc étudier de nouvelles architectures tant pour le codeur que le décodeur afin d'augmenter le débit de sortie de ces circuits.

II.2. Implantation du codeur

II.2.1. Choix de l'architecture

II.2.1.1. Architecture série

L'architecture série étudiée dans le chapitre I est l'architecture la plus souvent utilisée pour implanter des codes détecteurs/correcteurs d'erreurs car jusque à maintenant, son débit de sortie suffisait pour les applications. Pour rappel, elle est fondée sur le calcul série des restes partiels de la division d'un polynôme M de degré k par un polynôme générateur de degré r.

Deux circuits sont essentiellement utilisés pour effectuer ce calcul. Le premier circuit (fig II-7a) impose une prémultiplication de la donnée par x^r. Par contre, le circuit de la figure II-7b n'a pas cette contrainte et le calcul du reste se fait alors en (k) cycles au lieu de (k+r), soit un gain de (r) cycles ce qui correspond au degré du polynôme générateur. Par exemple, pour le code DSCC(72,44), le calcul du reste nécessite 72 cycles de traitement pour la première méthode et seulement 44 cycles avec le deuxième circuit. De plus, la deuxième méthode n'utilise pas plus de surface que la première méthode. Cependant, le débit peut diminuer légèrement car dans le deuxième circuit, le fan-out de la dernière porte Xor doit être important (feedback).

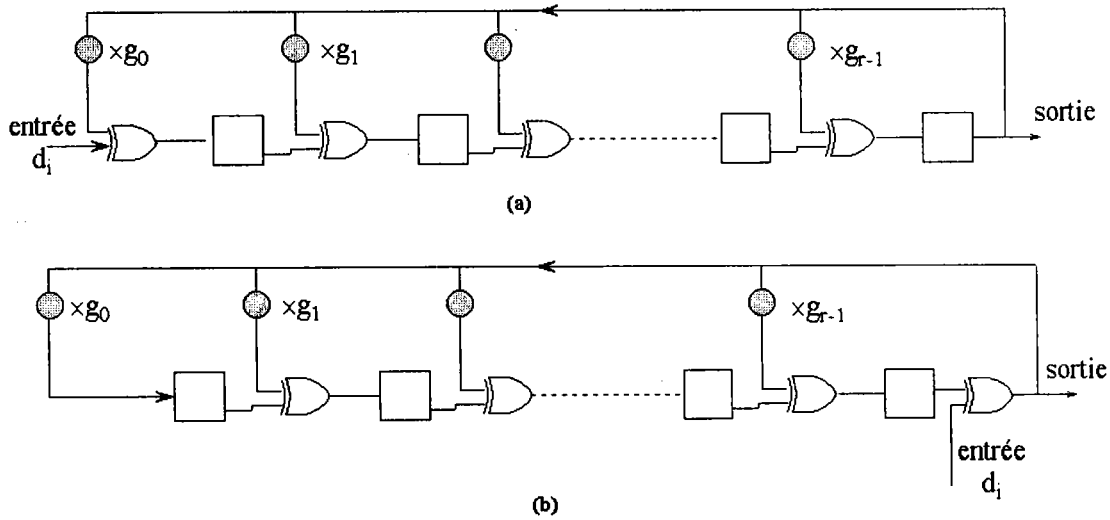


Fig II-7. Architecture série du codeur

II.2.1.2. Architecture systolique [17]

II.2.1.2.1 Présentation

L'architecture systolique permet d'effectuer le calcul du reste de la division de deux polynômes en utilisant qu'un seul élément de base dupliqué selon le degré du polynôme diviseur. Nous allons utiliser un exemple de division pour expliquer le fonctionnement de cette architecture et la construction de l'élément de base.

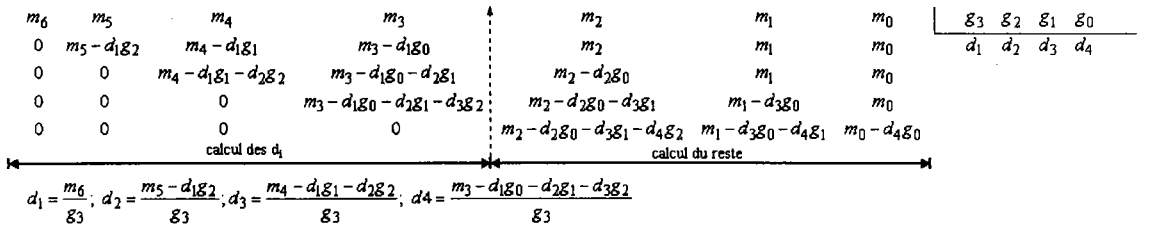


Fig II-8. Exemple formel d'une division polynomiale

La figure (II-8) est un exemple de division d'un polynôme de degré 6 [m_0..m_6] par un polynôme de degré 3 [g_0..g_3]. Dans cette division, deux opérations doivent être réalisées :

- le calcul des coefficients d_i,
- le calcul du reste.

En examinant la figure II- 8, deux remarques peuvent être faites :

1- les termes d_i sont égaux aux termes de plus haut degré des différents restes partiels de la division. Par exemple, d_1 et d_2 sont respectivement égaux à m_6/g_3 et $(m_5 - d_1g_2)/g_3$,

2- les termes consécutifs d'une colonne sont reliés entre eux par la récurrence suivante:

$$\begin{cases} U_n = m_k + d_i g_j \\ U_{n+1} = U_n + d_{i+1} g_{j+1} \end{cases} \quad (2.11)$$

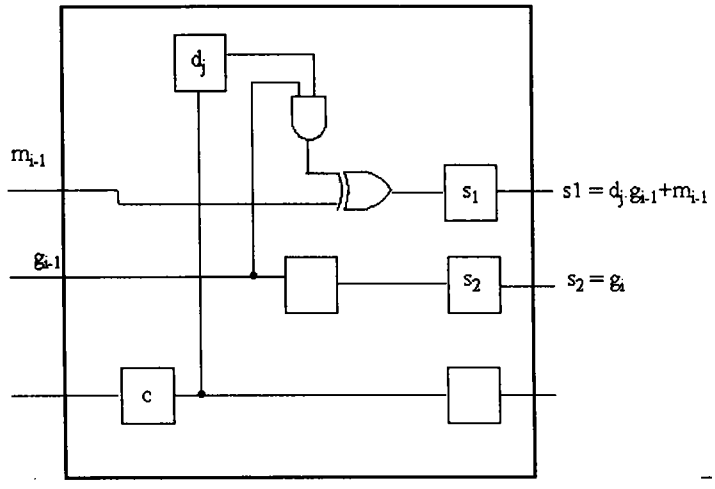


Fig II-9. Circuit d'implantation de la formule (2.11)

La formule (2.11) peut être implantée par la cellule de la figure (II-9). Par conséquent, en mettant côte à côte ces cellules, il est possible de calculer les différents termes de la figure II-8.

Le problème reste alors le cas du calcul des termes d_i. Or, dans le cas des codes correcteurs, le terme de plus haut degré du diviseur g_r (g_3 dans l'exemple) est toujours égal à « 1 ». Par conséquent les termes d_i vérifient les formules :

$$\begin{cases} d_1 = m_{k-1} \\ d_i = m_{k-i} + \sum_{\substack{j=1 \\ \delta_{r-i,j}=0 \text{ si } r-i+j<0}}^{j=i-1} \delta_{r-i,j} d_j g_{r-i+j} \quad \forall i \neq 1 \end{cases} \quad (2.12)$$

En modifiant légèrement le circuit de la figure II-9, il est alors possible de calculer à la fois les termes d_i et les autres coefficients de la division :

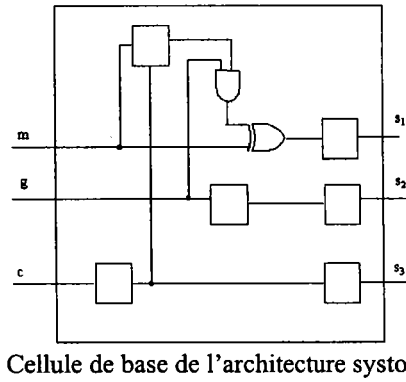


Fig II-10. Cellule de base de l'architecture systolique

En effet quand C est à un, le coefficient m entrant sera mémorisé au coup d'horloge suivant à la place de d_i.(fig II-11 a) Dans le cas du premier terme, c'est m_{k-1} (m_6 dans notre exemple) qui sera mémorisé (fig II-11 b). Ensuite C sera mis à zéro et les termes m_i et g_i vont être les entrées de ce circuit. Dans ce cas, les sorties du circuit respecteront la formule de récurrence (2.12) (voir figure II-11 c et d).

Supposons qu'il y ait une deuxième cellule connectée aux sorties de la cellule précédente. A l'instant 2T, la sortie de la première cellule est égale à $m_5+d_1g_2$. Au même instant, l'entrée C de la deuxième cellule est égale à 1. On se retrouve donc dans

le cas de la figure II-11 a mais avec une entrée m différente. La valeur $m_5+d_1g_2$ sera donc mémorisée dans la cellule 2 à la place d_j. Or d'après la figure II-8, on $m_5+d_1g_2$ qui est équivalent à d_2. On en déduit donc que d_2 sera mémorisé à t = 3T.

Cependant, la première cellule continue de traiter les données m et g entrantes et à l'instant 3T, la valeur $m_4+d_1g_1$ apparaît à sa sortie. A l'instant suivant, la valeur sortante de la deuxième cellule sera alors de $m_4+d_1g_1+d_2g_2$.

De ce fait, en utilisant plusieurs cellules successivement (fig II-13), il est possible de calculer de proche en proche l'ensemble des coefficients d_i et le reste de la division du polynôme M(x) de degré (k-1) par le polynôme g(x) de degré (r).

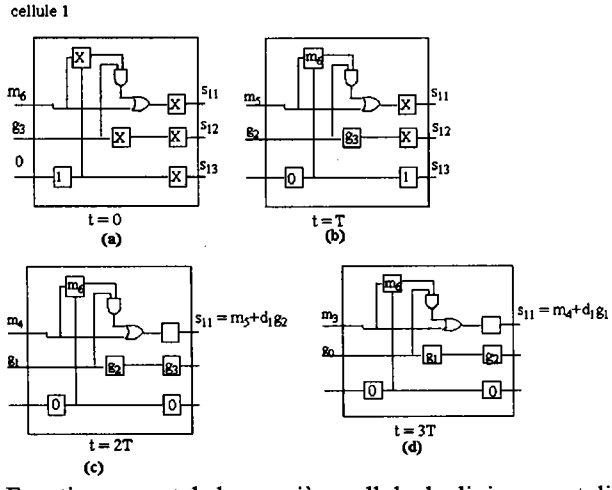


Fig II-11. Fonctionnement de la première cellule du diviseur systolique

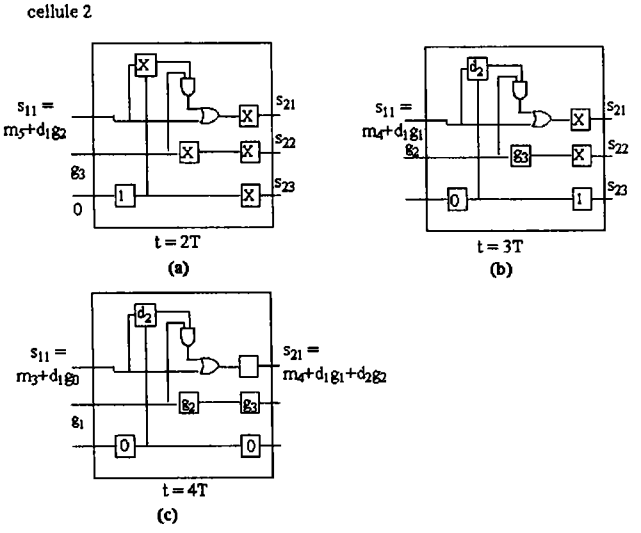


Fig II-12. Fonctionnement de la deuxième cellule

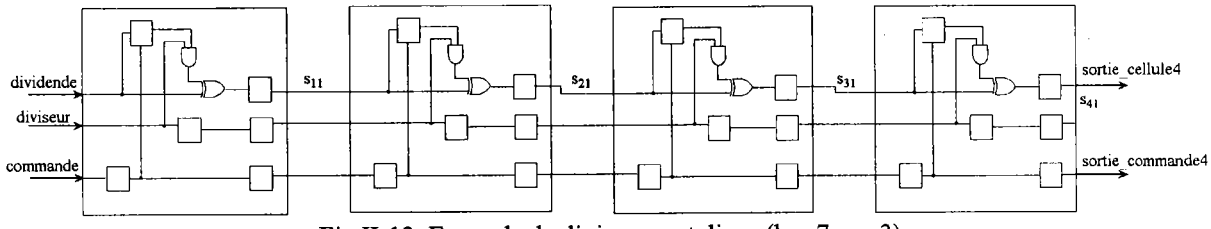


Fig II-13. Exemple de diviseur systolique (k = 7, r = 3)

II.2.1.2.2 Avantages et inconvénients

Les avantages de cette architecture sont nombreux :

1- Une seule cellule de base est nécessaire.

2- Le calcul du reste de la division d'un polynôme de degré (k-1) par un polynôme de degré (r) nécessite uniquement (k-r) cellules en série. Pour un couple (k, r) différent, il suffit de rajouter le nombre adéquat de cellules à la synthèse.

3- L'architecture ne dépend pas du polynôme diviseur g(x) mais uniquement du couple (k,r). On peut ainsi calculer les restes de différentes divisions à la volée (fig II-14).

4- Le diviseur est capable de traiter des flots continus de données. Le délai entre l'apparition de deux restes est alors de (k-r) cycles d'horloges.

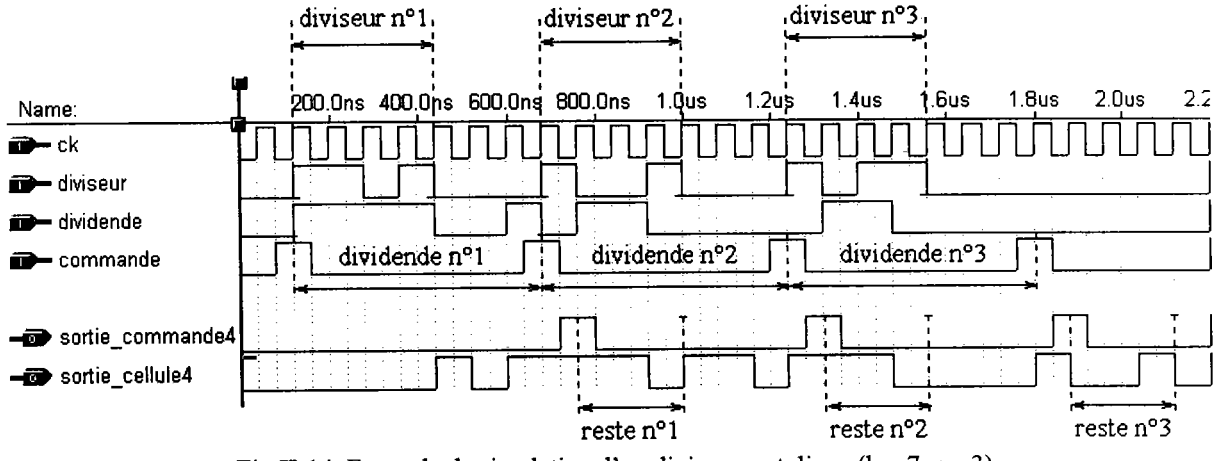


Fig II-14. Exemple de simulation d'un diviseur systolique (k = 7, r = 3)

Il existe cependant trois inconvénients à cette architecture :

1- Il est obligatoire (pour les codes correcteurs) d'effectuer au préalable la multiplication de la donnée par x^r afin de rajouter les r zéros nécessaires au déroulement de l'algorithme.

2- Le débit de l'architecture systolique est quasiment identique à celui de l'architecture série.

3- La surface occupée par l'architecture systolique est plus importante que celle occupée par l'architecture série. Le tableau V montre une comparaison des surfaces respectives des architectures systolique et série pour un couple (k,r) fixe. La dernière ligne donne les surfaces occupées sur un FPGA de la famille des FLEX10k d'Altera (EPF10K10LC84) par les architectures systolique et série pour un couple (k,r) égal à (7,4). Dans le cas de l'architecture série, le polynôme choisi pour la comparaison est le polynôme $1+x+x^2+x^3$ (cas le plus défavorable car $N_1 = r$)

(k,r)	architecture systolique	architecture série
nb Flip-flop	**6(k-r)**	**r**
nb XOR	**(k-r)**	**$N_1 \leq r$**
nb AND	**(k-r)**	**0**
surface (epf10k10lc84) (k,r) = (7,4)	**18 LCE indépendant du polynôme générateur**	**6 LCE $g(x) = 1+x+x^2+x^3$ $N_1 = 3$**

Tableau 5. comparaison des surfaces des architectures série et systolique

NB : Un bloc logique élémentaire (LCE) correspond à une table logique permettant d'implanter toute les fonctions combinatoire à 4 entrée, un registre et un multiplexeur permettant de contourner le registre

II.2.1.4. Architecture Parallèle

II.2.1.4.1. Architecture asynchrone ICC

Le système série (fig II 7-b) est uniquement constitué de porte XOR et de flip-flop. Ce circuit peut être décrit par l'équation suivante [27] :

$$[Y(t+1)] = \begin{bmatrix} r(t) \\ d(t) \end{bmatrix} = [S][Y(t)] = \begin{bmatrix} g_{r-1} & & \\ g_{r-2} & & \\ \vdots & I_{2r-1} & \\ g_0 & & \\ 0 & & \\ \vdots & & \\ 0 \ldots\ldots & .0 & \end{bmatrix} [Y(t)] \qquad (2.13)$$

avec r(t) le contenu des registres du restes à l'instant t, d(t) les entrées des données à l'instant t, $[g_{r-1},g_{r-2},..,g_0]$ le vecteur représentant le polynôme générateur de degré r et I_{2r-1} la matrice(2r-1)×(2r-1) unitaire.

Par récurrence, il est possible de connaître le reste à l'instant k en utilisant la formule :

$$[Y(k)] = \begin{bmatrix} r(k) \\ d(k) \end{bmatrix} = [S]^k[Y(0)] \quad (2.14)$$

D'un point de vue physique, la formule (2.13) peut être considérée comme l'équation d'une machine à état fini régulière. On peut alors démontrer que cette machine peut se mettre sous la forme d'un réseau de cellules (fig II-15) en cascades [28] (Iterative Combinatorial Circuit-ICC).

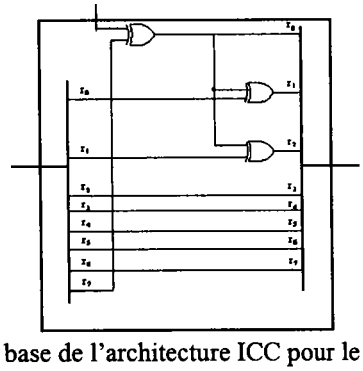


Fig II-15. Cellule de base de l'architecture ICC pour le polynôme $1+x+x^2+x^8$

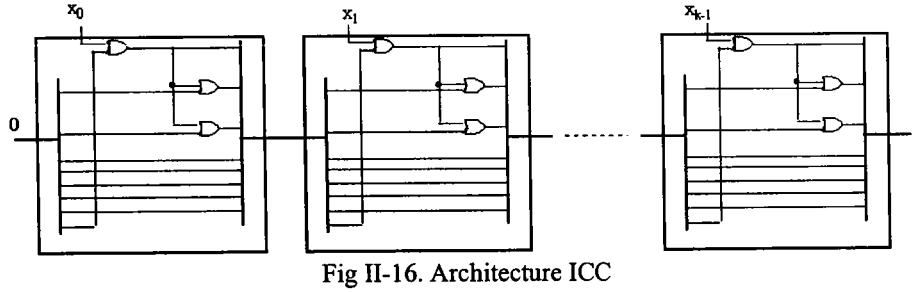


Fig II-16. Architecture ICC

Toutes les données étant appliquées aux cellules en même temps (fig II-16), les valeurs de sortie apparaissent simultanément (à part le délai de propagation de chaque cellule). Le calcul du reste est alors immédiat. Cette méthode a cependant une incidence très forte sur la surface utilisée par le codeur. Le tableau 6 donne la comparaison entre l'architecture série et totalement parallèle (ICC) pour deux codes.

	Code utilisé	n	k	circuit	série	parallèle	ratio
surface (1) débit (2)	DSCC(72,44)	72	44	EPF10K10LC84-3	37 101	237 459 (61 ns)	6,4 4,54
surface débit	EG	60	34	EPF10K10LC84-3	36 115	145 577 (47 ns)	4,02 5,01

(1) (2) La surface et le débit sont donnés respectivement en nombre de Cellule (LCE) et en Mbits/s pour un Flex10k10lc84-3

Tableau 6. Performances des architectures séries et ICC

On s'aperçoit que l'avantage de l'architecture ICC dépend du code utilisé. Par exemple, la surface du codeur pour le DSCC(72,44) est multipliée par 6,4 alors que le débit n'est multiplié que par 4.54. Pour le code EG considéré, le phénomène inverse se produit.

De plus, l'entrée des données à coder est constituée d'un bus de k lignes (respectivement 44 et 34 pour les codes servants d'exemple)

II.2.1.4.2. Architecture parallèle synchrone

II.2.1.4.2.1. Architecture de type 1

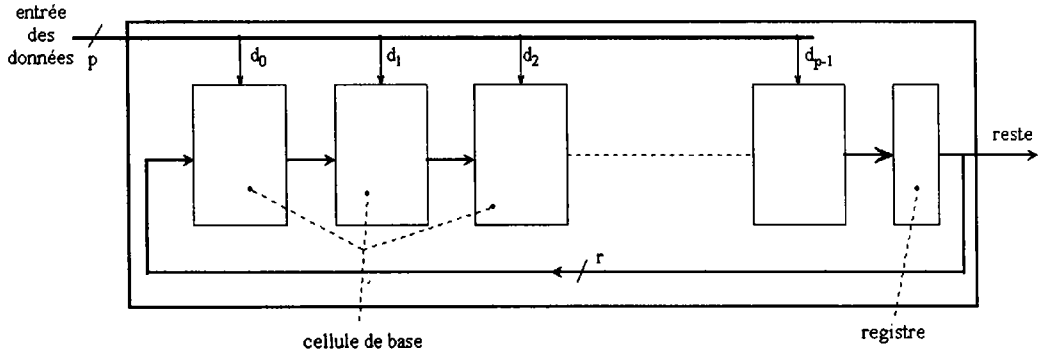


Fig II-17. Principe de l'architecture parallèle

soit p un diviseur de k (i.e. k = ap). L'équation (2.14) peut alors aussi s'écrire :

$$[Y(k)] = [S]^k [Y(0)] = \left([S]^p\right)^a [Y(0)] \quad (2.15)$$

Une méthode pour diminuer la surface occupée par le codeur consiste alors à coder non plus les k bits en une seule fois mais en utilisant des paquets de p bits. Le codeur est alors décrit par la formule suivante :

$$[Y(t+p)] = \begin{bmatrix} r(t+p) \\ d(t+p) \end{bmatrix} = [S]^p [Y(t)] \quad (2.16)$$

Matériellement, cette architecture peut être implantée avec p cellules ICC en cascades utilisées conjointement avec des registres (fig II 17). Le reste est alors calculé en (k/p) cycles. Les tableau 7 et 8 donnent des exemples des performances (débit et surface) d'un diviseur parallèle pour deux codes.

code utilisé	Architecture			
	Série	ICC	parallèle type1	
DTI(0,3,2)	21 LCE	154 LCE	p = 2 p = 23	24 LCE 92 LCE
DSCC(72,44)	37 LCE	237 LCE	p = 2 p = 4 p = 11 p = 11	46 LCE 54 LCE 74 LCE 125 LCE

Tableau 7. Surface d'un diviseur parallèle

code utilisé	Architecture				nb de cycles pour le calcul du reste		
	Série	ICC	parallèle type1		série	ICC	parallèle type1
DTI(0,3,2)	125 Mbits/s	85 ns	p = 2 p = 23	183,48 Mbits/s 525,09 Mbits/s	46	1	23 2
DSCC(72,44)	125 Mbits/s	61 ns	p = 2 p = 4 p = 11 p = 11	229,88 Mbits/s 335,32 Mbits/s 569,91 Mbit/s 740,74 Mbits/s	44	1	22 11 4 2

Tableau 8. Débit et temps de calcul du reste pour un diviseur parallèle

Etant basée sur l'architecture série (fig II-7), la prémultiplication par x^r est déjà effectuée. Par conséquent, il est nécessaire que le rapport (k/p) soit un entier. Cette condition limite alors le nombre de degré de parallélisme possible à implanter. Par exemple, seuls les degrés de parallélisme égaux à 2, 4, 11 ou 22 sont possibles pour le code DSCC(72,44). Cependant, en modifiant légèrement le circuit série de base, il est possible de contourner cette contrainte.

II.2.1.4.2.2. Architecture de type 2

Supposons que le degré de parallélisme p soit tel que (k/p) ne soit pas un entier. Il existe alors deux entiers tels que :

$$k = qp+\delta \quad (2.17)$$

Le mot de code C(x) peut alors s'écrire :

$$C(x) = x^r M(x)+R(x) = [x^{r-(p-\delta)}]x^{(p-\delta)}M(x)+R(x) = [x^{r-(p-\delta)}]M'(x)+R(x)$$

La donnée M'(x) est de taille (k+p-δ) et correspond à la donnée M(x) à laquelle on a rajouté (p-δ) zéros. Le calcul de mot code revient alors au calcul du reste de la division de $[x^{r-(p-\delta)}]M'(x)$ par le polynôme générateur g(x).

Deux opérations sont alors nécessaire :

1- rajouter les (p-δ) zéros au dernier paquets de bits de la donnée.

2- Multiplier la donnée par $[x^{r-(p-\delta)}]$ et non plus par x^r.

Cette dernière opération est rendue possible en utilisant conjointement un circuit de multiplication par H(x) (fig II-18 a) et un circuit de division par g(x) (fig II-18 b). Le circuit résultant (fig II-18 c) calcule alors le reste de la division de H(x)M(x) par g(x).

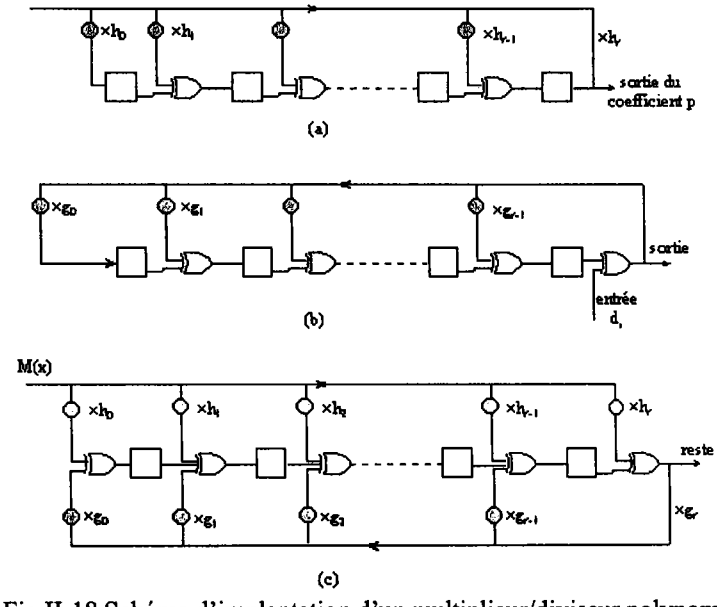


Fig II-18 Schéma d'implantation d'un multiplieur/diviseur polynomial

L'architecture parallèle de type 1 est alors toujours valable. Seule la cellule de base change et implante le circuit de la figure II-18 c qui vérifie l'équation :

$$R(t+1) = R(t)[0|Ir-1] \oplus [r \oplus h_r m(t)][G] \oplus m(t)[H] \quad (2.18)$$

Pour cette architecture, le degré de parallélisme doit alors être un diviseur de n (taille du mot code) et non plus un diviseur de k. Cette dernière contrainte permet un ensemble de degré de parallélisme plus important que dans le cas précédent. Pour le code DSCC(72,44), cet ensemble est égal à {2, 3, 4, 6, 8, 9, 12, 18, 36} au lieu d'être limité à {2, 4, 11, 22}.

II.2.1.4. Conclusions

Finalement, quatre architectures sont possibles pour implanter le diviseur polynomial nécessaire au circuit de codage. Ces quatre architectures peuvent être séparées en deux groupes (tableau 9) :

1- un groupe série (architecture série et systolique),

2- un groupe parallèle (architecture ICC et parallèle).

	groupe série		groupe parallèle	
	architecture			
	série	systolique	ICC	parallèle
surface	faible		elevée	Moyenne-elevé
débit	faible		elevé	Moyen-elevé
latence	k	(k-r)	1	(k/p) ou (k+p-δ)/p

(δ représente le reste de la division de k par p)

Tableau 9. Caractéristiques des architectures étudiées.

Dans le groupe série, le diviseur polynomial se caractérise par une surface faible. Cependant, le débit d'entrée des données est faible avec un temps de latence élevé. A l'opposé, le groupe parallèle est caractérisé par un débit élevé et une latence moins importante que dans le cas série. Cependant, la surface occupée par le diviseur polynomial est importante. De plus, celle ci dépend de façon importante du polynôme

générateur utilisé, du degré de parallélisme et des options de compilations choisis pour la synthèse.

Dans le cadre de cette thèse, nous allons nous focaliser sur la dernière architecture avec pour objectif de réaliser un codeur parallèle générique capable de traiter des flots continus de données.

II.2.2 Architecture parallèle du codeur

II.2.2.1. Le circuit de codage

II.2.2.1.1. Principe de fonctionnement du circuit de codage

Le circuit de codage doit être capable de traiter des flots continus de données. Cela impose que leur traitement soit réalisé en k cycles d'horloges (k le nombre de bits de données à coder). Comme les données arrivent en série, il est impossible de faire successivement la conversion série/parallèle, le codage et la conversion parallèle/série en un seul cycle de traitement (k cycles d'horloges). Cette limitation peut être supprimée en utilisant des mémoires internes. L'architecture choisie pour notre application utilise plusieurs mémoires fonctionnant en alternance (fig II-19 a et b).

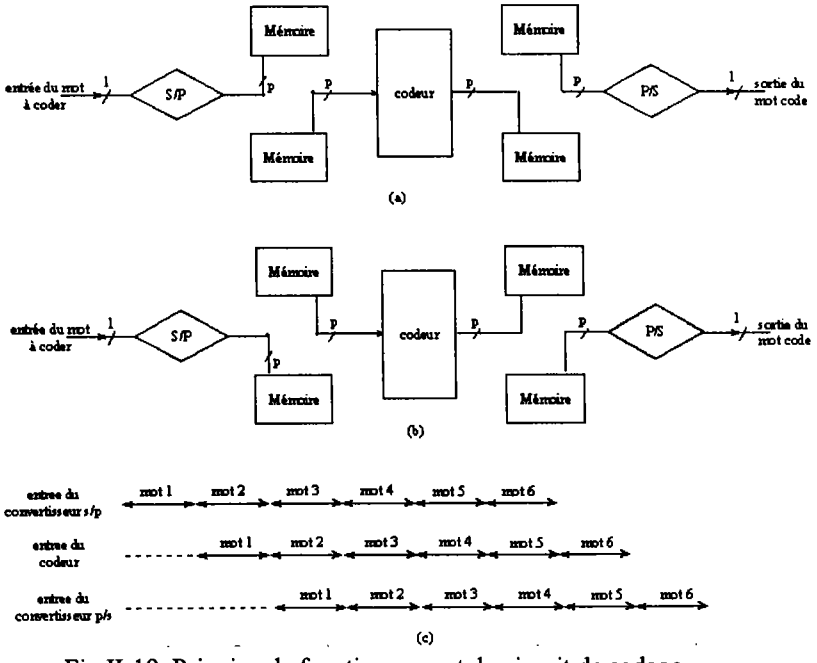


Fig II-19. Principe de fonctionnement du circuit de codage

Avec cette architecture, trois mots sont traités simultanément par le circuit de codage (fig II-18 c). L'avantage de cette architecture est que la latence du circuit ne dépend pas du parallélisme choisi mais uniquement de la taille de la donnée à coder. Cependant pour que le fonctionnement de cette architecture soit valable, deux conditions régissent les fréquences de fonctionnement des circuits :

Soient f_{in}, f_{out} respectivement la fréquence d'entrée du mot à coder et la fréquence de sortie du mot code. Pour respecter le diagramme (II-18 c), le temps mis pour convertir le mot code (conversion série/parallèle) doit être inférieur ou égal au nombre de cycles nécessaire à l'entrée du mot à coder (conversion parallèle/série), soit :

$$n(f_{in}) \leq k(f_{out}) \quad (2.21)$$

Pour la même raison, le temps mis pour effectuer le codage d'un mot doit être inférieur à k/f_{in}. Par conséquent, si f_f est la fréquence de fonctionnement du codeur, on a l'inégalité suivante :

$$(\lceil k/p \rceil + \lceil r/p \rceil + 3)(fin) \leq k(f_f) \quad (2.22)$$

avec :

- $\lceil k/p \rceil$: le nombre de cycles d'horloges (f_f) nécessaire au calcul du reste.
- $\lceil r/p \rceil$: le nombre de cycles d'horloge (f_f) utilisés pour remettre en forme le reste.
- 3 cycles d'horloges utilisés pour initialiser le circuit de codage

Ces contraintes sont d'ordre temporelles. Néanmoins, il ne faut pas négliger la surface occupée par le codeur (compromis surface/performance) directement liée au degré de parallélisme utilisé. Les différentes parties du circuit travaillant à des fréquences différentes (fonctions synchrones mutuellement asynchrones), les mémoires devront lire et écrire les données sur le bus à des vitesses différentes. Il sera donc nécessaire d'avoir recours à des interfaces asynchrones.

II.2.2.1.2. Vue générale du circuit de codage

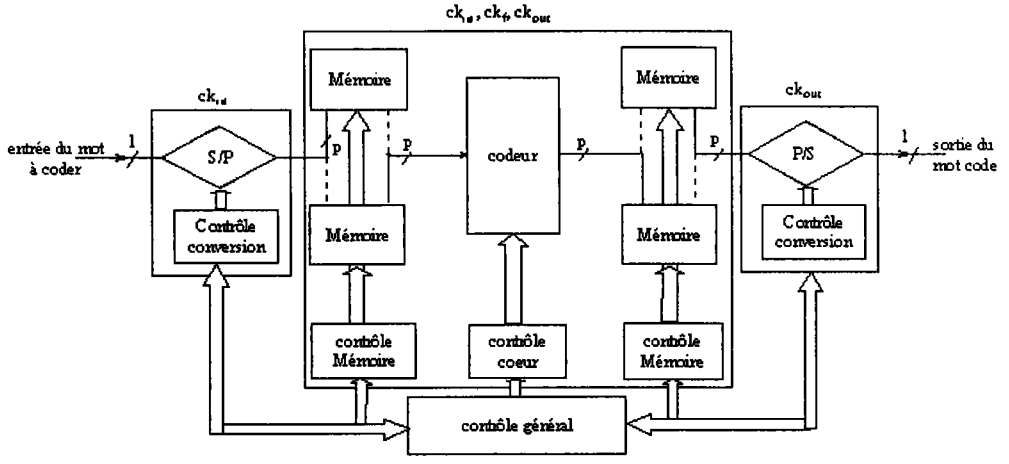


Fig II-20. Architecture générale du circuit de codage.

La figure II-20 donne une vue générale de l'architecture retenue pour le circuit de codage permettant de répondre aux contraintes demandées. Elle est constituée de 6 blocs :

1- un circuit de conversion série/parallèle qui a aussi pour charge de rajouter les éventuels (p-δ) zéros à la donnée à coder. Il fonctionne à la fréquence F_{in}.

2- deux mémoires d'entrées avec leur interface asynchrone qui permettent d'enregistrer les données entrantes.

3- un codeur implantant le code choisi (DSCC, EG, DTI) et remettant en forme le reste (suite de symboles de p bits). Il fonctionne à la fréquence F_f

4- deux mémoires de sortie utilisées pour enregistrer le mot code.

5- un circuit de conversion parallèle/série fonctionnant à la fréquence F_{out}.

6- un circuit de contrôle.

II.2.2.2. Mémoires

D'après les contraintes sur le circuit de codage, la mémoire doit lire et écrire les données sur le bus avec des vitesses différentes. Deux méthodes peuvent être utilisées pour implanter une mémoire répondant à cette contrainte.

II.2.2.2.1. Systèmes avec RAM

On peut utiliser une RAM avec des bus d'entrée et de sortie distincts (Macrofunction LPM_RAM_DQ). Comme la lecture et l'écriture se font à des fréquences différentes, il est nécessaire que deux circuits d'adressage (un pour la lecture, un pour l'écriture) soient utilisés. Le choix de l'adresse utilisée se fait alors via un multiplexeur (système est en mode lecture/écriture) (fig II-21).

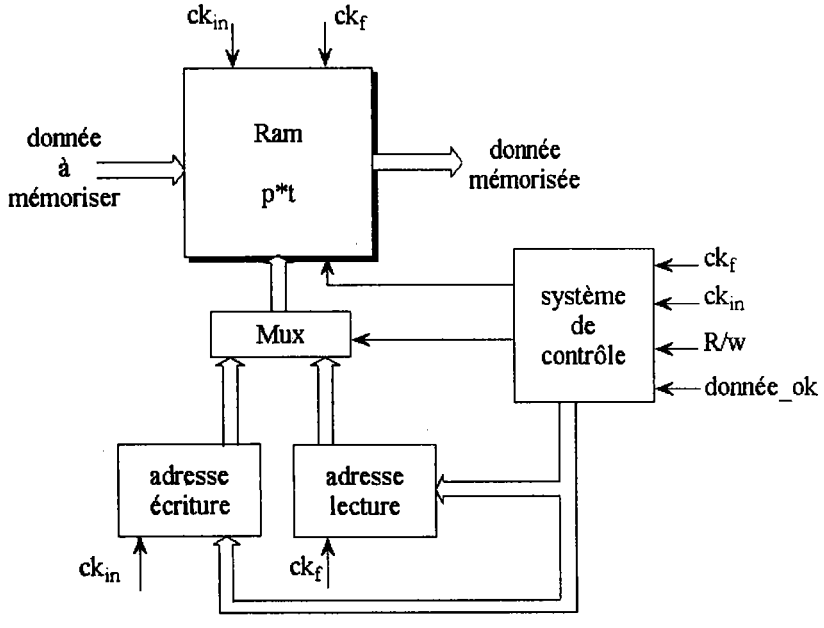


Fig II-21. Schéma de principe d'un système utilisant des RAM.

Il existe deux désavantages à cette méthode :

1- Les RAM ont des temps d'accès beaucoup trop long pour l'application visée. Sur le FLEX10K par exemple, l'accès aux données mémorisées est de l'ordre de 20 ns. De plus, les données à mémoriser ne remplissent pas toute la RAM (perte de surface).

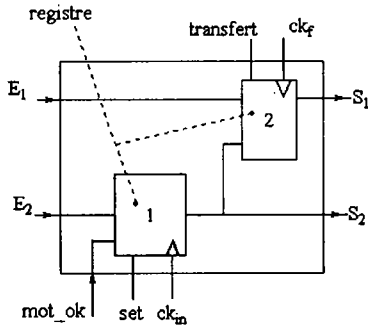
2- Il y a nécessité d'utiliser des compteurs et des multiplexeurs, ce qui va encore augmenter le temps nécessaire pour accéder aux données mémorisées.

Cependant, pour des codes nécessitant la mémorisation de données importante, cette solution est intéressante car la mémoire se caractérise surtout par une utilisation prédominante des portes logiques par rapport aux registres.

II.2.2.2.1. Systèmes avec Fifo

D'après l'algorithme de codage, la première donnée mémorisée est aussi la première utilisée par le codeur. On peut donc utiliser des FIFO (First In First Out) pour servir de mémoire. Aucun bloc d'adressage n'est alors nécessaire ce qui constitue un avantage par rapport au système avec RAM. Cependant, la lecture et l'écriture de la donnée ne se faisant pas à la même fréquence, la FIFO doit être asynchrone.

II.2.2.2.1.1. Cellule de base de la FIFO asynchrone d'entrée



registre 1

ck_{in}	mot_ok	set	E_2	S_2
↓	0	X	X	S_2
↓	1	0	0	0
↓	1	0	1	1
↓	X	1	X	S_7

registre 2

ck_{in}	transfert	E_1	S_1
↓	0	0	0
↓	0	1	1
↓	1	X	S_7

Fig II-22. Cellule de base de la FIFO asynchrone

Le schéma de la cellule utilisé pour générer la FIFO asynchrone se trouve à la figure II-22. Elle est constitué de deux registres travaillant à des fréquences différentes (ck_{in} et ck_f). Le contrôle de la FIFO permet alors quatre modes de fonctionnement de la cellule de base.

1- Mode de Mémorisation : la brochea « transfert » est forcée à zéro. La sortie S2 recopie alors l'entrée E2 quand « mot_ok » est à 1. La broche « set » permet de bloquer la cellule d'entrée (attente de la commande d'enregistrement). Ce mode est utilisé pendant la conversion série/parallèle.

2- Mode de blocage : quand toute la conversion est finie, le système est bloqué par le contrôle (set à 1, transfert à 0). Ce mode est utilisé pour permettre de caler les données sur la fréquence de sortie Ck_f.

3- Mode de transfert : Le premier registre de la cellule restant bloquée (set à 1), un ordre de transfert entre la cellule 1 et la cellule 2 est donnée (transfert à 1). La sortie de la cellule 2 recopie alors la sortie de la cellule 1.

4- Mode de sortie : « transfert » étant mis à 0, la cellule 2 est assimilable à un registre. La sortie de cellule 2 recopie l'entrée 1 de la cellule de base.

II.2.2.2.1.2. Cellule de base de la FIFO asynchrone de sortie

Au contraire des FIFO d'entrées, les FIFO de sortie mémorisent les données en continu (phase d'enregistrement du mot code) et écrivent les données sur le bus de sortie quand le convertisseur parallèle/série en fait la demande. Les registres de la cellule de base de la FIFO d'entrée sont réutilisés mais leur ordre est inversé de façon à permettre ce fonctionnement.

II.2.2.2.1.3. Génération de la FIFO asynchrone

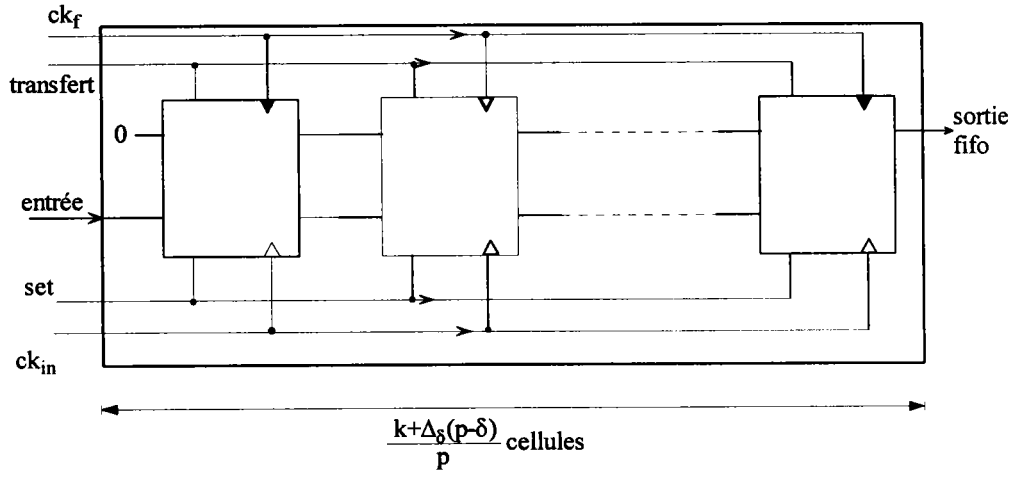


Fig II-23. Registre à décalage pour la FIFO asynchrone

Lors de la synthèse de la mémoire, il est nécessaire de calculer la profondeur de la FIFO qui sera utilisée. Deux cas sont possibles. Si k/p n'est pas entier, alors la profondeur de la FIFO sera égale à (k+p-δ)/p avec δ le reste de la division de k par p.

Dans le cas où k/p est entier, la profondeur sera égale à k/p. Finalement la profondeur de la FIFO suit la formule :

$$PF_{fifo} = \frac{k + \Delta_\delta (p - \delta)}{p} \quad (2.21)$$

avec Δ_δ qui vérifie les conditions :

$$\Delta_\delta = \begin{cases} 0 \text{ si } \delta = 0 \\ 1 \text{ si } \delta \neq 0 \end{cases} \quad (2.22)$$

La profondeur du registre à décalage étant connue, la FIFO est créée en dupliquant p fois ce registre (voir figure II-24). La figure II-25 donne un exemple de fonctionnement de la FIFO pour un code DSCC(72,44) avec un parallélisme de 12.

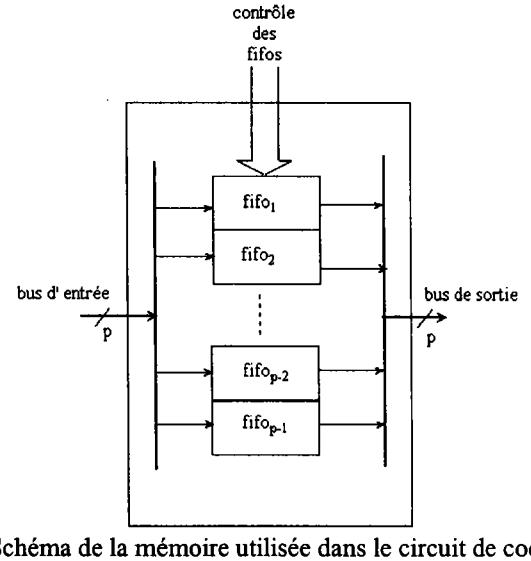


Fig II-24. Schéma de la mémoire utilisée dans le circuit de codage.

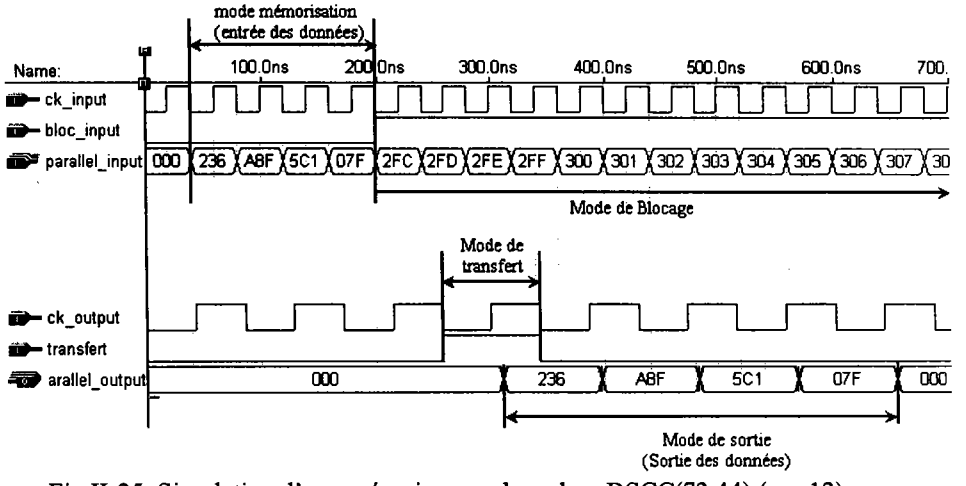


Fig II-25. Simulation d'une mémoire pour le codeur DSCC(72,44) (p = 12)

II.2.2.3. Conversion série/parallèle

Plusieurs contraintes régissent le convertisseur série/parallèle :

1- Il doit réaliser la conversion série/parallèle et maintenir la donnée sortante pour permettre de la mémoriser

2- Si le rapport (k/p) n'est pas entier, il doit rajouter les (p-δ) zéros nécessaires à l'algorithme de codage.

Pour réaliser la conversion série/parallèle, on utilise un registre à décalage. Comme la sortie de ce registre doit être maintenue (le temps que la FIFO la mémorise), il est nécessaire de permettre le blocage du convertisseur et d'envoyer un signal à la mémoire pour la prévenir que la donnée est prête. Comme la mémoire ne fonctionne pas à même vitesse que le convertisseur, deux registres doivent être utilisés en alternance (un est en position de conversion et l'autre en phase de blocage) afin de permettre une conversion d'un flot continu de données.

Cependant lors de conversion du dernier symbole (dernier bits du mot à coder), il est nécessaire de rajouter (p-δ) zéros (qui seront éliminés après codage). Cet ajout se fait après seulement δ cycles d'horloges. On utilise alors un troisième registre dans lequel l'entrée des données se fait à la $\delta^{ème}$ cellule (fig II-26).

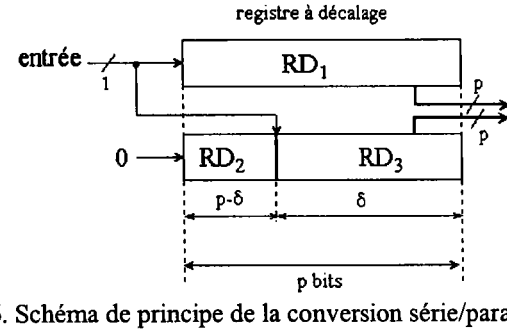


Fig II-26. Schéma de principe de la conversion série/parallèle

Conjointement aux trois registres cités précédemment, on utilise un multiplexeur bufférisé qui envoie la sortie du registre à décalage (alors en mode bloqué) vers les mémoires.

L'architecture globale retenue pour réaliser la conversion est représentée à la figure II-27.

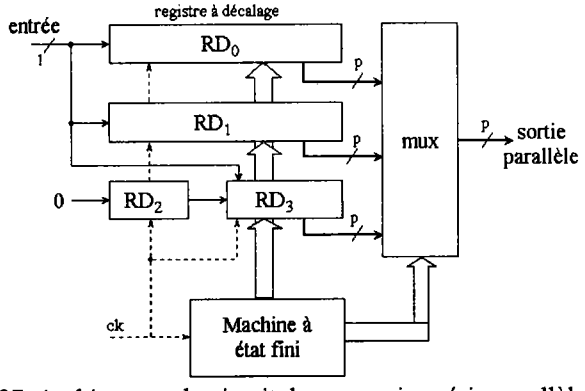


Fig II-27. Architecture du circuit de conversion série-parallèle

La figure II-28 (a) est un exemple de simulation d'une session de conversion pour le code DSCC(72,44) avec un parallélisme de 3. On a choisi ce degré pour mettre en évidence le rajout des zéros au mot à coder. En effet 72 est divisible par 3 mais pas 44 (44 = 14*3+2). Pendant 14 cycles ($\lfloor k/p \rfloor$) le circuit réalise donc la conversion série/parallèle habituelle en utilisant en alternance les deux registres à décalage de longueur p (broche set_i0). Pour le dernier cycle, on rajoute un zéro ($\delta = 1$) aux deux derniers bits de la donnée (fig II-28 b) et c'est le dernier registre qui est utilisé (broche set_i1). Le système revient alors au stade initial et une nouvelle donnée peut être convertie.

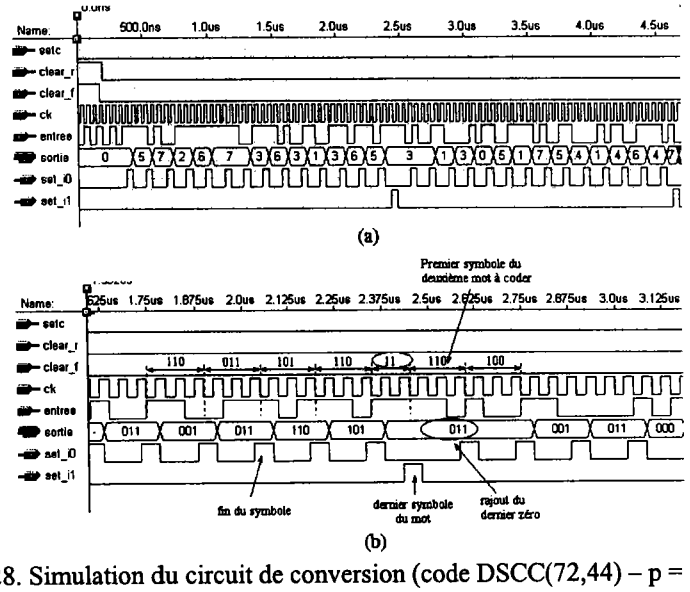
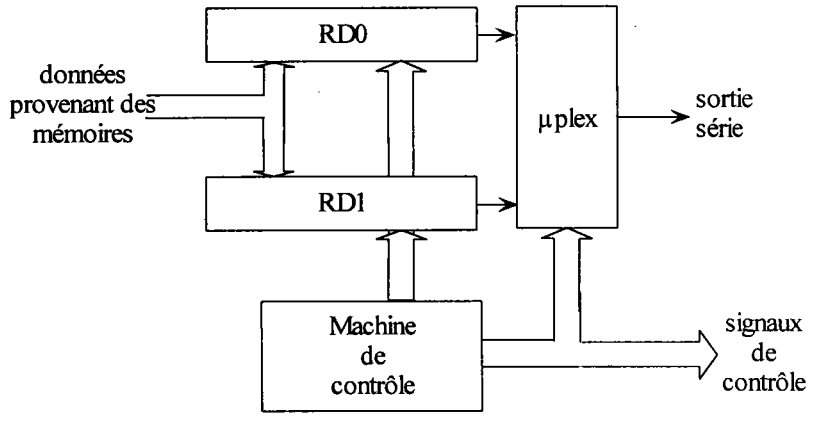


Fig II-28. Simulation du circuit de conversion (code DSCC(72,44) – p = 3)

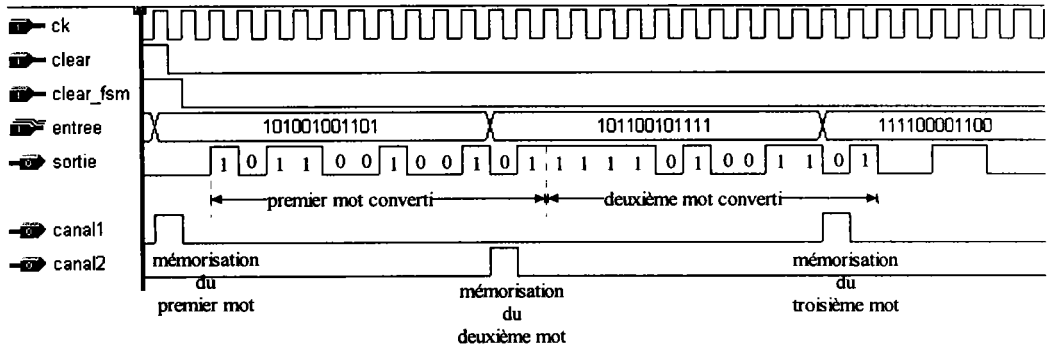
II.2.2.4. Conversion parallèle/série

Comme p est un diviseur de n, la conversion parallèle/série est beaucoup plus simple que la conversion série/parallèle (pas de rajout de bits). Cependant le registre a besoin d'un cycle d'horloge pour charger le symbole a convertir. Pour rendre possible le traitement de flot continu de données, deux registres à décalage sont donc utilisés en alternance (broche canal1 et canal2 de la figure II-29 b).

Comme précédemment, un multiplexeur bufférisé est utilisé pour aiguiller la donnée convertie vers la sortie du circuit de codage (fig II-29 a).



(a)



(b)

Fig II-29. Architecture et simulation du convertisseur série/parallèle

II.2.2.5. Codeur

Deux contraintes régissent le circuit de codage :

• Par construction, la donnée sortante du convertisseur série/parallèle est équivalente au message original auquel on a ajouté (p-δ) zéros. Pour être en accord avec l'algorithme de codage, le codeur doit donc effectuer la prémultiplication par $x^{(r-p+\delta)}$.

• La donnée entrante (de taille p) et le reste calculé (de taille r) sont envoyés vers deux mémoires (fig II-19). Comme le dernier symbole de la donnée contient (p-δ) zéros, ceux ci doivent être supprimés et remplacés par les premiers (p-δ) bits du reste. Un circuit de mise en forme de ces deux vecteurs (symbole modifié et reste tronqué) est alors nécessaire avant l'envoi des données vers la mémoire (fig II-30).

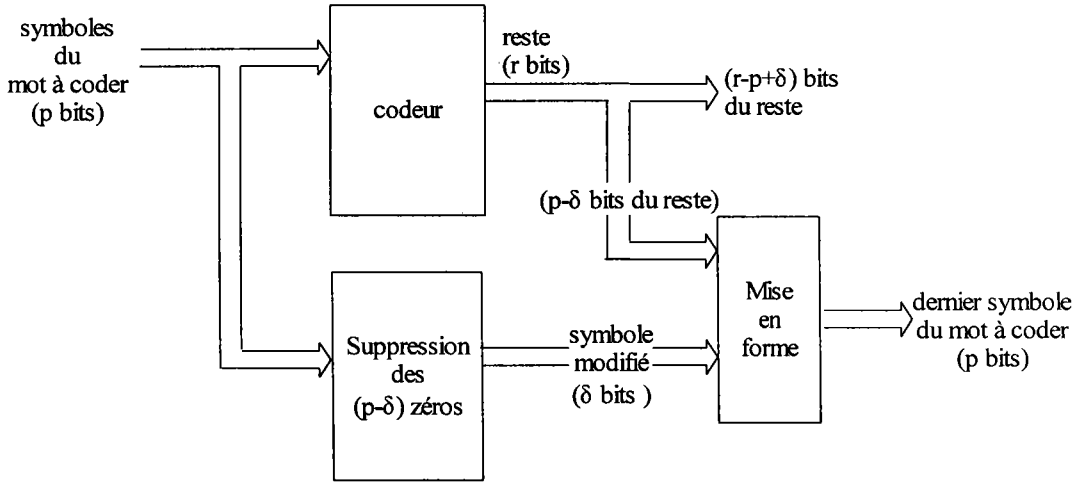


Fig II-30. Schéma de principe du codeur parallèle

II.2.2.5.1. Diviseur

L'architecture parallèle de type 2 (fig II-18) est la plus adaptée pour effectuer la prémultiplication et le calcul du reste. Dans cette architecture, deux paramètres sont considérés :

• Le vecteur (g) dont les coefficients décrivent le polynôme générateur utilisé

• Le vecteur (h) qui permet de réaliser la prémultiplication requise.

Par construction, le reste est liés aux vecteurs (g) et (h) par les relations :

$$\left\{ r_{i+1}^{j+1} = r_i^j \oplus ((r_{r-1}^j \oplus (d \times h_r)) \times g_{i+1}) \oplus (d \times h_{i+1}) \middle| i \in \{1,..;r-2\} \right\} \quad (2.23)$$

$$r_0^{j+1} = ((r_{r-1}^j \oplus (d \times h_r)) \times g_0) \oplus (d \times h_0) \qquad (2.24)$$

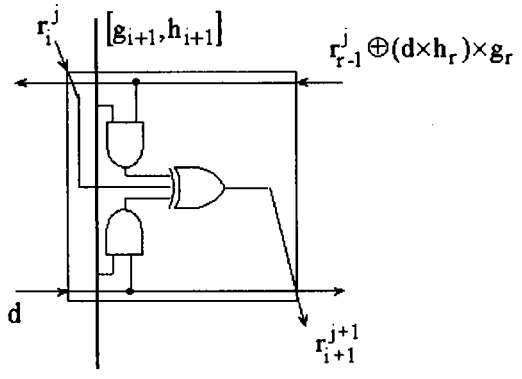


Fig II- 31. Cellule de base du diviseur série

Les deux formules (2.23) et (2.24) peuvent être implantées dans une seule cellule (fig II-31) constituée de deux portes AND et d'une porte XOR à trois entrées. Le diviseur est alors constitué de (r) cellules en série. La valeur $(r_{r-1}^j \oplus (d \times h_r) \times g_r)$ (signal de feedback) est alors égale à la sortie de la dernière cellule (fig II-32).

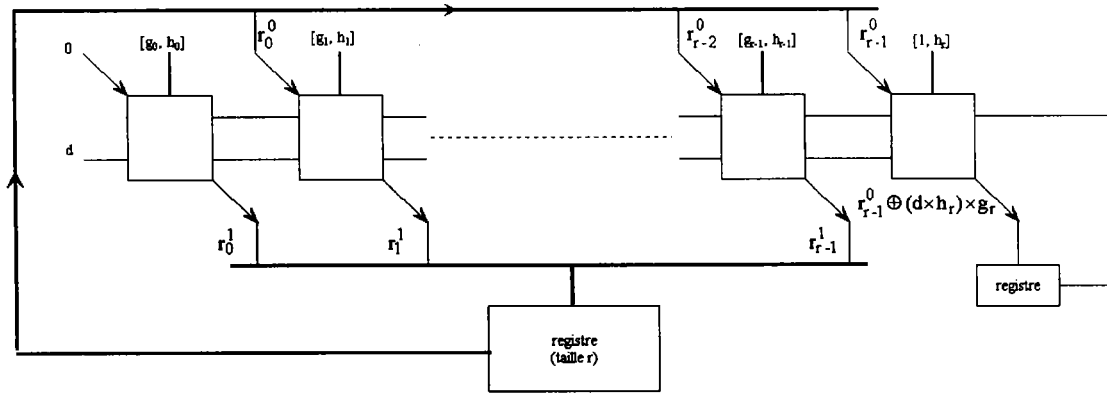


Fig II-32. Implantation du circuit série de division

L'architecture parallèle de type 2 est décrite par l'équation (2.16). D'un point de vue matériel, cette implantation consiste à mettre en série p diviseurs (récursivité sur j dans les formules 2.23 et 2.24). Un exemple d'implantation d'un diviseur parallèle de type 2 est représenté à la figure II-33.

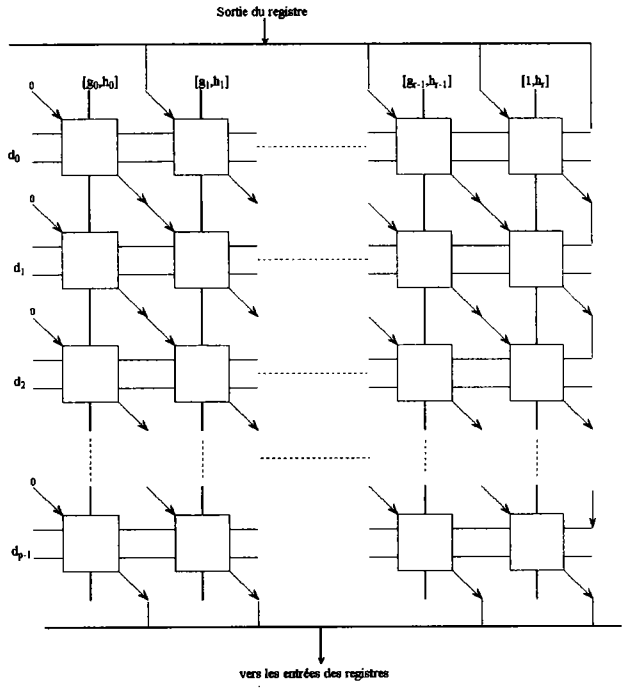


Fig II-33. Architecture du diviseur parallèle de type 2.

Cette architecture a été décrite en VHDL sur des circuits de la famille des FLEX10K d'Altera. La figure II-34 montre la simulation d'un diviseur parallèle (p =12) pour le code code DSCC(72,44). Dans cet exemple, on veut coder la donnée représentée par le polynôme P(x) :

$1+x^2+x^3+x^5+x^7+x^8+x^{12}+x^{13}+x^{15}+x^{19}+x^{23}+x^{24}+x^{25}+x^{26}+x^{27}+x^{29}+x^{31}+x^{32}+x^{36}+x^{37}+x^{39}+x^{43}$

Comme 44 n'est pas divisible par 12, il est nécessaire de rajouter p-δ (=4) zéros à la donnée (multiplication par x^4). Le nouveau polynôme peut alors se représenter en binaire par la séquence « 000011110101100011010001000111110101100011010001 ». Cette séquence est alors découpée en quatre symboles de 12 bits : « 8D1 », « 1F5 », « 8D1 », « 0F5 ».

Lors de l'entrée du dernier symbole, la sortie du diviseur est égale au reste de la division de $x^{28}P(x)$ par g(x), c'est à dire :

$$R(x) = 1+x^2+x^3+x^5+x^6+x^8+x^9+x^{13}+x^{14}+x^{16}+x^{19}+x^{20}+x^{21}+x^{23}+x^{25}$$

soit en hexadécimal :

$$R(x) = B6C69D4.$$

Ce reste est constitué de 28 bits soit 3 symboles (2 de 12 bits et 1 de 4 bits). Avant de mémoriser ce reste, il faut remplacer les quatre zéros rajoutés à la conversion série/parallèle par le dernier symbole du reste puis transformer les 24 bits restants en symboles de 12 bits. On va utiliser un circuit de mise en forme.

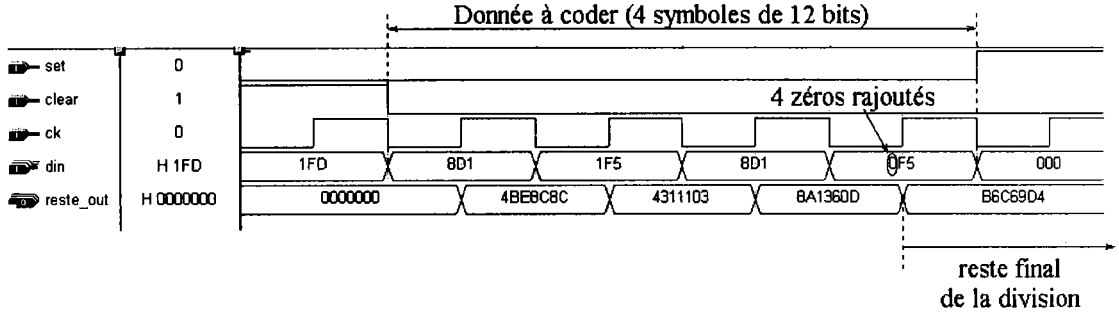


Fig II-34. Exemple de simulation du diviseur pour le code DSCC(42,44) avec un parallélisme de 12

II.2.2.5.2. Mise en Forme

Le reste étant constitué de r bits en parallèle, il est obligatoire d'utiliser un circuit annexe pour mettre en forme ce résultat (i.e. le reste est transformé en une suite de symbole de p bits). Cependant, ce circuit doit assurer deux autres fonctions.

1- pendant les $(\frac{k+p-\delta}{p}-1)$ premiers cycles (entrée des k-δ premiers bits de la donnée), le circuit de mise en forme doit être transparent (zone 1).

2- Lors de l'entrée du dernier symbole de la donnée (cycle $\frac{k+p-\delta}{p}$), celui doit être mémorisé et combiné avec les (p-δ) premiers bits du reste (remplacement des zéros) (zone 2).

3- Les (r-p+δ) bits restants du reste sont mis en forme (suite de symboles de p bits) (zone 3).

L'architecture choisie pour implanter le circuit effectuant ces opérations est représentée à la figure II-35.

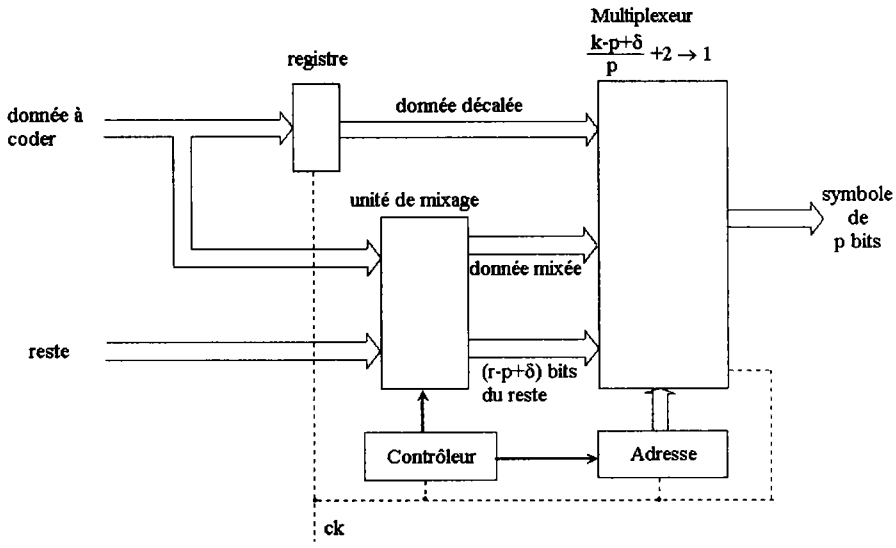


Fig II-35. Circuit de mise en forme du mot code

Ce circuit de mise en forme est essentiellement composé de trois sous-circuits.

• Une unité de mixage : elle a pour but de mélanger le dernier symbole du mot à coder et les (p-δ) premiers bits du reste (« donnée mixée »). La deuxième sortie de ce circuit est alors composée des (r-p+δ) derniers bits du reste.

• Un multiplexeur de symboles commandé par adresse : selon l'adresse sélectionnée, le circuit se trouvera dans une des trois zones définies précédemment.

• Un contrôleur : Il va commander les unités de mixages et l'adresse utilisée par le multiplexeur.

II.2.2.5.3. Cœur du codeur

Finalement, le circuit final de codage est celui de la figure II-36.

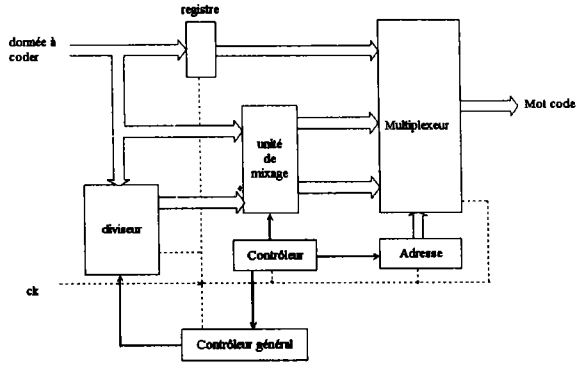


Fig II-36. Schéma du cœur du codeur

La figure II-37 montre la simulation du cœur du codeur pour le code DSCC(72,44) avec un parallélisme de12. Le mot à coder est identique à celui utilisé pour la simulation du paragraphe II.2.2.5.1. , c'est à dire qu'il est constitué de 4 symboles de 12 bits (« 8D1 », « 1F5 », « 8D1 », « 0F5 »). Le symbole « 0 » dans le dernier bit représente les quatre zéros rajoutés à la donnée. La redondance à rajouter à la donnée est égale à la séquence hexadécimale suivante : B6C69D4.

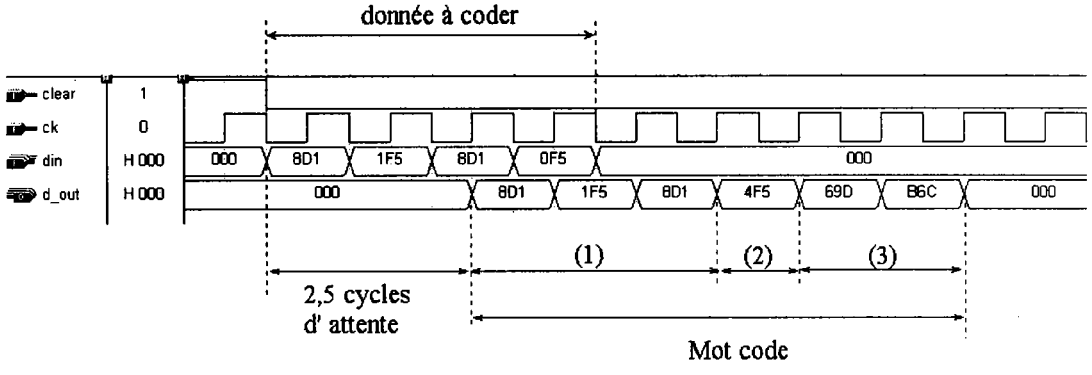


Fig II-37. Exemple de simulation du cœur du codeur pour le code DSCC (72,44)

Après 2,5 cycles d'attente (latence) correspondant au passage dans les différents registres du circuit de mise en forme, les trois premiers symboles de la donnée sortent du circuit de codage (1). Le dernier symbole du mot à coder est alors combiné avec le reste de la division (les quatre zéros sont remplacés par les 4 premiers bits du reste) (2). Enfin, les 24 bits restants sont transformés en symboles de 12 bits et envoyés vers la sortie du circuit (3). Le calcul du mot code nécessite alors $(\frac{n}{p}+2,5)$ cycles d'horloge ce qui respecte les contraintes de la formule (2.20).

II.2.3 Conclusions

Après une étude des différentes architectures possibles pour le diviseur, l'architecture parallèle a été retenue. On a montré que cette architecture permet de coder des flots continus de données grâce à l'utilisation de mémoires internes. Cependant, cette

architecture n'est probablement pas utilisable pour des codes possédant des polynômes générateur de haut degré comme le code DSCC(1057,813) (le retrobouclage se fait sur les bits du reste qui pour ce dernier code est un bus de 244 lignes).

Après la validation de l'architecture du codeur, nous allons réaliser la même étude pour le décodeur qui en plus de la division parallèle doit implanter les circuits de localisation et d'évaluation des erreurs.

II.3. Implantation du décodeur

II.3.1. Choix de l'architecture.

II.3.1.1. Architecture série.

L'architecture série (décodeur de Meggit), décrite dans le premier chapitre, est généralement celle utilisée pour les codes à logiques majoritaire [7],[8]. Le circuit de décodage est alors constitué de trois composants (fig II-38) :

- Un diviseur polynomial (syndrome)
- Une matrice W caractéristique du code (Composite Parity Check Sum Generator – CPCS)
- Une Porte à logique majoritaire (LMG).

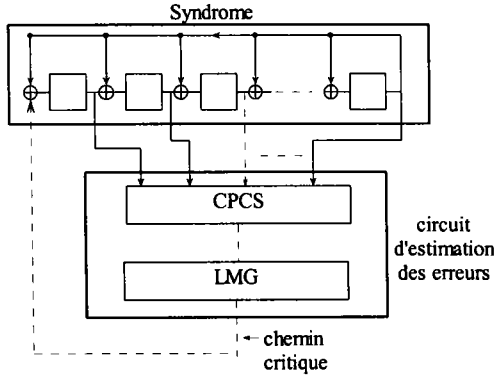


Fig II-38. Circuit conventionnel d'un décodeur à logique majoritaire

Dans cette architecture, le retro-bouclage (feedback) de l'erreur estimée constitue le chemin critique. Pour les codes de taille importantes, le nombre de portes logiques

utilisées dans le circuit d'estimation des erreurs rend difficile l'implantation d'un décodeur très rapide. Dans [29], la faisabilité d'une implantation pipelinée d'un décodeur à logique majoritaire a été étudiée.

II.3.1.2. Architecture en pipeline.

Dans cette étude, on suppose que le circuit d'estimation des erreurs (EEC) est composé de trois circuits (fig II-39) :

Un circuit implantant la matrice W (CPCS) et constitué de m_1 étages

Un compteur de population (compteur de 1) réalisé en m_2 étages

Un circuit de décision qui compare le nombre 1 avec un seuil T (m_3 étages)

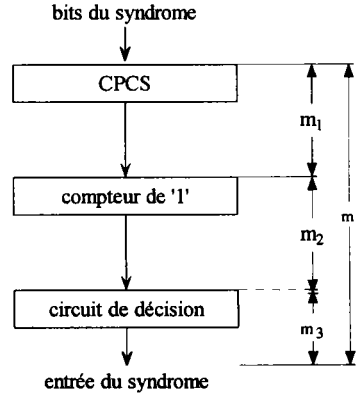


Fig II-39. Organisation temporelle du décodeur pipeliné

Le circuit d'estimation comporte donc au total m étages ($m_1+m_2+m_3$). La génération d'un signal d'erreur est alors décalée de m cycles d'horloges par rapport au circuit série d'origine. Pour compenser ce retard, le bit d'erreur ne rentre plus à la position zéro du syndrome (schéma classique de la figure II-40a) mais à la position m de celui ci (fig II-40b).

Cependant, il faut tenir compte du rebouclage car à l'instant t_0 le dernier bit du syndrome est injecté dans le syndrome (fig II-40 b). Comme l'erreur n'est connue que m cycles d'horloges plus tard, le syndrome peut contenir des symboles erronés (fig II-

40c). Ce symbole erroné engendré à l'instant t_0 ne pourra alors être corrigé que m cycles plus tard (fig II-40d).

Comme ces symboles erronés modifient le syndrome, ils vont avoir un effet sur le circuit générant les équations majoritaires (CPCS). Cependant, le bit s_0 du syndrome n'apparaît que dans une et une seule équation W_i. Par conséquent, seule la valeur de cette équation est inversée.(fig II-40 c). La valeur de sortie du circuit CPCS varie alors de ±1 par rapport à la valeur de sortie d'origine du CPCS.

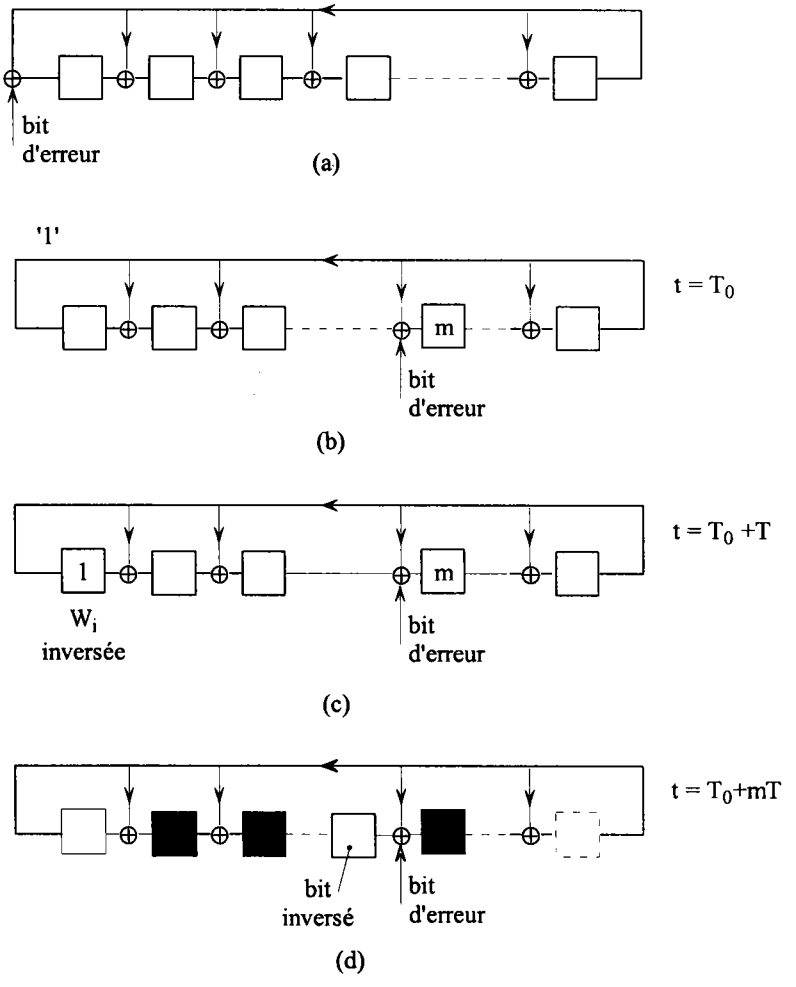


Fig II-40. Comportement du syndrome avec une architecture pipelinée

Dans le cas le plus défavorable, les m premiers bits du syndrome peuvent être occupés par des bits erronés. Comme certaines équations majoritaires peuvent contenir plusieurs bits erronés, le nombres d'équations inversées est au plus égal à m.

Comme la sortie du CPCS est changée, il faut modifier le seuil de décision (si le nombre de 1 en sortie du CPCS est supérieur au seuil, le bit examiné est considéré comme erroné) de façon synchrone avec la variation de la sortie du CPCS. Dans [30], un algorithme permettant de modifier ce seuil en fonction des nouvelles valeurs de CPCS a été étudié pour le code DSCC(1057,813). Le tableau de la figure II-41 indique les conditions utilisées pour décider de la variation du seuil (T) et de la sortie du CPCS (W) avec :

- A_{si} (t) : la valeur de l'équation majoritaire contenant le bit s_i du syndrome.
- Error (t) : signal indiquant une erreur à l'instant t.

Cette table est valable si une seule erreur survient pendant les m cycles d'horloge après t. Si plusieurs erreurs surviennent, cette table peut être modifiée de la façon suivante :

β tables sont créées (seul l'instant t utilisé pour ces tables change en fonction du moment où l'erreur se produit). β représente alors le nombre d'erreurs détectées entre l'instant t et t+m.

les variances (T,W) sont additionnées arithmétiquement ligne par ligne pour connaître la modification à apporter à T et W.

instant	condition	W	T
t_{1-m3}	$Error(t).A_{s0}(t_{m1-m+1})=1$	-1	+1
	$Error(t).A_{s0}(t_{m1-m+1})=1$	+1	-1
t_{2-m3}	$Error(t).A_{s1}(t_{m1-m+2})=1$	-1	+1
	$Error(t).A_{s1}(t_{m1-m+2})=1$	+1	-1
t_{3-m3}	$Error(t).A_{s2}(t_{m1-m+3})=1$	-1	+1
	$Error(t).A_{s2}(t_{m1-m+3})=1$	+1	-1
⋮	⋮	⋮	⋮
t_{m1+m2}	$Error(t).A_{sm-1}(t_{m1})=1$	-1	+1
	$Error(t).A_{sm-1}(t_{m1})=1$	+1	-1

Fig II-41. Conditions utilisées dans l'architecture pipeline du décodeur à logique majoritaire

Un décodeur DSCC (1057,813) utilisant cette technique a été réalisé par NHK. Les caractéristiques de ce circuit sont présentées dans le tableau de la figure II-42.

Technologie	1 µm CMOS
Nb de porte	13000
Registre	1062 bit (SRAM)
Puissance	0,75 W
m	5
Fmax	50 Mhz
Débit	25 Mbits./s

Fig II-42. Caractéristiques du décodeur pipeline pour le code DSCC(1075,813)

Cette technique du pipeline est très intéressante pour réaliser des décodeurs à haut débits pour des codes ayant des polynômes générateurs de degré élevé. De plus, cette technique permet d'implanter « facilement » le décodage majoritaire à seuil variable. Cependant elle ne permet pas de répondre à une de nos contraintes (traitement de flot continu de données). Comme pour le codeur, nous allons examiner une autre architecture : l'architecture parallèle.

II.3.1.3. Architecture Parallèle.

II.3.1.3.1. Détection et localisation des erreurs.

Si on reprend le schéma de la figure (II-38), la détection et la localisation se fait via la matrice W et la porte à logique majoritaire (LMG). Ce système peut être décrit par les deux équations :

$$R^{(1)} = R^{(0)}[T] \oplus [c_0 | 0_{1\times r-2}] \quad (2.25)$$

$$W^{(1)} = R^{(1)}W = R^{(0)}TW \oplus [c_0 | 0_{1\times r-2}]W \quad (2.26)$$

Avec :

- $R^{(i)}$ le reste (syndrome) à l'instant i
- c_0 la correction à effectuer
- $W^{(i)}$ la sortie du CPCS à l'instant i

Par récurrence, l'équation (2.26) devient :

$$W^{(p)} = R(0)[T]^p W \oplus \left(\sum_0^{p-1} [c_i | 01\times r-2][T]^{p-1+i}\right)W \quad (2.27)$$

Il est alors possible de connaître la sortie du circuit CPCS à l'instant p en fonction de la valeur du reste à l'instant 0 et des bits de correction des cycles précédents. Il est donc obligatoire d'attendre les valeurs c_i pour pouvoir calculer $W^{(p)}$. La figure (II-43) donne un exemple d'architecture utilisée pour implanter la formule (2.27) (p=3).

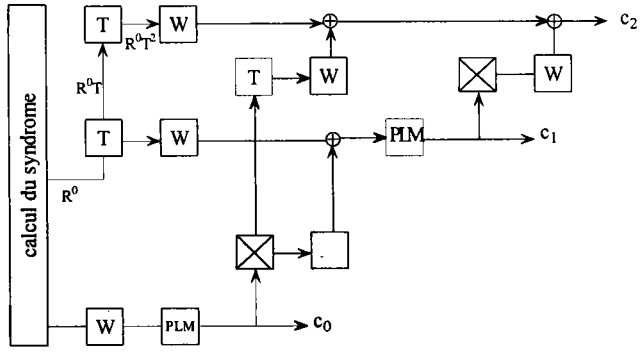


Fig II-43. Architecture implantant la détection des erreurs

Comme le circuit doit attendre de connaître les précédents bits de correction, le chemin critique correspond au temps nécessaire pour calculer la valeur du dernier bit de correction. Avec cette méthode, le gain en débit par rapport à un modèle série est assez faible et s'accompagne d'un accroissement important de la surface.

Par conséquent, cette méthode revient à :

Etape 1 :Calculer les équations majoritaires W correspondant à R_0

Etape 2 :Evaluer l'erreur C_i du bit à la position (n-1)

Etape 3 :Recalculer le reste en tenant compte d'une éventuelle correction et retourner en étape 1

Ces trois étapes sont nécessaires car la matrice W utilisée ne teste que les bits du reste en position n-1. Cependant, il est possible de générer des matrices W(i) qui testent de façon indépendante les positions i appartenant à l'ensemble {n-2, n-3,…,n-p}.

Pour le code DSCC, les équations de la matrice W sont données par la formule :

$$W = x^{n-1-q_{2^s}+q_j} Z^*(x) \; \forall j \in \{0,\ldots,2^s\}$$

Le terme (n-1) représente le bit testé. Si on veut tester les bits (n-i) on implante la série d'équation :

$$W_i = x^{n-i-q_{2^s}+q_j} Z^*(x) \; \forall j \in \{0,\ldots,2^s\}$$

Le test en parallèle des bits en position (n-1,n-2,..,n-p) se fait alors en implantant les matrices W_1 , W_2 ,..., W_p. (autre exemple : pour le code DTI, au lieu d'appliquer la permutation $\alpha Y + \alpha^{n-1}$ on utilise la permutation $\alpha Y + \alpha^{n-i}$).

Cette technique permet alors le test de façon simultanée et en parallèle des bits situés au positions (n-1,n-2,...n-p). L'architecture du décodeur devient alors celle de la figure II-44 :

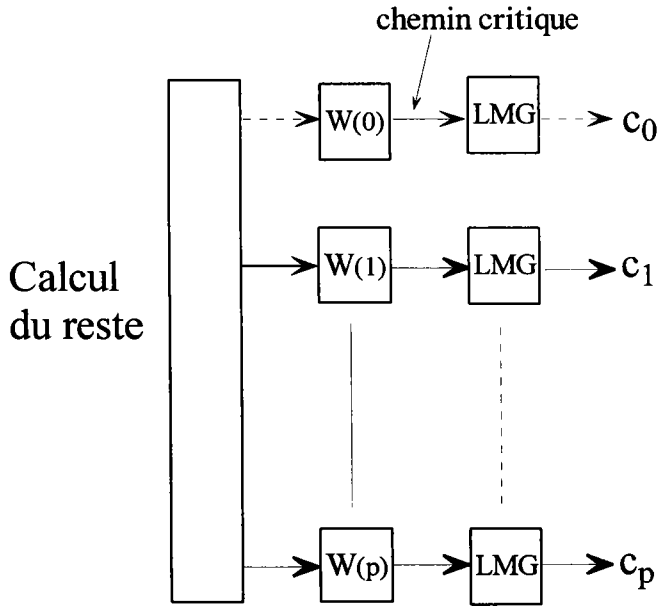


Fig II-44. Nouvelle architecture du décodeur

Les différents bits de correction étant évalués, il faut réintroduire ces bits dans le calcul du syndrome permettant de corriger les p bits suivants du mot code reçu.

II.3.1.3.2. Correction des erreurs

Après l'évaluation des p bits de correction, le mot corrigé peut s'écrire :

$$C_{corr}(x) = C(x) \oplus \eta(x) = \left(\sum_0^{n-1} c_i x^i\right) \oplus \left(\sum_{n-p}^{n-1} \eta_{n-1-i} x^i\right) \quad (2.28)$$

Multiplions le vecteur C_{corr} par xp. On a alors :

$$x^p C_{corr}(x) = \left(\sum_p^{n-1+p} c_{j-p} x^j\right) \oplus \left(\sum_n^{n-1+p} \eta_{n-1+p-j} x^j\right) \quad (2.29)$$

Or par définition, x^n est égal à 1. Par conséquent, l'équation (2.29) peut aussi s'écrire :

$$x^p C_{corr}(x) = \left(\sum_p^{n-1} c_{j-p} x^j\right) \oplus \left(\sum_0^{p-1} \left[\eta_{p-1-L} \oplus c_{L-n-p}\right] x^L\right) \quad (2.30)$$

La multiplication par x^p revient alors à un décalage circulaire de p bits vers la droite (fig II-45).

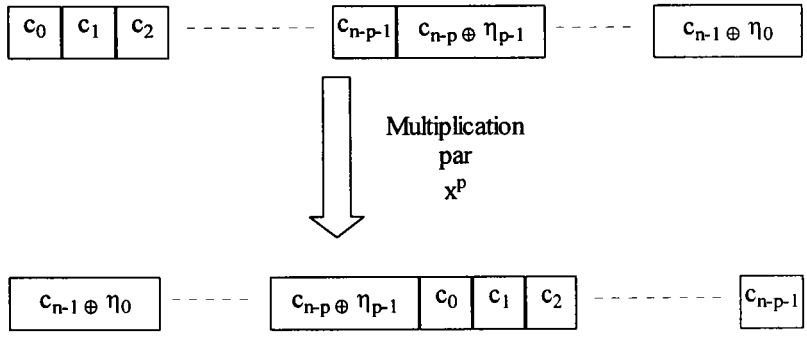


Fig II-45. Effet de la multiplication par x^p sur le mot de code

Le code utilisé est un code linéaire. Le nouveau mot est donc un mot de code. On va alors pouvoir utiliser ce mot pour tester les p bits suivants (généralisation du décodeur de Meggitt).

Pour pouvoir tester ces bits, on doit d'abord calculer le reste de la division de ce mot par g(x). On a alors :

$$x^p C_{corr}(x) = x^p (C(x) + \eta(x)) = S_{corr}(x) + a(x)g(x) \quad (2.31)$$

Appelons s(x) le syndrome de C(x). La formule (2.31) est transformée en :

$$x^p C_{corr}(x) = x^p [S(x) + b(x)g(x)] + x^p \eta(x) = S_{corr}(x) + a(x)g(x) \quad (2.32)$$

D'après [13 p 100] le reste de la division de $x^p S(x)$ par g(x) est égal au syndrome S(x) décalé de p bits vers la droite $S^{(p)}(x)$. $x^p \eta(x)$ étant un polynôme de degré p-1, si ce degré est inférieur à celui de g(x), le reste de la division de $x^p \eta(x)$ par g(x) est égal à $x^p \eta(x)$.

On en conclut alors que :

$$S_{corr}(x) = S^{(p)}(x) \oplus x^p \eta(x) \quad (2.33)$$

Le calcul du nouveau reste peut alors être implanté par l'architecture suivante :

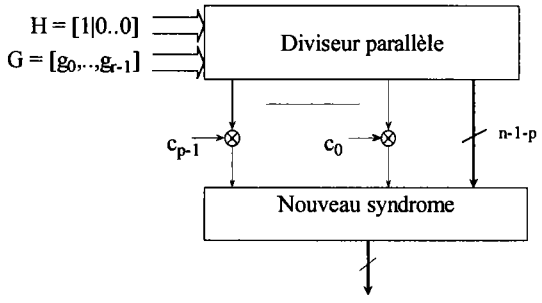


Fig II-46. Architecture parallèle pour le calcul du nouveau syndrome

Cependant, pour des mesures pratiques, on utilise parfois des codes raccourcis. Par exemple, le code DSCC(272,190) utilisé pour protéger les transmissions Teletext est le code DSCC(273,191) raccourci d'un bit (en fait, on se limite aux mots de code pour lesquels le bit (273) est égal à zéro). La méthode présentée ici doit alors être modifiée.

II.3.1.3.3. Cas des codes raccourcis.

On considère maintenant que le mot de code est de taille m+1 inférieure à n (code raccourci). Le mot code peut alors s'écrire :

$$C(x) = \sum_0^m c_i x^i + \sum_{m+1}^{n-1} 0x^i \quad (2.34)$$

Supposons que l'on corrige les p derniers bits $[c_{m-p+1},\ldots,c_m]$ du mot reçu. Le mot de code corrigé devient alors :

$$C'(x) = \sum_0^m c_i x^i + \sum_{m+1}^{n-1} 0x^i + \sum_{m-p+1}^{m} (\eta_{m-i})x^i \quad (2.35)$$

en utilisant la même procédure que précédemment, on a :

$$x^p C'(x) = x^p C(x) + \sum_{m+1}^{m+p} (\eta_{m+p-j})x^j \quad (2.36)$$

Soient :

- $S^{(p)}$ les restes de la division de $x^p C(x)$ par g(x).

- $S_I^{(p)}$ le reste de la division de x^p par g(x).

Le reste S' de la division de $x^p C'(x)$ par g(x) s'exprime alors en fonction de $S^{(p)}$ et de $S_I^{(p)}$ par :

$$S' = S^{(p)} + \sum_{m+1}^{m+p} (\eta_{m+p-j}) S_I^{(j)} \quad (2.37)$$

Supposons que l'inégalité ($m+1 \leq n \leq m+p$) soit vérifiée, l'équation (2.36) s'écrit :

$$S' = S^{(p)} + \sum_{m+1}^{n-1} (\eta_{m+p-j}) S_I^{(j)} + \sum_{n}^{m+p} (\eta_{m+p-j}) S_I^{(j)} \quad (2.38)$$

Or pour $i \geq n$, on a $x^i = S_I^{(j)}$. Par conséquent,

$$S' = S^{(p)} + \sum_{m+1}^{n-1} (\eta_{m+p-j}) S_I^{(j)} + \sum_{n}^{m+p} (\eta_{m+p-j}) x^j \quad (2.39)$$

L'architecture du diviseur pour les codes raccourcis est alors représentée à la figure II-47 :

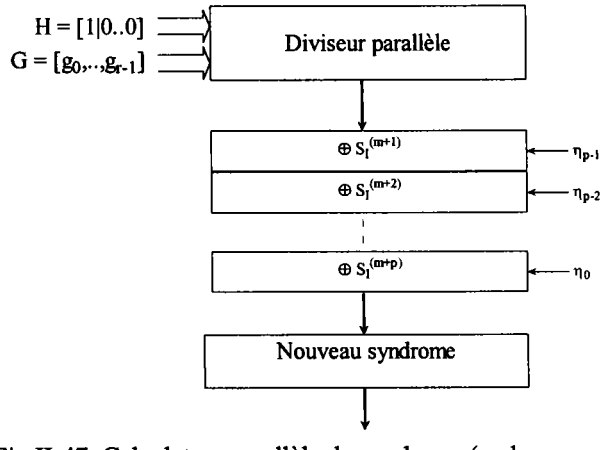


Fig II-47. Calculateur parallèle du syndrome (code raccourci)

II.3.1.3.4. Circuit de décodage

L'architecture générale du cœur du décodeur est représentée à la figure II-48.

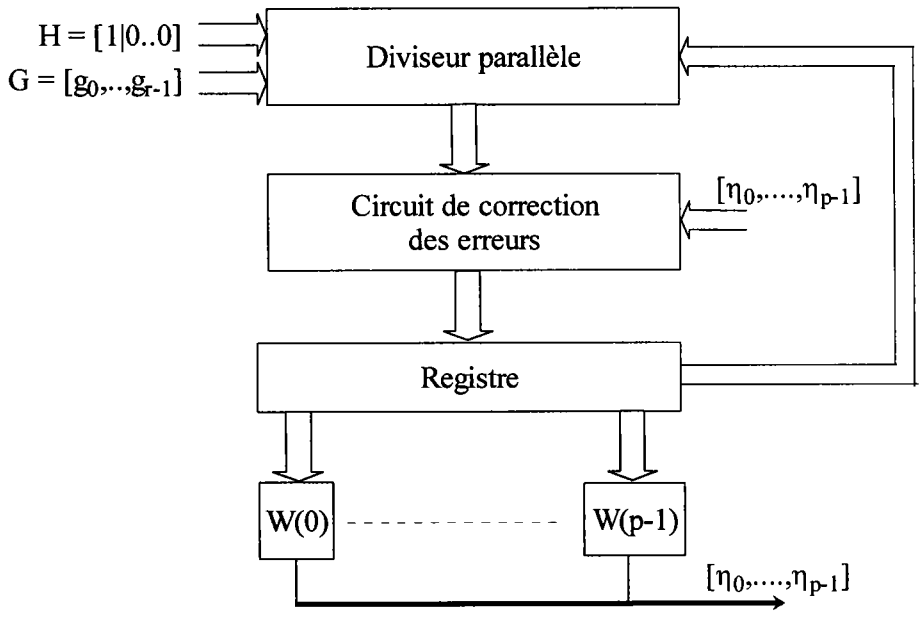


Fig II-48. Structure du circuit de décodage parallèle

Comme pour le modèle série, le chemin critique correspond au rebouclage des bits de correction. Les matrices W et les portes à logique majoritaire étant identiques à celles utilisées dans l'architecture série, le chemin critique est augmenté essentiellement par deux circuits.

- Le circuit de correction des erreurs : dans le cas parallèle, on introduit des vecteurs (reste). Par contre dans le cas série, seul un bit de correction est introduit.

- Le diviseur parallèle

Cette augmentation est cependant contrebalancée par le fait que l'on puisse corriger plusieurs bits en un seul cycle de traitement. Le débit de sortie des données peut alors être fortement augmenté.

Toutefois, comme les portes à logique majoritaire et les matrices W sont dupliquées, on risque un accroissement important de la surface du décodeur selon le degré de parallélisme choisi. Une étude préalable concernant cet accroissement est donc nécessaire (partie II.4).

II.3.1.4.1. Conclusions

Trois architectures sont possibles pour implanter le décodage à logique majoritaire. Les caractéristiques de ces architectures sont regroupées dans le tableau de la figure II-49. On peut s'apercevoir que les seules architectures pouvant être retenues pour l'implantation à haut débit sont les architectures systolique et parallèle.

Architecture	série	systolique	parallèle
débit	Faible	Moyen	Moyen à élevé
surface	Faible	Faible-moyenne	Moyenne à élevée
code raccourci	oui	Difficile à implanter	oui

Fig II-49. Caractéristiques des différentes architectures

Le choix d'une méthode ou d'une autre dépend alors de l'application visée. Si elle nécessite l'utilisation d'un code avec un polynôme générateur de haut degré, il est préférable d'utiliser l'architecture systolique. Il sera cependant très difficile de traiter des codes raccourcis ou des flots continus de données. Par contre, si le degré de parallélisme est raisonnable (limité à 32 bits dans notre cas), il est possible de traiter des flots continus de données à haut débit en utilisant l'architecture parallèle.

Après ce rapide survol des différentes architectures possibles pour le décodeur à logique majoritaire, nous allons expliquer l'architecture parallèle du circuit de décodage

II.3.2. Réalisation du décodeur

II.3.2.1. Schéma général du décodeur

La contrainte sur le décodeur est identique à celle du codeur. L'architecture générale (fig II-50) est similaire à l'architecture du codeur.

Cependant, les relations (2.19) et (2.20) régissant les différentes fréquences du décodeur (f_{in}, f_{out} et f_f) sont respectivement transformées en :

$$nf_{out} \leq kf_{in} \quad (2.40)$$

$$3\,(n/p+1)f_{in} \leq n(f_f) \quad (2.41)$$

En effet, n/p cycles sont nécessaires pour calculer le premier reste (syndrome initial). Ensuite les n bits doivent être testés et combinés avec les bits de correction (2n/p cycles d'horloges). Les 2 cycles restant de la formule sont des cycles d'initialisation et de commande

En outre, le circuit de décodage peut utiliser des composants présents dans le circuit de codage. On va par exemple réutiliser le circuit de conversion série/parallèle du codeur mais avec des paramètres différents. De même les mémoires utilisées sont identiques à celle du codeur, seule la profondeur de ces FIFO est modifiée (elle est égale à n/p).

Par conséquent, trois circuits doivent être générés pour le codeur :

- la conversion parallèle série.
- le circuit de décodage et de mise en forme
- le circuit de contrôle du décodeur.

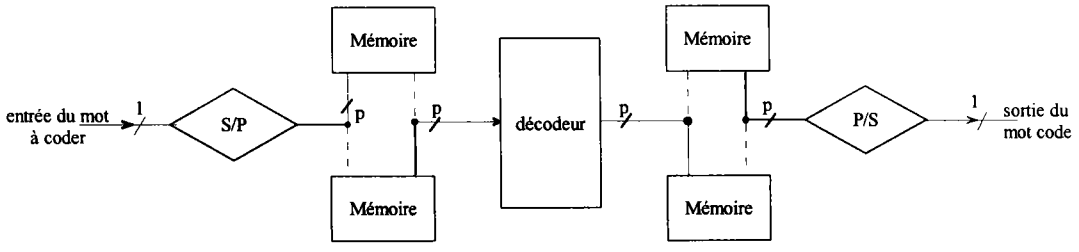


Fig II-50. Schéma de principe du décodeur parallèle

II.3.2.2. La conversion parallèle/série

Le problème rencontré pour la conversion parallèle/série du décodeur est similaire à celui de la conversion série/parallèle du codeur. Le mot code corrigé est constitué de (n/p) symboles de p bits. Mais seuls les k premiers bits du mot intéresse l'utilisateur. Or il est possible que (k/p) ne soit pas un entier. Par conséquent, les $\lfloor k/p \rfloor$ premiers symboles doivent être entièrement convertis et seuls les δ derniers bits du symbole suivant sont à considérer pour la conversion. L'architecture du convertisseur qui réalise cette fonction est similaire à celle du convertisseur série/parallèle du codeur. Trois

registre à décalage avec chargement parallèle sont utilisés. Sur ces trois registres, deux sont de tailles (p) et un est de taille (δ). Les deux premiers registres sont utilisés alternativement pour la conversion des $\lfloor k/p \rfloor$ symboles. Les (δ) derniers bits sont convertis en utilisant le dernier registre. Les sorties de ces convertisseurs sont alors multiplexées pour constituer la sortie du mot code (fig II-51).

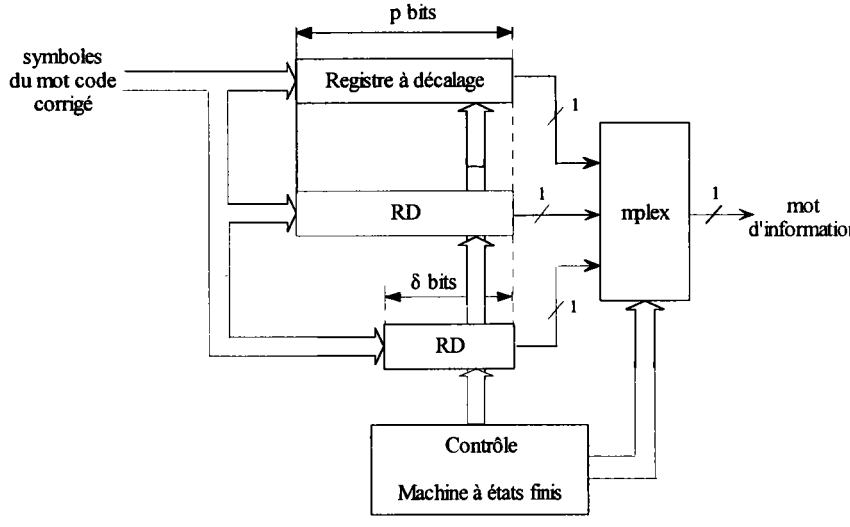


Fig II-51. Architecture du convertisseur parallèle/série.

Une simulation de cette architecture est représentée à la figure II-52. Dans cet exemple, le mot code corrigé est constitué de 5 symboles de 6 bits et le mot d'information est de longueur de 29 bits. Par conséquent, les 4 premiers symboles sont entièrement convertis et seuls les 5 derniers bits du symbole suivant sont considérés (le premier zéro du dernier symbole est supprimé).

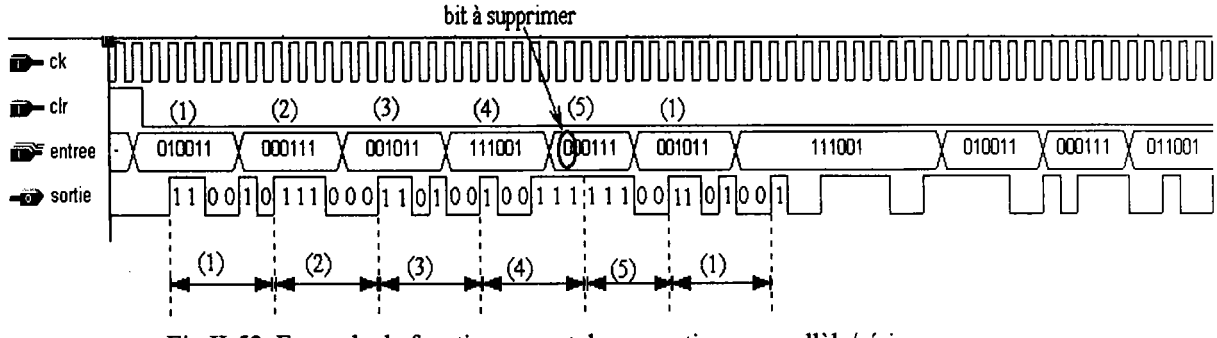


Fig II-52. Exemple de fonctionnement du convertisseur parallèle/série

II.3.2.3. Le circuit de décodage

II.3.2.3.1. La porte à logique majoritaire

D'après l'algorithme de décodage, la porte à logique majoritaire doit réaliser deux fonctions :

1- compter le nombre Nb de « 1 » en sortie de la matrice W considérée,

2- comparer ce nombre Nb à un seuil T (Threshold) et générer la sortie S en fonction de cette comparaison (si Nb ≥ T.alors S = 1 sinon S = 0).

Ces fonctions peuvent être implantées de deux manières différentes, soit par comptage soit en utilisant un circuit de tri.

II.3.2.3.1.1. Méthode par comptage

Une méthode « naïve » consiste à utiliser un réseau de compteurs dont la sortie (i.e. le nombre de lignes à « 1 ») est comparée au seuil T. Cependant, cette méthode présente plusieurs inconvénients :

la profondeur P_R du réseau dépend du nombre de ligne à tester et du nombre de bits sur lequel est codé le résultat. Pour les codes de la famille DSCC, le nombre de lignes à tester est en première approximation multiplié par 2 entre chaque code (le nombre de ligne à tester suit la formule 2^s+1 où s est le numéro du code). Sachant qu'un additionneur complet utilise 3 portes AND et 2 portes OR, la surface du circuit de comptage sera égale à en première approximation :

$$S_f = P_R(3S_{AND}+2S_{OR})$$

De même, si on appelle T_{FA} le temps mis pour traverser un additionneur complet, le temps T_f pour que la sortie finale du compteur soit valide est de :

$$T_f = P_R T_{FA}.$$

Dans [30], NHK a utilisé un circuit de comptage en pipeline (3 étages) pour le code DSCC(1057,813). Il est constitué de 34 additionneurs complets avec une profondeur de 9. Il est capable de tester 33 lignes.

II.3.2.3.1.2. Méthode par tri

Pour réaliser la porte à logique majoritaire, on peut utiliser une « trieuse de lignes » qui va comparer les valeurs des lignes (1 ou 0) et les séparer en deux sous-groupes (fig II 53 a) :

Un groupe G1 de lignes à « 1 »,

Un groupe G2 de lignes à « 0 ».

Toutes les lignes étant triées, il suffit de tester la ligne L_T correspondant au seuil (i.e. examiner sa valeur) pour décider de la valeur de sortie de la porte à logique majoritaire. En effet, si la valeur de la ligne LT est de « 1 », alors le nombre de lignes à 1 est supérieur ou égal au seuil voulu et la sortie de la porte à logique majoritaire vaut 1 (fig II-53 b). Dans le cas contraire (ligne à zéro), la sortie de la porte à logique majoritaire prend la valeur zéro (fig II-53 c).

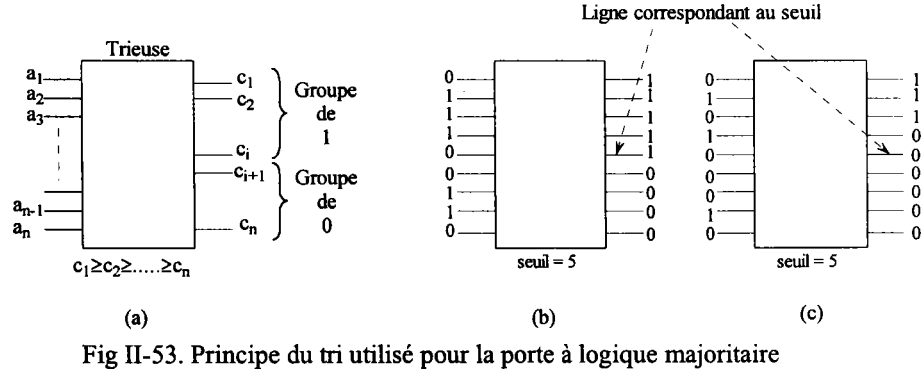


Fig II-53. Principe du tri utilisé pour la porte à logique majoritaire

Pour implanter cette fonction, on peut utiliser un circuit qui fusionne deux listes ordonnées en une seule liste ordonnée. Les deux listes d'entrée pouvant elles même être les sorties de deux autres circuits de fusion, on peut, grâce à un réseau de circuit de fusion, trier n'importe quelle liste (fig II-54).

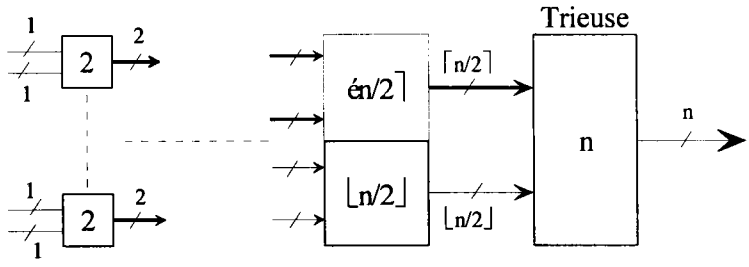


Fig II-54. Architecture générale d'une trieuse de n lignes

Dans notre cas, nous avons utilisé un circuit de fusion Impair-Pair (Odd-Even Merge –OEM) [31],[32] qui fonctionne de la façon suivante :

Les lignes des deux listes entrantes A et B sont indexées (1 à s pour la liste A et 1 à t pour la liste B). Toutes les lignes indexées par un nombre impair sont réunies dans une seule liste qui constitue l'entrée d'un circuit de fusion de taille plus petite (Odd Merge). De même, les lignes indexées par un nombre pair forment une nouvelle liste qui rentre dans un autre circuit de fusion OEM de taille plus faible (Even Merge). Les sorties de ces deux circuits sont alors comparées pour créer la liste ordonnée C. Cette comparaison suit les règles suivantes :

1-La première sortie du circuit de fusion impair constitue la valeur la plus faible de la liste ordonnée résultante.

2-La $i^{ème}$ sortie du circuit de fusion pair est comparée à la $(i+1)^{ème}$ valeur du circuit de fusion impair. Les sorties du comparateur forment les $(2i)^{ème}$ et $(2i+1)^{ème}$ éléments de la liste finale.

3- Dans le cas où une des sorties des circuits de fusion ne pourrait être comparée, cette ligne reste seule et constitue la valeur la plus élevée de la liste.

Un exemple de circuit (liste à 9 éléments) utilisant cet algorithme est représenté à la figure II-55. Il est constitué de 10 comparateurs à 2 entrées et d'un comparateur à 3 entrées. Dans ce circuit, le chemin critique est constitué par le passage à travers 4 comparateurs (le comparateur à 3 entrées peut être assimilé à deux comparateurs à 2 entrées).

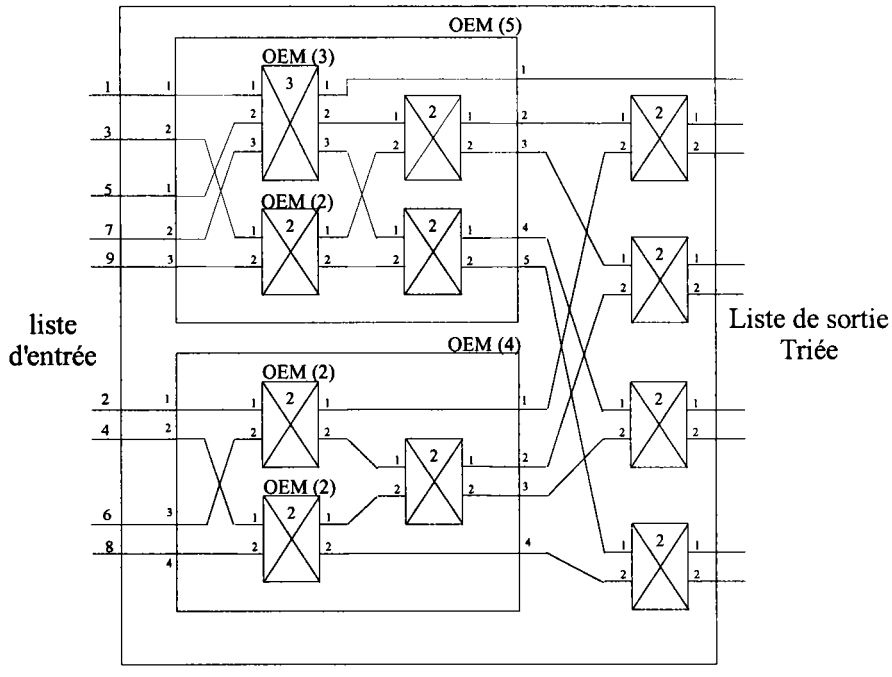


Fig II-55. Exemple d'un OEM à 9 éléments

II.3.2.3.1.3. Choix de la méthode

En simulant les deux méthodes (tri et comptage) avec le logiciel Maxplus II, on s'aperçoit qu'en terme de débit et de surface elles sont équivalentes sur les FLEX10k. En effet, lors de la synthèse, le programme met à plat toutes les équations combinatoires de façon à pouvoir effectuer les réductions nécessaires. Les deux méthodes implantant la même fonction combinatoire (mais décrite différemment), le résultat obtenu est similaire. Dans notre cas, nous avons retenu la méthode de tri car sa structure régulière est facilement décrite en VHDL.

II.3.2.3.2. La matrice W

La matrice W est très particulière car sa construction dépend du code utilisé. Par exemple pour le code DSCC, sa construction est basée sur des opérations polynomiales. Par contre pour les codes DTI, on travaille avec des permutations affines dans des corps de Galois. Un descriptif en VHDL n'est pas possible ou vraiment trop complexe pour être efficace. Un programme annexe a alors été utilisé sous Matlab pour calculer les

différentes matrices nécessaires au décodeur (les paramètres d'entrée sont le nom du code et le degré de parallélisme) et générer le fichier VHDL correspondant.

II.3.2.3.3. Le syndrome

Le troisième élément nécessaire au décodage est le calcul du syndrome. Ce syndrome est le reste de la division d'un polynôme (donnée à décoder ou reste partiel) par le polynôme générateur g(x). Le syndrome peut alors être implanté via un diviseur parallèle identique à celui utilisé pour le codage mais avec des paramètres d'entrées différents. Deux modes d'utilisation de ce diviseur sont à distinguer :

Au départ, on doit calculer le reste de la division de la donnée à décoder par le polynôme g(x). On ne doit pas tenir compte d'éventuelles corrections. Par contre lors de la correction des données, on doit bloquer toute entrée possible de données parasites. Un schéma possible d'implantation du calcul du syndrome est représenté à la figure II-56

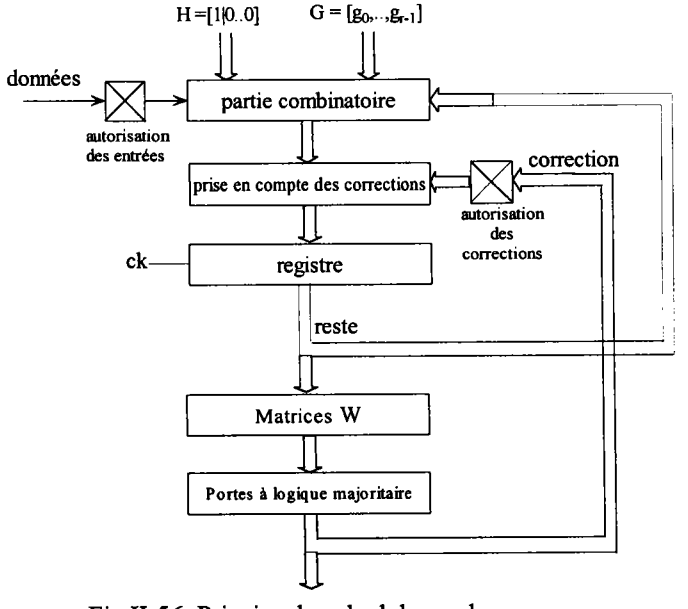


Fig II-56. Principe du calcul du syndrome

Le problème d'un tel circuit est sa lenteur. En effet, le calcul du reste doit traverser à la fois le circuit de prise en compte des corrections (auquel il faut rajouter le circuit d'évaluation des corrections) et le circuit d'entrée des données. Pour augmenter le débit

du circuit de calcul du syndrome, on a préféré utiliser l'architecture décrite à la figure II-57

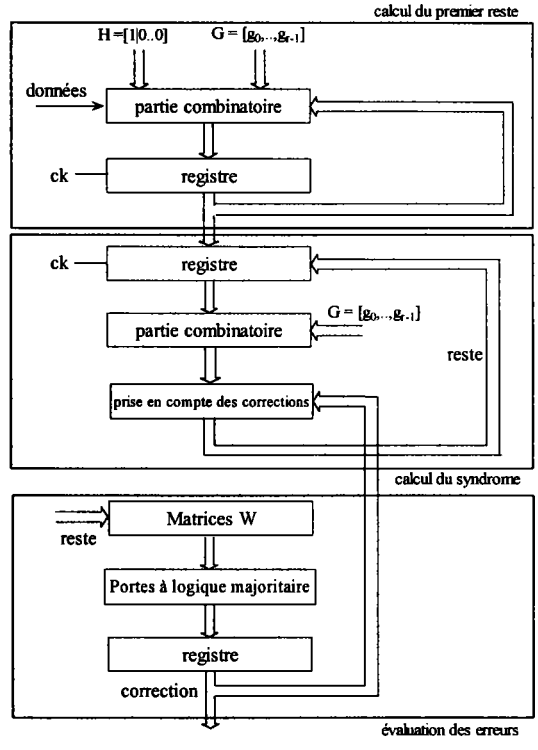


Fig II-57. Architecture utilisée pour le calcul du syndrome

Dans le circuit initial, le calcul du premier reste est ralenti par les circuits d'évaluation et de correction des erreurs. Un seul sous-circuit est utilisé pour réaliser cette fonction et améliorer la fréquence de fonctionnement du système.

De même, les circuits d'évaluation et de correction des erreurs sont séparés du calcul du syndrome (registre). Il faut alors deux cycles d'horloges pour obtenir le résultat. Il est cependant nécessaire de vérifier que ce nouveau temps est inférieur à celui donné par l'architecture précédente (partie II.4).

II.3.2.3.4.Cœur du décodeur

Les circuits précédents permettent l'évaluation des corrections à effectuer sur le mot reçu. Comme les corrections ne sont valables que [(n/p) +2] cycles d'horloges (temps mis pour calculer le premier reste et le premier symbole de correction) après l'entrée du

premier symbole de la donnée reçue, le circuit de décodage doit mémoriser la donnée pendant ces [(n/p) +2] cycles. On peut alors utiliser une FIFO de profondeur [(n/p)+2] afin de synchroniser la correction et la donnée entrante. Comme la correction n'est valable que tout les deux cycles d'horloges, la FIFO doit être verrouillée pendant un cycle d'horloge au moyen d'un signal interne de blocage. L'architecture générale du décodeur est alors représentée à la figure II-58.

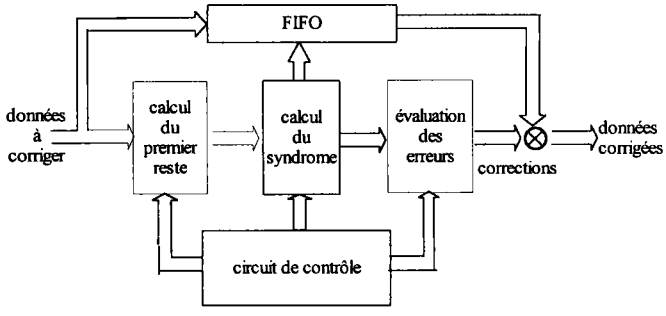


Fig II-58. Architecture du cœur du décodeur

Sur la figure II-59, on peut voir un exemple de la simulation du décodeur parallèle DSCC(72,44) pour un parallélisme de 3. Le mot code est représenté par la séquence héxadécimale « 123456701234563253234555 ». Quatre erreurs binaires (capacité maximale de correction du code) ont été ajoutées au mot code initial (4→5, 4→0, 3→5). Le mot reçu est alors représenté par la séquence hexadécimale « 123556701230563253254555 ». Après passage dans le décodeur, la sortie (d_corrected) renvoie la séquence corrigée. Comme on l'avait annoncé dans la description de l'architecture parallèle, deux cycles d'horloges sont nécessaires pour la correction d'un symbole. Le temps total pour lire et corriger la donnée est de 3(n/p+1) cycles d'horloges ce qui correspond aux contraintes demandées (formule 2.40)

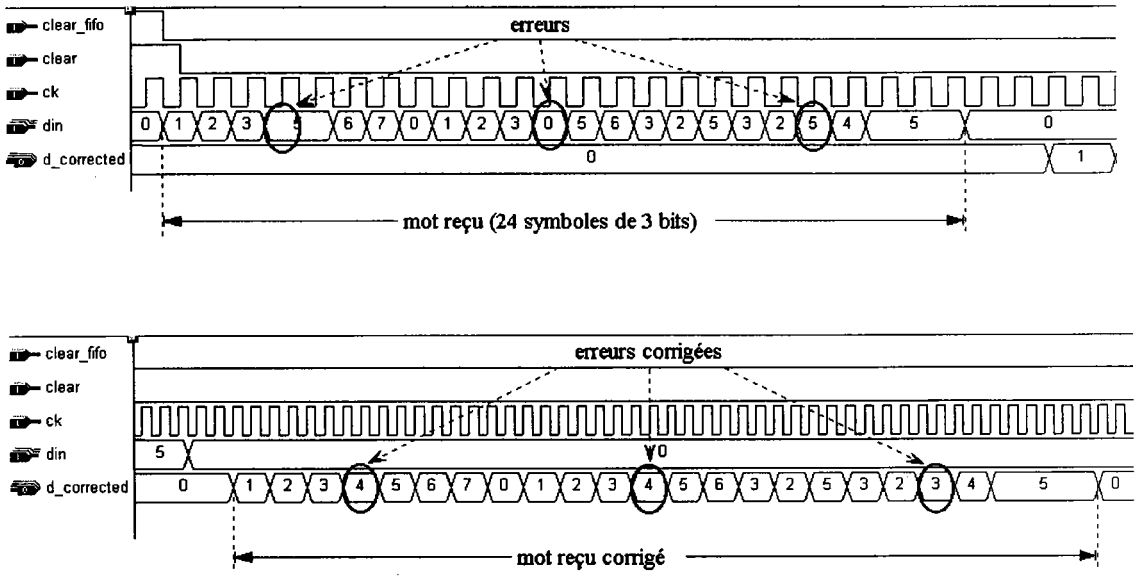


Fig II.59. Simulation du décodeur parallèle

II.3.3. Conclusions.

Comme pour le codeur, différentes architectures de décodeur ont été étudiées. Il s'ensuit que si l'architecture série pipelinée est très intéressante pour les codes dont le degré du polynôme générateur est grand, elle ne peut pas répondre à nos contraintes. Par contre, l'architecture parallèle peut répondre à nos besoins. Cependant, les modifications apportées à l'algorithme (prise en compte des codes raccourcis) ainsi que l'architecture choisie pour le décodeur vont avoir un impact non négligeable sur la surface occupée par le circuit de décodage. L'étape suivante va donc consister à simuler les circuits de codage et de décodage pour différents codes et degré de parallélisme afin d'étudier conjointement les variations de surface et de débit de ces circuits.

II.4 Résultat

II.4.1. Etude du codeur.

II.4.1.1. Méthodologie

Notre première tâche a été de valider l'algorithme parallèle du codeur. Pour effectuer cette opération, deux programmes ont été développés.

Le premier programme écrit avec le logiciel Matlab implante l'algorithme parallèle en permettant le choix du polynôme générateur et du degré de parallélisme. La donnée à coder est mémorisée dans une matrice $\lceil k/p \rceil \times p$. Dans cette matrice, les $(p-\delta)$ zéros utiles pour l'algorithme ont déjà été rajoutés. Après codage, le résultat est enregistré dans une deuxième matrice de dimension $(n/p)\times p$. Parallèlement à ce programme, la matrice correspondant à la donnée à coder M est utilisée avec un logiciel de calcul formel afin de calculer le reste de la division de $x^r M(x)$ par $g(x)$. La validation de l'algorithme est effectuée en comparant la « matrice résultat » (matrice où sont stockées les résultats) et les informations données par ce logiciel.

Après validation de l'algorithme, une première version du codeur décrite en VHDL a été synthétisée avec le logiciel Maxplus II de la société Altera. Dans cette version, le polynôme générateur et le degré de parallélisme sont fixes. Le logiciel Maxplus II a permis de simuler (temps de retard, fréquence de fonctionnement) le codeur et de valider l'architecture.

Cette version a ensuite été modifiée afin de permettre le choix du polynôme générateur et du degré de parallélisme. Deux informations nous ont intéressées : la fréquence maximum de fonctionnement du circuit et le nombre de cellules utilisées pour implanter le codeur (LCE). Pour réaliser l'étude en surface et en débit du codeur, les différents polynômes ont été choisis en fonction de leur taille (mot du code) et de leur redondance. Chacun de ces codes se caractérise par des tailles sensiblement identiques. De plus, un autre code a été simulé pour pouvoir effectuer une comparaison avec le modèle de diviseur parallèle de la société Altera.

II.4.1.3. Etude en débit et en surface du diviseur parallèle

Nom du Code	n	k	G(x)
DTI(6,3,21)	60	46	$1 + x^2 + x^3 + x^4 + x^9 + x^{10} + x^{11} + x^{14}$
DSCC(72,44)[8]	72	44	$1 + x^2 + x^4 + x^6 + x^8 + x^{12} + x^{16} + x^{22} + x^{25} + x^{28}$
EG(0,3,2)	60	34	$1 + x^2 + x^6 + x^{10} + x^{12} + x^{13} + x^{14} + x^{15} + x^{16} + x^{24} + x^{26}$

Fig II-60. Caractéristiques des polynômes utilisés pour l'étude de la surface et du débit

Les caractéristiques des polynômes utilisés pour l'étude de la surface et du débit du diviseur parallèle sont regroupés dans le tableau de la figure II-60. Ces trois codes sont à peu de chose près équivalent (taille du reste et taille du mot code). Les figures II 61a et II 61b regroupent les résultats obtenus à partir de la synthèse du diviseur parallèle ces différents polynômes. L'étude comparée de l'évolution des débits et de la surface du diviseur parallèle montre une augmentation significative du débit (il est multiplié par 5 pour un degré de parallélisme de 9) et un accroissement limité de la surface (multiplié par 3 pour un degré de parallélisme de 9). Bien que le débit ne soit pas multiplié par le degré de parallélisme, le rapport D(p)/D(1) est de l'ordre de 0,6p (avec p le degré de parallélisme) et est supérieur à celui prévu dans [33] (0,4p~0,5p).

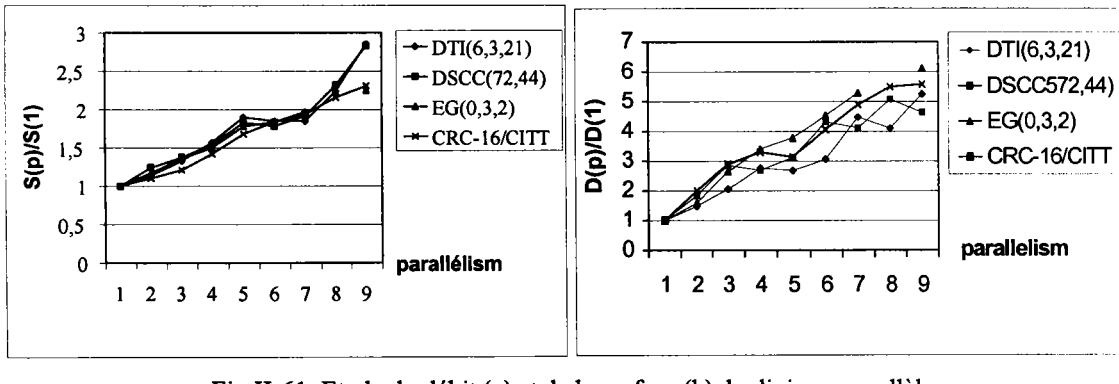


Fig II-61. Etude du débit (a) et de la surface (b) du diviseur parallèle

Outre cette étude, le diviseur parallèle a été comparé à une autre architecture de diviseur parallèle. La synthèse s'effectuant sur un FPGA de la société Altera, nous ne pouvons comparer nos résultats qu'avec des fonctions disponibles pour ces circuits. La

seule fonction possible (CRC Megacore Function commercialisée par la société Altera [34]) implante un diviseur parallèle dont la principale contrainte est que le rapport R_p entre la taille du reste (r) et le degré de parallélisme soit entier. Le tableau de la figure II-62 regroupe les résultats de la comparaison de notre architecture avec celle d'Altera.

p	code testé	Nombre de LCE		Fréquence de travail (Mhz)	
1	DSCC(72,44	Altera	28	Altera	125
		Arch parallèl	37	arch parallèle	125
2	DSCC(72,44	Altera	28	Altera	125
		Arch parallèl	48	Arch parallèl	114,9
4	DSCC(72,44	Altera	32	Altera	103,09
		Arch parallèl	57	Arch parallèl	84,03
7	DSCC(72,44	Altera	52	Altera	66,22
		Arch parallèl	71	Arch parallèl	66,66
14	DSCC(72,44	Altera	108	Altera	45,44
		Arch parallèl	125	Arch parallèl	45,45
1	DTI (6,3,21)	Altera	14	Altera	125
		Arch parallèl	21	Arch parallèl	125
2	DTI (6,3,21)	Altera	16	Altera	125
		Arch parallèl	24	Arch parallèl	115
7	DTI (6,3,21)	Altera	37	Altera	89
		Arch parallèl	39	Arch parallèl	90

Fig II-62. Comparaison des deux diviseurs parallèles

On peut remarquer que notre architecture occupe plus de surface que celle développée par Altera. Cela peut s'expliquer par le fait que cette fonction a été développée et optimisée par Altera pour des circuits d'Altera contrairement à notre solution qui n'est pas dédiée à un support type. On peut noter aussi que l'architecture développée par Altera est intéressante pour des degrés de parallélisme faibles.

L'accroissement de la surface circuit peut s'expliquer par la surface occupée par le circuit utilisé dans le cas où le rapport (k/p) n'est pas entier, cas qui constitue la valeur ajoutée de notre circuit.

II.4.1.3 Circuit de codage.

Après l'étude du diviseur parallèle, le circuit entier de codage a été synthétisé. La figure II-63 donne les caractéristiques du codeur parallèle DSCC(72,44) pour deux degrés de parallélisme 3 et 8.

Circuit	p	Surface (Lce)	Fin (Mhz)	Ft (Mhz)	Fout (Mhz)
EPF10K20TC144-3	3	982	45,24	32,67	46,29
EPF10K20TC144-3	8	1094	39,68	32,25	45,87

Fig II-63. Caractéristiques du codeur parallèle pour le code DSCC (72,44)

On peut remarquer que la différence de surface entre les deux codeurs parallèles n'est pas élevée (augmentation de 10%). Cette augmentation est due essentiellement à trois circuits :

- Les circuits de conversion série/parallèle et parallèle/série pour lesquels les registres passent d'une taille 3 à une taille 8.
- Le circuit de codage (diviseur parallèle et circuit de mise en forme).

Par contre les mémoires utilisées dans le codeur parallèle n'agissent pas sur l'accroissement de la surface car elles sont de tailles identiques (nombre de registre) mais disposées de façon différente (profondeur et taille de la FIFO).

On s'aperçoit aussi que les fréquences de fonctionnement des différents étages varie très peu et sont de l'ordre de 30 à 40 Mhz (les circuits de la famille des Flex10K peuvent fonctionner à une fréquence maximum de 125 Mhz). Mais alors que le circuit de division constitue le facteur limitatif du circuit de codage, ce sont les circuits d'entrée et de sortie qui limite le débit utile du codeur.

Enfin, les fréquences des différents étages permettent de traiter des flots continus de données (elles permettent de respecter les formules (2.21) et (2.22).

II.4.1.4 Conclusion.

Cette étude a permis de montrer que l'architecture parallèle permet d'augmenter de façon considérable le débit du diviseur parallèle. Bien que les performances de notre architecture soient légèrement inférieures à celle du diviseur développé par Altera, notre diviseur parallèle permet d'implanter des degrés de parallélisme avec la seule condition que le rapport entre la taille du mot code (n) et le degré de parallélisme (p) soit entier.

En ce qui concerne le codeur parallèle, il apparaît que les étages d'entrées et de sorties sont les facteurs limitatifs du circuit de codage. Une réécriture de ces étages devrait permettre d'augmenter les débits du circuit de codage (Ils seront au plus égaux à 125 Mbits car la fréquence maximale de fonctionnement des flex10K est de 125 Mhz).

II.4.2. Etude du décodeur.

II.4.2.1. Méthodologie de validation du décodeur

La démarche suivie pour la validation du décodeur est identique à celle du codeur. L'algorithme a d'abord été validé par une simulation logicielle. Pour ce faire, une matrice simulant un canal de transmission imparfait est additionnée à la matrice-résultat du codeur parallèle. Après application de l'algorithme de décodage, les erreurs trouvées sont comparées aux erreurs initiales afin de valider l'algorithme.

Après cette validation logicielle de l'algorithme, un modèle générique décrit en VHDL a été réalisé et synthétisé avec le logiciel MaxplusII d'Altera. L'étude de la surface et du débit a alors été effectuée avec les mêmes polynômes que pour le codeur.

II.4.2.2. Etude du décodeur.

II.4.2.2.1. Validation de l'architecture « pipelinée »

La première opération à consister à comparer les débits de l'architecture non pipelinée (Fig II-56) et pipelinée (fig II-57). Comme, nous n'ajoutons qu'une série de registres, la surface occupée par la deuxième architecture est plus importante que celle occupée par la première mais l'augmentation est faible. Il n'en est pas de même pour le débit. La figure II-64 montre l'évolution du débit Di en fonction du parallélisme pour ces deux architectures D1 et D2 (code DSCC(72,44)).

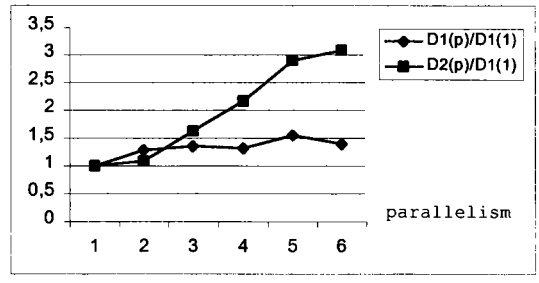


Fig II-64. Comparaison du débit des deux architectures proposées

On s'aperçoit que pour un degré de parallélisme faible, les deux architectures sont équivalentes. Par contre, pour des degrés élevés, le débit semble plafonner (debit série multiplié par 1,5) alors que le débit de la deuxième architecture est multiplié par 3 (multiplication par 2 du débit entre ces deux architectures). Cependant, il faut vérifier que la surface occupée par le circuit de décodage n'augmente pas plus vite que le débit.

II.4.2.2.2. 2. Etude de la surface de l'architecture « pipelinée »

Si les codes utilisés pour l'étude de la surface et du débit du codeur sont caractérisés par des tailles voisines, il n'en est pas de même pour la taille des portes à logique majoritaires qui varie entre 2 et 9 (fig II-65)

code	k	n	Taille de la porte à logique majoritaire(nombre d'entrées)
DSCC(72,44)	44	72	9
DTI(6,3,21)	60	46	2
EG(0,3,2)	60	34	8

Fig II-65. Tailles des différentes portes à logique majoritaires

Les figures II-66 a et b regroupent les résultats sur la variation en fonction du parallélisme de la surface et du débit pour le circuit de décodage des codes considérés. Le débit et la surface de référence sont les valeurs de l'architecture série. Cette architecture série comprend la FIFO (n registres) nécessaire pour la synchronisation de la correction. Il faut aussi noter que pour un code, le degré de parallélisme 5 n'a pas été synthétisé pour respecter la contrainte (n/p) entier.

Comme on peut le voir sur les deux graphes, le gain de l'architecture parallèle est similaire au gain réalisé pour le codeur parallèle. Le débit est fortement augmenté alors que la surface croit très peu. Alors que la variation lente de la surface du codeur parallèle peut facilement s'expliquer, il n'en est pas de même pour le décodeur parallèle où la duplication des portes à logiques majoritaires et des matrices W devrait contribuer fortement à l'augmentation de la surface du décodeur. En réalité, comme pour la porte à logique majoritaire, le synthétiseur remet toutes les équations combinatoires à plat afin d'effectuer une réduction des équations.

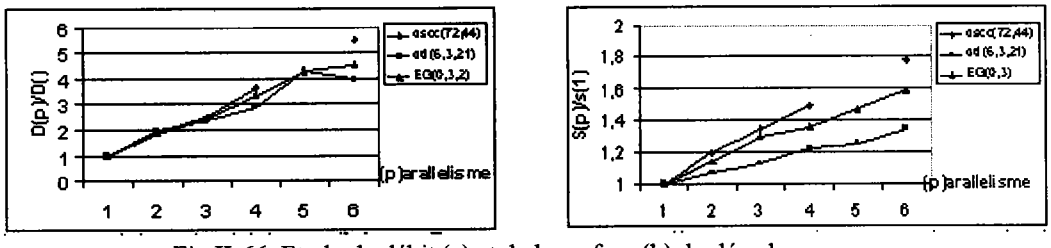


Fig II-66. Etude du débit (a) et de la surface (b) du décodeur

II.4.2.3. Circuit de décodage.

Comme pour le codeur, le circuit de décodage a été entièrement synthétisé sur un circuit de la famille des Flex 10K. Pour un parallélisme de 3, les caractéristiques du circuit sont les suivantes :

Circuit utilisé	: Flex 10K30RC208-3
Surface	: 1546 Lce sur les 1728 Lce disponibles
Fréquence maximum en entrée	: 45,87 Mhz
Fréquence maximum interne	: 31,25 Mhz
Fréquence maximum de sortie	: 30,25 Mhz

Les conclusions obtenues pour l'étude du codeur sont encore valables pour le décodeur. Les circuits d'entrées et de sorties sont les facteurs limitatifs du circuit. De même, les fréquences des différents étages permettent de traiter des flots continus de données.

II.5. Conclusions et perspectives

Après une présentation des différents codes à logique majoritaire et la description algorithmique de leur fonctionnement, une étude architecturale a permis la synthèse générique des circuits parallèles de codage et de décodage. Ceux ci sont capables de traiter des flots continus de données avec pour seule contrainte que le rapport entre la taille du mot code et le degré de parallélisme soit entier.

De plus, la fréquence maximum de fonctionnement pour ces circuits est de l'ordre de 30 Mhz (la fréquence maximum de fonctionnements des circuits Flex10K est de 125 Mhz) ce qui constitue un bon résultat.

En outre, les facteurs limitatifs de ces circuits sont les étages d'entrée et de sortie contrairement au modèle série. Une réécriture de ces étages utilisée conjointement avec une synthèse sur ASIC (contrôle du routage) devrait permettre d'augmenter les fréquences de fonctionnement et d'affiner les études sur la variation de la surface de ces circuits.

Enfin deux types d'utilisation existent pour ces deux circuits. Ils peuvent être utilisés directement dans des applications à haut débit traitant des flots continus de données (codeur et décodeur complet) ou bien ils peuvent être intégrés comme fonction (cœur du codeur et du décodeur) dans des DSP dédiés.

Chapitre III.

Le code Reed-Solomon (127,k,d) avec effacements

Dans ce chapitre, nous allons nous intéresser au code de Reed-Solomon et plus particulièrement au code Reed-Solomon avec effacements. Alors que dans le chapitre précédent, nous nous focalisions sur l'architecture permettant d'obtenir le meilleur débit possible, ici nous allons nous intéresser à l'amélioration de la capacité de correction (c'est à dire le nombre maximum d'erreurs pouvant être corrigées). Dans un premier temps, nous détaillerons la théorie des codes Reed-Solomon. Ensuite nous étudierons une technique de marquage développé [11] pour les codes RS et permettant de traiter des effacements. Enfin afin de valider la conception matérielle d'un codeur/décodeur RS (127,k,d) avec effacement (algorithme de codage et technique de marquage), une étude de faisabilité a été effectuée sur un cas particulier : le code RS (127,121,7) avec effacements.

III.1. La théorie des codes Reed-Solomon

III.1.1. Présentation du code

III.1.1.1. Construction du polynôme générateur

Soit m un entier positif strictement supérieur à 2. On pose t un entier positif inférieur à 2^m-1 et α un élément primitif de GF(2^m). On appelle g(x) le polynôme de degré le plus faible ayant les valeurs $\{\alpha,\alpha^2,..,\alpha^{2t}\}$ comme racines.

$$G(x) = \prod_1^{2t}(\alpha^i - x) \quad (3.1)$$

Ce polynôme définit une classe de code linéaire cyclique : Les codes Reed-Solomon [34]. Ils sont capables de corriger $\lfloor(2t+1)/2\rfloor$ erreurs dans un bloc de 2^m-1 bits.

Les codes de Reed-Solomon étant des codes linéaires cycliques, le mot de code est un multiple du polynôme générateur. Par conséquent, si le mot reçu n'est pas un multiple de g(x), on sait que ce n'est pas un mot de code. Cependant, le mot de code présente d'autres caractéristiques utiles pour le décodage. Nous allons les examiner.

III.1.1.2. Caractéristique du mot code

On considère maintenant un polynôme P de degré n-1 dont les coefficients appartiennent à GF(2). Si $\{\alpha,\alpha^2,..,\alpha^{2t}\}$ sont des racines de P(x), alors, de manière évidente, P(x) est aussi divisible par g(x). On peut donc en conclure que P est un mot de code.

De manière réciproque, si P est un mot de code, il est divisible par g(x). Donc $\{\alpha,\alpha^2,..,\alpha^{2t}\}$ sont les racines de P(x). On a alors démontré l'équivalence suivante :

$$P \text{ est un mot de code} \Leftrightarrow P(\alpha^i) = 0 \; \forall i \in \{1,...,2t\} \quad (3.2)$$

Cette relation peut s'écrire matriciellement par :

$$\text{P est un mot de code} \Leftrightarrow [v_0,\cdots,v_{n-1}]\begin{bmatrix}1\\ \alpha^i\\ \vdots\\ \alpha^{(n-1)i}\end{bmatrix}=[0]\,\forall i\in\{1,\cdots,2t\} \quad (3.3)$$

Appelons respectivement H et S les quantités définies par :

$$H=\begin{bmatrix}1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1}\\ \vdots & & & & \\ \vdots & & & & \\ 1 & \left(\alpha^{2t}\right) & \left(\alpha^{2t}\right)^2 & \cdots & \left(\alpha^{2t}\right)^{n-1}\end{bmatrix} \quad (3.4)$$

$$S = V.H^t \quad (3.5)$$

D'après (3.3) et (3.5), il s'ensuit que :

$$\text{P est un mot de code} \Leftrightarrow S = V.H^t = 0 \quad (3.6)$$

La matrice H (matrice de contrôle) permet donc de caractériser et de vérifier que le mot utilisé est bien un mot de code (détection des erreurs).

III.1.1.3. Exemple

On considère le code linéaire cyclique Reed-Solomon défini dans $GF(2^3)$ pouvant corriger au plus 2 (t = 2) erreurs. Les paramètres de ce code sont :

$n = 2^p-1 = 2^3-1 = 7$,

$k = 2^p-1-2t = 8-1-4 = 3$.

Le polynôme générateur de ce code vérifie la formule :

$$G(x) = \prod_0^3\left(x-\alpha^i\right) = x^4+\alpha^2x^3+\alpha^5x^2+\alpha^5x+\alpha^6$$

En utilisant les résultats du paragraphe précédent, la matrice H est égale à :

$$H = \begin{bmatrix} 1 & (\alpha) & (\alpha)^2 & (\alpha)^3 & (\alpha)^4 & (\alpha)^5 & (\alpha)^6 \\ 1 & (\alpha^2) & (\alpha^2)^2 & (\alpha^2)^3 & (\alpha^2)^4 & (\alpha^2)^5 & (\alpha^2)^6 \\ 1 & (\alpha^3) & (\alpha^3)^2 & (\alpha^3)^3 & (\alpha^3)^4 & (\alpha^3)^5 & (\alpha^3)^6 \\ 1 & (\alpha^4) & (\alpha^4)^2 & (\alpha^4)^3 & (\alpha^4)^4 & (\alpha^4)^5 & (\alpha^4)^6 \end{bmatrix} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \\ 1 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha & \alpha^4 \\ 1 & \alpha^4 & \alpha & \alpha^5 & \alpha^2 & \alpha^6 & \alpha^3 \end{bmatrix}$$

Supposons maintenant que l'on veuille coder le mot $M(x) = x^2+\alpha x+\alpha^2$. De même que pour les codes à logique majoritaire, on calcule le reste de $x^4M(x)$ par $g(x)$.

$$\begin{array}{lllll} x^6 & \alpha x^5 & \alpha^2 x^4 & & \\ -x^6 & -\alpha^2 x^5 & -\alpha^5 x^4 & -\alpha^5 x^3 & -\alpha^6 x^2 \\ \hline & \alpha^4 x^5 & \alpha^3 x^4 & \alpha^5 x^3 & \alpha^6 \\ & -\alpha^4 x^5 & -\alpha^6 x^4 & -\alpha^9 x^3 & -\alpha^9 x^2 & -\alpha^{10} x \\ \hline & & \alpha^4 x^4 & \alpha^3 x^3 & \alpha^0 x^2 & \alpha^3 x \\ & & -\alpha^4 x^4 & -\alpha^6 x^3 & -\alpha^9 x^2 & -\alpha^9 x^1 & \alpha^{10} \\ \hline & & & \alpha^4 x^3 & \alpha^6 x^2 & \alpha^5 x & \alpha^3 \end{array} \qquad \begin{array}{|lllll} x^4 & \alpha^2 x^3 & \alpha^5 x^2 & \alpha^5 x & \alpha^6 \\ \hline x^2 + \alpha^4 x & +\alpha^4 & & & \end{array}$$

Le mot de code P est alors : $[1\ \alpha\ \alpha^2\ \alpha^4\ \alpha^6\ \alpha^5\ \alpha^3]$

On peut alors vérifier que (P est un mot de code $\Leftrightarrow$ $S = V.H^t = 0$).

Le code Reed-Solomon étant un code linéaire cyclique, l'algorithme de codage utilisé est analogue à celui des codes à logique majoritaire (calcul de reste). Cependant, la difficulté provient des opérations qui se font dans les corps de Galois. Il est donc primordial de bien choisir le multiplieur dans le corps de Galois utilisé pour qu'il n'ait pas un impact négatif sur le débit.

III.1.2. Algorithme de décodage

III.1.2.1. Evaluation de S

La matrice H (formule (3.6)) caractérise les mots codes mais est difficilement implantable dans des circuits (toutes les opérations ont lieu dans un corps de Galois). Il faut donc trouver une écriture équivalente de S mais sans utiliser de matrice.

Or d'après le théorème d'Euclide, il existe deux polynômes v_i et q_i tels que :

$$P(x) = v_i(x)\,(1-\alpha^{-i}x)+q_i(x) \quad (3.7)$$

Par conséquent,

$$S_i = P(\alpha^i) = v_i(\alpha^i)\,(1-\alpha^{-i}\alpha^i)+q_i(\alpha^i) = q_i(\alpha^i) \quad (3.8)$$

Les coefficients de S peuvent donc être évalués à partir des restes des divisions de la donnée reçue par les polynômes minimaux. Cependant, si la détection des erreurs est possible, encore faut-il les localiser et les corriger.

III.1.2.2. Localisation des erreurs et correction

III.1.2.2.1. Localisation des erreurs

Supposons que le vecteur reçu P soit composé d'un mot de code C et d'une erreur E. On peut écrire :

$$P(\alpha^i) = E(\alpha^i)+C(\alpha^i)=E(\alpha^i) \text{ car C est un mot de code}$$

On en déduit alors que :

$$S_i = E(\alpha^i) \;\forall i \in \{1,\cdots,2t\} \quad (3.9)$$

Supposons qu'il y ait ν erreurs. Le vecteur E peut alors s'écrire : $E(x) = \sum_{j=1}^{\nu}\left(X^{L_j}\right)$

En utilisant la formule (3.8), on a alors :

$$S_i = \sum_{j=1}^{\nu}\left(\alpha^{L_j}\right)^i = \;\forall i \in \{1,\cdots,2t\} \quad (3.10)$$

A partir des valeurs de $\beta_j = \alpha^{L_j}$, on peut définir un polynôme σ (x) appelé polynôme localisateur tel que :

$$\sigma(x) = \sum_{i=0}^{v} \sigma_i x^i = \prod_{i=1}^{v} (1 - \beta_i x) = \prod_{i=1}^{v} (1 - \alpha^{L_i} x) \quad (3.11)$$

Les zéros du polynôme localisateur donnant les inverses des positions des erreurs, l'examen de $\sigma(\alpha^i)$ permet de trouver les positions des erreurs. Ayant localisé les erreurs, il faut maintenant les évaluer.

III.1.2.2.2. Correction des erreurs

On considère maintenant le produit $S(x)\sigma(x)$. On a

$$S(x)\sigma(x) = \sigma(x) \sum_{i=1}^{v} e_i \left(\frac{1 - (\beta_i x)^{2t}}{1 - \beta_i x} \right) = \left(\sum_{i=1}^{v} e_i \frac{\sigma(x)}{1 - \beta_i x} \right) - x^{2t} \left(\sum_{i=1}^{v} e_i (\beta_i)^{2t} \frac{\sigma(x)}{1 - \beta_i x} \right)$$

Par conséquent,

$$S(x)\sigma(x) \equiv \omega(x)[x^{2t}] \quad (3.12).$$

C'est l'équation clé de décodage.

La quantité $\omega(x) = \sum_{i=1}^{v} e_i \frac{\sigma(x)}{1 - \beta_i x}$ est appelée polynôme évaluateur de P.

Par construction de σ, si $\sigma(\alpha^{-i})$ est différent de zéro la position examinée i n'est pas erronée. Par contre si cette quantité est égale à zéro alors il y une erreur et il faut l'évaluer.

En dérivant la formule (3.12) et en remplaçant x par α^{-i}, on obtient l'évaluation de l'erreur :

$$e_i = -\alpha^i \frac{\omega(\alpha^{-i})}{\sigma'(\alpha^{-i})} \quad (3.13)$$

Plusieurs méthodes ont été développées pour calculer le couple (ω,σ) [35],[36]. Dans notre cas, nous allons uniquement nous intéresser à l'algorithme de Berlekamp.

L'avantage de cet algorithme séquentiel est de pouvoir être modifié pour tenir compte des effacements comme nous le verrons plus loin.

III.1.2.3. Algorithme de Berlekamp.

L'algorithme de Berlekamp consiste à trouver à chaque itération i comprise entre 0 et (2t) un couple (ω_i,σ_i) vérifiant les relations suivantes :

$$\begin{cases} \sigma_j S \equiv \omega_j \left[x^j\right] \\ \sigma(0) = 1 \\ \text{Max}(\deg(\sigma_j), 1 + \deg(\omega_j)) \text{ est minimal} \end{cases}$$

A la fin de l'algorithme (étape 2t) et si le nombre d'erreurs ne dépasse pas (t), le couple $(\omega_{2t},\sigma_{2t})$ correspond au localisateur et à l'évaluateur de E.

Pour trouver l'algorithme, on va considérer deux ensembles Ej et Ej* défini par :

$$Ej = \left\{(\sigma,\omega)/\sigma\sigma(x)S(x \equiv \omega(x)\left[x^j\right]\right.$$

$$E_j^* = \left\{(\sigma,\omega) \in E_j / \sigma(0) = 1 \text{ et } \deg(\sigma,\omega) = d_j = \min(\deg(\sigma,\omega))\right\}$$

avec deg(a,b) égal à Max (deg(a),1+deg(b)).

On peut alors démontrer les deux propriétés suivantes :

Si $(2d_j > j)$ ou $(2d_j \leq j$ et $(\sigma_j,\omega_j) \in E_{j+1})$ alors $d_{j+1}=d_j$ (3.14)

Si $2d_j \leq j$ alors $d_{j+1} = j+1-d_j$

En effet posons (σ_j,ω_j) des éléments E_j*. On peut écrire l'équation suivante :

$$\sigma_j(x)S(x) \equiv \omega(x) + \Delta_j \left[x^{j+1}\right] \quad (3.15)$$

Deux cas peuvent alors de produire. Si Δ_j est égal à zéro alors le couple (σ_j,ω_j) appartient à l'ensemble E_{j+1}. Par contre, si Δ_j est différent de zéro, on suppose qu'il

existe un autre couple (σ'_j,ω'_j) tel que $(\sigma_j-\Delta_j\sigma'_j\ ,\ \omega_j-\Delta_j\omega'_j)$ appartient à l'ensemble E_{j+1}.

On vérifie alors immédiatement que :

$2d_j>j \Rightarrow d_{j+1}= d_j$

$2d_j \leq j \Rightarrow d_{j+1} = j+1-d_j$.

Le principal problème reste alors de trouver un couple (σ'_j,ω'_j) vérifiant les conditions précédentes. Pour le construire, on va utiliser un processus itératif.

En supposant connu à l'instant j le couple (σ'_j,ω'_j), deux cas doivent être considéré :

Si $d_{j+1}=d_j$, une simple multiplication par x des polynômes du couple (σ'_j,ω'_j) répond au problème.

Si $d_{j+1}>d_j$,on démontre [34]que le couple $(x\Delta_j^{-1}\sigma_j,\ x\Delta_j^{-1}\omega_j)$ convient.

L'algorithme de recherche de Berlekamp [21] utilisant les propriétés décrites précédemment est représenté à la figure suivante :

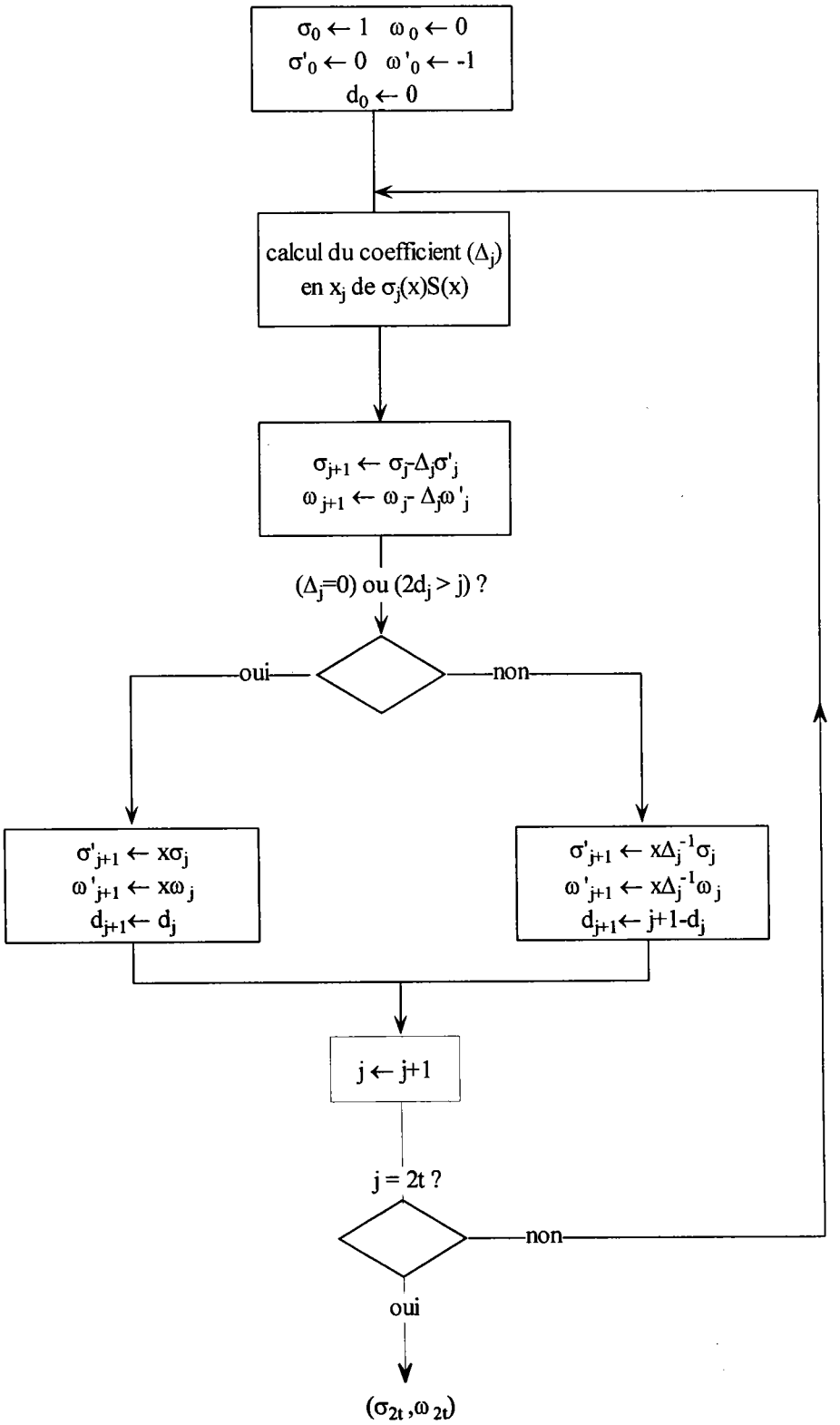


Fig III-1. Algorithme de Berlekamp

III.1.2.4. Exemple.

On réutilise l'exemple du paragraphe III.1.1.3. Supposons que l'on ait deux erreurs en position 1 et 4 d'amplitude respective (α^2) et (α). Le mot de code [1 α α^2 α^4 α^6 α^5 α^3] devient C' = [1 α α^4 α^4 α^6 α^3 α^3].

En premier lieu, nous allons calculer le syndrome $S(x) = \sum_1^4 C'(\alpha^{i-1})x^i = \sum_1^4 S_i x^i$

On a
$$\begin{cases} S_1 = C'(1) = 1+\alpha^1+\alpha^4+\alpha^4+\alpha^6+\alpha^3+\alpha^3 = \alpha^4 \\ S_2 = C'(\alpha) = (\alpha)^6+\alpha(\alpha)^5+\alpha^2(\alpha)^4+\alpha^4(\alpha)^3+\alpha^6(\alpha)^2+\alpha^3(\alpha)^1+\alpha^3 = \alpha^2 \\ S_3 = C'(\alpha^2) = (\alpha^2)^6+\alpha(\alpha^2)^5+\alpha^2(\alpha^2)^4+\alpha^4(\alpha^2)^3+\alpha^6(\alpha^2)^2+\alpha^3(\alpha^2)^1+\alpha^3 = \alpha \\ S_4 = C'(\alpha^3) = (\alpha^3)^6+\alpha(\alpha^3)^5+\alpha^2(\alpha^3)^4+\alpha^4(\alpha^3)^3+\alpha^6(\alpha^3)^2+\alpha^3(\alpha^3)^1+\alpha^3 = \alpha \end{cases}$$

Par conséquent, le syndrome S est égal au polynôme $\alpha^4+\alpha^2x+\alpha x^2+\alpha x^3$.

Connaissant le syndrome, nous pouvons appliquer l'algorithme de Berlekamp pour déterminer le polynôme localisateur d'erreur $\sigma(x)$. Comme t est égal à 2, 4 cycles sont nécessaire pour l'obtenir.

Pour j = 0, on a :

$\sigma_0 = 1$, $\omega_0 = 0$, $\sigma'_0 = 0$, $\omega'_0 = -1$, d0 = 0.

Par conséquent, $\sigma_0.S(x) = S(x) = \alpha^4+\alpha^2x+\alpha x^2+\alpha x^3$. D'où $\Delta_0 = \alpha^4$.

On a alors :

$\sigma_1 = \sigma_0 - \Delta_0\sigma'_0 = 1$ $\omega_1 = \omega_0 - \Delta_0\omega'_0 = 0 + \alpha^4 = \alpha^4$.

Comme ($\Delta_0 \neq 0$) et ($2d_0 \leq 0$) σ'_1, ω'_1 et d_1 prennent les valeurs :

$\sigma'_1 = x\Delta_0^{-1}\sigma_0 = \alpha^3x$ $\omega'_1 = x\Delta_0^{-1}\omega_0 = 0$ $d_1 = j+1-d_j = 1$

Pour j = 1,

$\sigma_1.S(x) = S(x) \Rightarrow \Delta_1 = \alpha^2$

$\sigma_2 = \sigma_1 - \Delta_1\sigma'_1 = 1-\alpha^2\alpha^3x = 1+\alpha^5x$

$\omega_2 = \omega_1 - \Delta_1\omega'_1 = \alpha^4$

$2d_1 = 2 > 1\ (=j) \Rightarrow \sigma'_2 = x\sigma_1 = \alpha^3x^2$ $\omega'_2 = x\omega_1 = 0$ $d_2 = d_1 = 1$

<u>j = 2</u>

$\sigma_2.S(x) = (1+\alpha^5x)(\alpha^4 + \alpha^2x + \alpha x^2 + \alpha x^3) = \alpha^4 + \alpha^3x^2 + \alpha^5x^3 + \alpha^6x^4 \Rightarrow \Delta_2 = \alpha^3$

$\sigma_3 = \sigma_2 - \Delta_2\sigma'_2 = 1- \alpha^5x -\alpha^3\alpha^3x^2 = 1- \alpha^5x-\alpha^6x^2$

$\omega_3 = \omega_2 - \Delta_2\omega'_2 = \alpha^4+\alpha^30 = \alpha^4$

$2d_2 = 2 \leq j \Rightarrow$

$\sigma'_3 = x\Delta_2^{-1}\sigma_2 = x\alpha^4(1+\alpha^5x) = \alpha^4x+\alpha^2x^2$

$\omega'_3 = x\Delta_2^{-1}\omega_2 = x\alpha^4(\alpha^4) = \alpha x$

$d_3 = j+1-d_j = 2+1-1 = 2$

<u>j = 3</u>

$\sigma_3.S(x) = (1- \alpha^5x-\alpha^6x^2)(\alpha^4 + \alpha^2x + \alpha x^2 + \alpha x^3) = \alpha^4 + \alpha^6x^3 + \alpha^5x^4 + x^5 \Rightarrow \Delta_3 = \alpha^6$

$\sigma_4 = \sigma_3 - \Delta_3\sigma'_3 = (1- \alpha^5x-\alpha^6x^2)- \alpha^6(\alpha^4x-\alpha^2x^2) = 1+\alpha^2x+\alpha^5x^2$

$\omega_4 = \omega_3 - \Delta_3\omega'_3 = \alpha^4-\alpha^6(\alpha x) = \alpha^4-x$

Les polynômes localisateur $\sigma(x)$ et évaluateur d'erreur $\omega(x)$ sont donc égaux à :

$\sigma(x) = \sigma_4(x) = 1+\alpha^2x+\alpha^5x^2$

$\omega(x) = \omega_4(x) = \alpha^4-x$

La recherche des zéros de $\sigma(x)$ nous permet de déterminer les positions des erreurs dans le mot reçu. On a :

$$\begin{cases} \sigma(1) = 1 + \alpha^2 + \alpha^5 = \alpha \\ \sigma(\alpha^{-1}) = 1 + \alpha^2\alpha^{-1} + \alpha^5\alpha^{-2} = 1 + \alpha + \alpha^3 = 0 \\ \sigma(\alpha^{-2}) = 1 + \alpha^2\alpha^{-2} + \alpha^5\alpha^{-4} = \alpha \\ \sigma(\alpha^{-3}) = 1 + \alpha^2\alpha^{-3} + \alpha^5\alpha^{-6} = 1 \\ \sigma(\alpha^{-4}) = 1 + \alpha^2\alpha^{-4} + \alpha^5\alpha^{-8} = 1 + \alpha^5 + \alpha^4 = 0 \\ \sigma(\alpha^{-5}) = 1 + \alpha^2\alpha^{-5} + \alpha^5\alpha^{-10} = 1 + \alpha^4 + \alpha^2 = \alpha^3 \\ \sigma(\alpha^{-6}) = 1 + \alpha^2\alpha^{-6} + \alpha^5\alpha^{-12} = 1 + \alpha^3 + 1 = \alpha^3 \end{cases}$$

Les seules racines du polynôme $\sigma(x)$ sont α-1 et α-4. Il y a donc :2 erreurs en position 1 et 4. Connaissant ces positions, la formule (3.13) nous permet d'évaluer les erreurs :

$$e_1 = -\alpha^1 \frac{\omega(\alpha^{-1})}{\sigma'(\alpha^{-1})} = -\alpha^1 \frac{\alpha^4 - \alpha^{-1}}{\alpha^2} = -\frac{\alpha^4 + \alpha^6}{\alpha} = -\frac{\alpha^3}{\alpha} = -\alpha^2$$

$$e4 = -\alpha^4 \frac{\omega(\alpha^{-4})}{\sigma'(\alpha^{-4})} = -\alpha^4 \frac{\alpha^4 - \alpha^{-4}}{\alpha^2} = -\alpha^2 (\alpha^4 + \alpha^3) = -\alpha$$

Par conséquent, le mot code C(x) initial est égal à :

$$C(x) = (x^6+\alpha x^5+\alpha^4x^4+\alpha^4x^3+\alpha^6x^2+\alpha^3x+\alpha^3) - (\alpha^2x+\alpha x^4)$$

C'est à dire :

$$C(x) = (x^6+\alpha x^5+\alpha^2x^4+\alpha^4x^3+\alpha^6x^2+\alpha^5x+\alpha^3)$$

Cela correspond bien au mot de code envoyé.

Bien que permettant de connaître le couple (σ,ω) en 2t cycles de traitement, cet algorithme est séquentiel et ne se prête pas à un traitement parallèle. De plus, comme pour l'algorithme d'origine), le point le plus compliqué consiste à implanter un circuit de traitement dans les corps de Galois (calcul de la multiplication de deux polynômes pour trouver le coefficient Δ_i).

III.1.3. Prise en compte des effacements

III.1.3.1. Position du problème et première conséquence

Dans les paragraphes précédents, nous avons traité des erreurs dont ni la position ni la valeur ne sont connues. Toutefois, on peut supposer que, grâce à un moyen de marquage des symboles, on puisse connaître la position de quelques symboles erronés. Seule la valeur de l'erreur est inconnue. De tels symboles sont appelés « effacements ».

Les « errata » (erreurs + effacements) pouvant corrompre un mot code sont alors constitués d'une part par un certain nombre d'erreurs t' et d'autre part par des effacements e'.

Connaissant la position des effacements, on peut définir un polynôme localisateur σ_{eff} et un polynôme évaluateur ω_{eff} d'effacements :

$$\sigma_{eff} = \prod_{i=1}^{e'} \left(1 - \alpha^{L_i} x\right) \quad (3.16)$$

$$\omega_{eff} = \sum_{i=1}^{e'} p_i \frac{\sigma_{eff}(x)}{1 - \alpha^{L_i} x} \quad (3.17)$$

avec L_i la position des effacements.

On peut de même définir un polynôme localisateur σ_{err} et un polynôme évaluateur d'erreurs ω_{err} :

$$\sigma_{err} = \prod_{j=1}^{t'} \left(1 - \alpha^{L_j} x\right) \quad (3.18)$$

$$\omega_{err} = \sum_{j=1}^{t'} p_i \frac{\sigma_{err}(x)}{1 - \alpha^{L_j} x} \quad (3.19)$$

Les polynômes localisateur et évaluateur des errata σ est alors égal à :

$$\sigma(x) = \sigma_{err}(x)\sigma_{eff}(x) \quad (3.20)$$

$$\omega(x) = \sigma_{eff}(x)\omega_{err}(x) + \sigma_{err}(x)\omega_{eff}(x) \quad (3.21)$$

D'où :

$$S(x)\sigma(x) = \left(\sum_{i=1}^{e'} e_{L_i} \frac{1-(\beta_i x)^{2t}}{1-\beta_i x} + \sum_{j=1}^{t'} \frac{1-(\beta_j x)^{2t}}{1-\beta_j x} \right) \sigma_{err}(x)\sigma_{eff}(x) \quad (3.22)$$

C'est à dire :

$$S(x)\sigma(x) \equiv \omega(x)[x^{2t}] \quad (3.23)$$

La formule (3.23) est similaire à (3.12) mais on ne connaît pas σ_{err}. Une manière de résoudre cette équation est d'essayer de se rapporter au cas sans effacement. C'est la méthode développée par Forney [21].

III.1.3.2. Méthode de Forney

Considérons un couple (σ,ω) vérifiant les propriétés suivantes :

$\sigma(0) = l$, deg $(\omega) < l$, deg $(\sigma) \leq l$,

σ_{eff} divise σ,

PGCD $(\sigma_{err}, \omega) = 1$,

$S(x)\sigma(x) \equiv \omega(x)[x^{2t}]$

On peut démontrer que tout couple du type précédent peut être converti en un autre couple (β,γ) tels que

$\beta(0) = 1$, deg $(\gamma) <$ l-e', deg $(\sigma) \leq$ l-e',

PGCD $(\beta,\gamma) = 1$,

$S(x)T(x) \equiv \gamma(x)[x^{2t-e'}]$ avec T défini par $\sigma_{eff}(x)S(x) = \varepsilon(x) + x^s T(x)$

Connaissant $\varepsilon(x)$ et $\sigma_{eff}(x)$, On revient au cas précédent (RS sans effacement) et on peut calculer les polynômes localisateur d'erreurs $\beta(x)$ $(=\sigma(x)/\sigma_{eff}(x))$ et évaluateur d'erreurs $\gamma(x)$ en utilisant l'algorithme de Berlekamp.

L'accès aux polynômes β et γ nous permet :

- de connaître la position des erreurs dans le mot reçu

- d'évaluer l'amplitude des erreurs $e_i = -\frac{\alpha^{i(1-e')}\gamma(\alpha^{-i})}{\sigma_{eff}(\alpha^{-i})\beta'(\alpha^{-i})}$ (3.24)

- d'évaluer l'amplitude des effacements $e_i = -\frac{\alpha^{i(1-e')}\gamma(\alpha^{-i})}{\beta(\alpha^{-i})\sigma'_{eff}(\alpha^{-i})} - \alpha i \frac{\varepsilon(\alpha^{-i})}{\sigma'_{eff}(\alpha^{-i})}$ (3.25)

III.1.3.2. Algorithme de Berlekamp modifié

L'algorithme de Berlekamp a été modifié afin de tenir compte des effacements. Il permet de calculer σ_{eff} puis σ. On peut alors en déduire σ_{err} ($\sigma = \sigma_{eff} \times \sigma_{err}$).

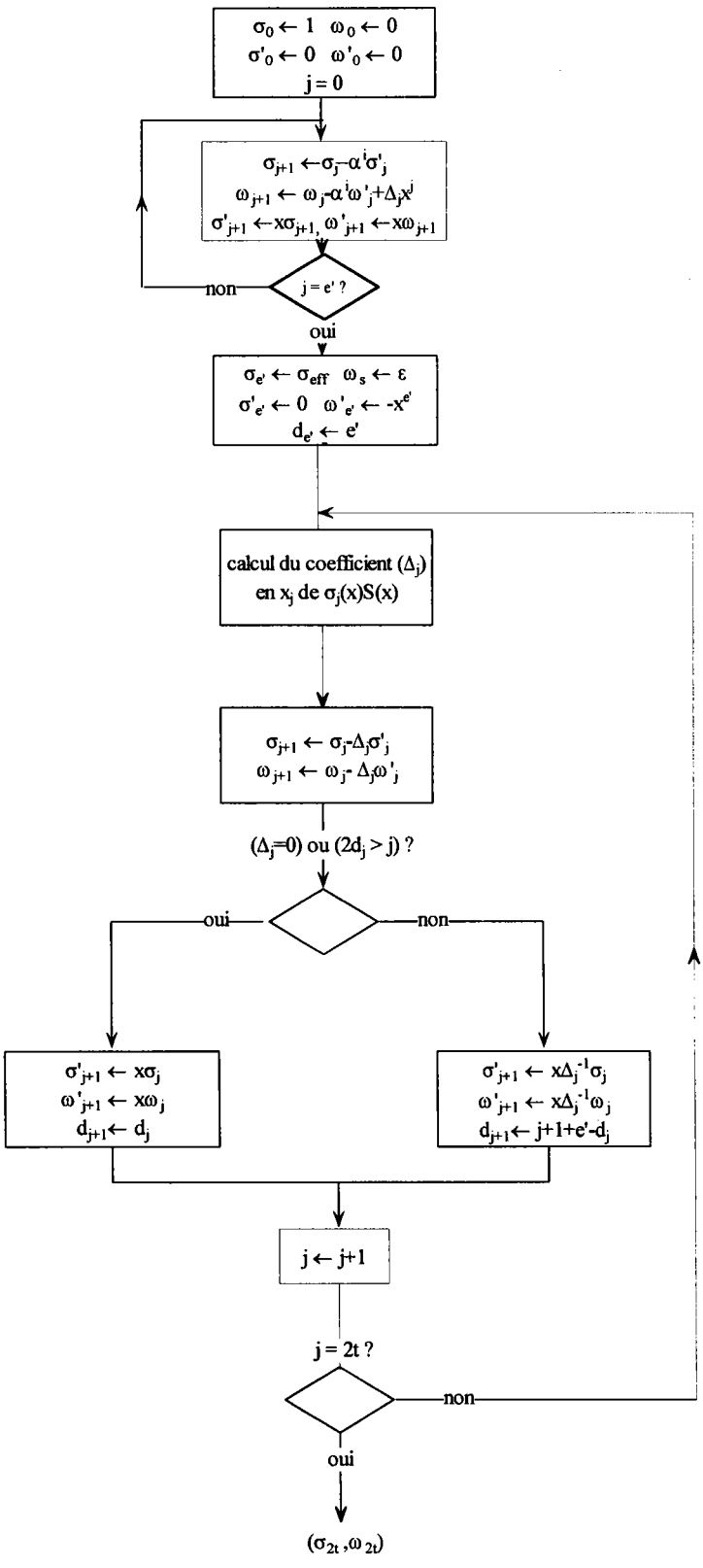


Fig III-2. Algorithme modifié de Berlekamp-Massey

III.1.3.3. Exemple

On utilise toujours le même exemple que dans le cas sans effacement. On ne considère plus qu'une seule erreur en position 6 d'amplitude α^6. On suppose par contre que l'on peut détecter deux effacements en position 1 et 4 d'amplitude α et α^2. Le mot reçu est alors le suivant :

$C'(x) = \alpha^2x^6+\alpha x^5+\alpha^4x^4+\alpha^4x^3+\alpha^6x^2+\alpha^3x+\alpha^3$

Comme dans le cas sans effacement, on va calculer le syndrome.

$$\begin{cases} S_1 = C'(1) = \alpha^2+\alpha^1+\alpha^4+\alpha^4+\alpha^6+\alpha^3+\alpha^3 = \alpha^3 \\ S_2 = C'(\alpha) = \alpha^2(\alpha)^6+\alpha^1(\alpha)^5+\alpha^4(\alpha)^4+\alpha^4(\alpha)^3+\alpha^6(\alpha)^2+\alpha^3(\alpha)+\alpha^3 = \alpha^3 \\ S_3 = C'(\alpha^2) = \alpha^2(\alpha^2)^6+\alpha^1(\alpha^2)^5+\alpha^4(\alpha^2)^4+\alpha^4(\alpha^2)^3+\alpha^6(\alpha^2)^2+\alpha^3(\alpha^2)+\alpha^3 = \alpha^2 \\ S_4 = C'(\alpha^3) = \alpha^2(\alpha^3)^6+\alpha^1(\alpha^3)^5+\alpha^4(\alpha^3)^4+\alpha^4(\alpha^3)^3+\alpha^6(\alpha^3)^2+\alpha^3(\alpha^3)+\alpha^3 = 1 \end{cases}$$

Le Syndrome S(x) est donc égal à : $\alpha^3+\alpha^3x+\alpha^2x^2+x^3$

Connaissant le syndrome, on peut appliquer la première partie de l'algorithme de Berlekamp-Massey. A l'initialisation, on a :

j = 0,

$\sigma_0 = 1$ $\omega_0 = 0$ $\sigma'_0 = x$ $\omega'_0 = 0$

On a alors $\sigma_0(x)S(x) = S(x) = \alpha^3+\alpha^3x+\alpha^2x^2+x^3$. D'où $\Delta_0 = \alpha^3$.

Par conséquent, on a :

$\sigma_1 = (\sigma_0 - \alpha\sigma'_0) = 1-\alpha x$ $\sigma'_1 = x\sigma_1 = x-\alpha x^2$

$\omega_1 = (\omega_0 - \alpha\omega'_0 + \Delta_0x_0) = 0-\alpha 0+\alpha^3 = \alpha^3$ $\omega'_1 = x\omega_1 = x\alpha^3$

j = 1,

$\sigma_1(x)S(x) = (1+\alpha x)(\alpha^3 + \alpha^3 x + \alpha^2 x^2 + x^3) = \alpha^3 + \alpha^6 x + \alpha x^2 + \alpha x^3 + \alpha x^4$. D'où

$\Delta_1 = \alpha^6$

Par conséquent :

$\sigma_2 = (\sigma_1 - \alpha^4 \sigma'_1) = (1+\alpha x)+\alpha^4(x - \alpha x^2) = 1+\alpha^2 x+\alpha^5 x^2$

$\sigma'_2 = x\sigma_2 = x+\alpha^2 x^2+\alpha^5 x^3$.

$\omega_2 = (\omega_1 - \alpha^4 \omega'_1 + \Delta_1 x^1) = \alpha^3+\alpha^4(x\alpha^3)+ \alpha^6 x = \alpha^3+\alpha^2 x$

$\omega'_2 = x\omega_2 = \alpha^3 x+\alpha^2 x^2$

Comme le nombre d'effacement est de 2, on peut écrire les égalités :

$$\sigma_{eff}(x) = \sigma_2(x) = 1+\alpha^2 x+\alpha^5 x^2$$

$$\varepsilon(x) = \omega_2(x) = \alpha^3+\alpha^2 x$$

Maintenant que l'on connaît les polynômes localisateur et évaluateur d'effacements, on peut calculer les polynômes localisateur et évaluateur d'erreurs.

j = 2. On pose

$\sigma_2 = 1+\alpha^2 x+\alpha^5 x^2$ $\omega_2 = \alpha^3+\alpha^2 x$ $\sigma'_2 = 0$ $\omega'_2 = x^2$ $d_2 = 2$

On a alors :

$\sigma_2(x)S(x)=(1+\alpha^2 x+\alpha^5 x^2+x^3)(\alpha^3 + \alpha^3 x + \alpha^2 x^2 + x^3)$ c'est à dire

$\sigma_2(x)S(x)=\alpha^3 + \alpha^2 x + x^2 + \alpha^6 x^3 + \alpha^6 x^4 + \alpha^5 x^5$.

D'où $\Delta_2 = 1$

On a alors :

$\sigma_3 = \sigma_2 - \Delta_2 \sigma'_2 = 1+\alpha^2 x+\alpha^5 x^2$ $\omega_3 = \omega_2 - \Delta_2 \omega'_2 = \alpha^3+\alpha^2 x+x^2$

Or $2d_2 = 4 \leq (j+2)$. D'où :

$\sigma'_3 = x\Delta_2^{-1}\sigma_2 = x+\alpha^2 x^2+\alpha^5 x^3$ $\omega'_3 = x\Delta_2^{-1}\omega_2 = \alpha^3 x+\alpha^2 x^2$ $d_3 = 3$

j = 3

$\sigma_2(x)S(x) = \alpha^3 + \alpha^2 x + x^2 + \alpha^6 x^3 + \alpha^6 x^4 + \alpha^5 x^5 =$. D'où $\Delta_3 = \alpha^6$

d'où :

$\sigma_4 = \sigma_3 - \Delta_3 \sigma'_3 = (1+\alpha^2 x+\alpha^5 x^2)+\alpha^6(x+\alpha^2 x^2+\alpha^5 x^3) = 1+x+\alpha^6 x^2+\alpha^6 x^2$

$\omega_4 = \omega_3 - \Delta_3 \omega'_3 = \alpha^3+\alpha^4 x^2$

Comme j est égal à 3 (=2t-1), l'algorithme s'arrête et on peut écrire les deux égalités suivantes :

$$\sigma_{err+eff}(x) = \sigma_4(x) = 1+x+\alpha^6 x^2+\alpha^6 x^2$$

Par conséquent,

$$\beta(x) = [\sigma_{err+eff}(x)/ \sigma_{eff}(x)] = (1-\alpha^6 x)$$

La seule racine de $\beta(x)$ est α^{-6}, c'est à dire que l'on a une erreur en $6^{ème}$ position.

Finalement, deux effacements et une erreur ont été localisées.

Comme l'algorithme d'origine, le problème consiste à implanter des unités de calcul dans les corps de Galois. Cela est d'autant plus crucial que l'on doit maintenant évaluer 4 polynômes au lieu de 2.

III.2. Simulation et implantation du code Reed Solomon (127,k,d) avec effacements

Cette partie, consacrée à la simulation logicielle et à l'implantation sur FPGA du code Reed-Solomon (127,k,d), est divisée en trois paragraphes. En premier lieu, nous présenterons le code RS (127,k,d) et la technique de marquage utilisée pour localiser les effacements [11]. Le deuxième paragraphe sera consacré à la simulation de ce code et

des résultats que l'on peut en tirer (simulations d'environnement réel ou de phénomènes purement aléatoire). Enfin, afin de valider la conception en ASIC d'un circuit de codage/décodage, nous avons effectué une étude de faisabilité de l'implantation sur un seul circuit de l'algorithme de codage (ou de décodage) et de la technique de marquage. Pour cette étude, nous avons alors utilisé un cas particulier : le code RS(127,121,7) avec effacements.

III.2.1. Code de RS(127,k,d) avec effacements

D'après les propriétés des codes RS, la taille du mot code doit vérifier la formule :

$$n = 2^p - 1$$

Dans notre cas, la taille du mot code vaut 127, c'est à dire que p est égal à 7. Ce code est donc défini dans le corps de Galois $GF(2^7)$. Le polynôme primitif irréductible utilisé pour générer ce corps est le polynôme : $P(x) = P(x) = x^7 + x^3 + 1$. La taille des symboles est alors de 7.

Les caractéristiques de cette famille de code sont :

- Taille du mot code (127) symboles
- Nombre maximum d'erreurs pouvant être corrigées (t)
- Nombre maximum d'effacements pouvant être corrigés (0)
- Taille de la donnée à coder (127 –2t) symboles

Pour pouvoir utiliser les effacements, il est nécessaire de marquer les symboles afin de pouvoir détecter les éventuelles erreurs pouvant les affecter. Le corps de Galois étant fixé, la technique de marquage doit donc coder chaque symbole sans nuire à la validité du codage RS. Une des techniques les plus simples pour cela est d'ajouter de la redondance aux symboles suivant une loi connue de l'émetteur et du récepteur. En

même temps, cette redondance ne doit pas être excessive afin de ne pas trop diminuer le débit utile.

Pour pouvoir répondre à ces contraintes, la technique de marquage utilisée est le bit de parité impair qui détecte un nombre impair d'erreurs dans un symbole. Les erreurs en nombre pair qui ne sont pas détectées par le bit de parité seront alors considérées comme des erreurs par l'algorithme de décodage. En contrepartie, la taille du symbole passe de 7 à 8 bits ce qui est un avantage pour les implantations logicielles ou sur DSP.

Les caractéristiques du code de Reed-Solomon ainsi formé sont :

- taille du mot code 127 symboles,
- taille du message (127-2t) symboles,
- nombre d'erreurs corrigibles t,
- nombre maximal d'effacements $2t$
- correction conjointe erreurs t' et effacements e' avec $2*t' + e' \leq 2*t$,
- symbole composé d'un bit de parité et de 7 bits du code Reed-Solomon (127, k,d) d'origine. En résumé, cette technique de marquage revient à la concaténation du code de parité (8,7,2) à chacun des symboles du code RS(127,k,d).

III.2.2.Simulation

III.2.2.1. Principe de simulation

La technique de marquage expliquée précédemment et l'algorithme de décodage du Reed-Solomon (127,k,d) avec effacements ont été étudiés à TDF-C2R [11] en utilisant des fichiers d'erreurs de format binaire pour représenter le canal de transmission. Pour ces simulations, deux types de fichiers ont été choisis :

- des modèles aléatoires obtenus à partir d'un logiciel distribuant aléatoirement les erreurs selon la taille en octets du fichier et le taux d'erreur bit donné (TEB),

- des fichiers de mesures permettant l'approche d'un environnement réel (diffusion hertzienne, rétrodiffusion). Ces mesures ont été réalisées sur le terrain avec tout l'environnement analogique (modulateur, émetteur, récepteur).

Le principe de la simulation est représenté sur la figure (III-3)

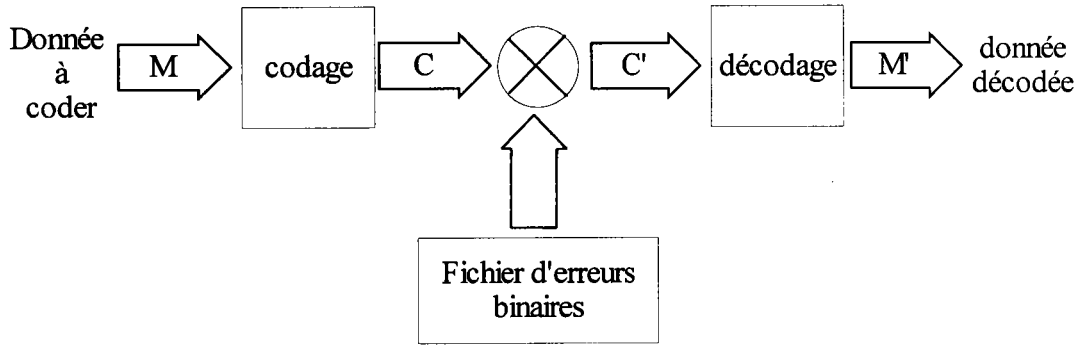


Fig III-3. Principe de Simulation

Le mot d'information M est envoyé dans un circuit de codage qui calcule le code de Reed-Solomon original ainsi que le bit de parité de chaque symbole. Au mot de code C obtenu, on ajoute les erreurs provenant du fichier choisi (environnement réel ou aléatoire). Le mot reçu C' est envoyé dans le circuit de décodage. Ce circuit décèle les effacements e' et les comptabilise. La détection des erreurs est activée si le nombre d'effacement e' dépasse 2t. Dans le cas contraire, la correction conjointe des erreurs et des effacements est activée. Le message de sortie M' est alors comparé au message original M.

III.2.2.2. Résultat

La figure III-4 donne un aperçu des simulations réalisées sur des modèles aléatoire et hertzien :

Modèle	TEB fichier d'erreurs	Taux d'erreurs résiduelles	Code Reed-Solomon (127,k,d)	Rendement
Aléatoire Erreurs isolées	$1,12.10^{-2}$	$1,82.10^{-3}$	RS (127,105,23) Sans effacements	82,68 %
		0	RS (127,105,23) Avec effacements	72,34 %
		0	RS (127,89,39) Sans effacements	70,08 %
Hertzien bande II Paquets d'erreurs de taille variée	$4,53.10^{-3}$	0	RS (127,65,63) Avec effacements	44,78 %
		0	RS (127,57,71) Sans effacements	44.88 %

Fig III-4. Résultats des simulations.

• Pour le modèle aléatoire, le fichier d'erreurs de format binaire contient uniquement des erreurs isolées avec un taux d'erreurs bits de $1,12.10^{-2}$. On peut remarquer que le code Reed-Solomon (127,105,23) sans effacements est en dépassement et ne permet pas de corriger toutes les erreurs et en a même rajouté (taux d'erreurs résiduelles égal à de $1,82.10^{-3}$). Le code RS (127,105,23) avec effacements permet de résoudre ce problème (i.e. on a un taux d'erreur résiduel nul). Mais cela se fait au détriment du rendement (il passe de 82,68 % à 72,34 %). Cependant, utiliser ce code est nettement plus avantageux que le code RS (127,89,39) sans effacement qui lui aussi permet un taux résiduel nul (le rendement passe de 72,34% à 70,08%).

• Pour le modèle simulant un support hertzien en bande II (ce modèle a été obtenu lors d'une campagne de mesures organisée par TDF-C2R.), on a une prédominance des erreurs adjacentes et une multitude de paquets d'erreurs de taille variée. La taille du plus grand paquet d'erreurs est de 217 bits. Les erreurs adjacentes ne représentent que 8,43%, la taille du fichier d'erreurs est de 267.246 octets.

Deux solutions sont possibles pour un taux d'erreurs résiduelles nul :

- code Reed-Solomon (127,57,71) sans effacements,
- code Reed-Solomon (127,65,63) avec effacements.

Les avantages du code de Reed-Solomon (127,65,63) avec effacement sur le code Reed-Solomon (127,57,71) sans effacements pour cette application sont :

1- le code Reed-Solomon (127,65,63) avec effacements nécessite une capacité de correction t = 31 pour un taux d'erreurs résiduelles nul. Par contre le code Reed-Solomon (127,57,71) nécessite une capacité de correction de 35 pour obtenir le même résultat. (Le nombre de symboles codés diminue de 65 à 57),

2- le code RS (127,65,63) n'induit pas de fausses corrections en cas de dépassement de la capacité du code. De plus, au-delà de 2*t, les erreurs sont détectées. Par contre, de fausses corrections apparaissent en cas de dépassement du code RS (127,57,71) sans effacements.

Code Reed-Solomon (127,k,d)		
	DECODAGE	
Paramètres du code	Sans effacements	Avec effacements
Rendement (k/n) en %	k/127	7k/8*127
Capacité maximale de correction	t erreurs	2*t erreurs
Nombre d'erreurs corrigibles	compris entre 1 à t	compris entre 1 à 2*t
Nombre maximale d'effacements	0	2*t
Détection si effacements > 2*t	Non	Oui
Panachage erreurs et effacements $2*t'+e' \leq d-1$	Non	Oui
Distance Hamming d	2*t + 1	2*t + 1

Fig III-5. Comparaison des performances.

Le tableau de la figure III-5 regroupe les performances du décodage du code Reed-Solomon (127,k,d) selon l'option "avec ou sans effacements". Malgré un rendement plus faible pour le code RS avec effacements, il y a de nombreux avantages à cette technique :

- détection des erreurs pour un nombre d'effacements $e' > 2*t$
- correction conjointe des erreurs t' et des effacements e
- absence de fausses corrections en cas de dépassement de la capacité de correction du code détecteur/correcteur d'erreurs.

Enfin, le code de Reed-Solomon (127,k,d) avec effacements a été comparé à d'autres codes présentant la même capacité de correction. Le rendement nous donne des indications sur la ressource nécessaire à leur mise en œuvre. Les profils d'erreurs corrigibles sont explicités ainsi que les possibilités d'effacements et de détection d'erreurs.

Code détecteur correcteur d'erreurs	Rendement	Profils d'erreurs corrigibles
Code de Golay (23,12,7)	52,17 %	Correction de trois erreurs aléatoires (t=3).
Code de Golay (24,12,8)	50,00 %	Correction de trois erreurs aléatoires (t=3) avec possibilités de détection d'erreurs (au-delà de 3).
Code à résidu quadratique (31,16,7)	51,61 %	Correction de trois erreurs aléatoires (t=3).
Code BCH (127,106,7)	83,46 %	Correction de trois erreurs aléatoires (t=3), nécessite l'usage des corps de Galois.
Code Reed-Solomon (127,121,7)	95,28 %	Correction de trois erreurs aléatoires (t=3), pas d'effacements ni détection des erreurs en cas de dépassement de la capacité de correction.
Code Reed-Solomon (127,121,7) concaténé à un code de parité (8,7,2)	83,37 %	Correction de trois erreurs aléatoires (t=3), 6 effacements et détection des erreurs en cas de dépassement de la capacité de correction, absence de fausses corrections.

Fig III-6. Comparaison avec d'autres codes.

III.2.3. Implantation

La simulation logicielle ayant permis de valider la technique de marquage utilisée pour les codes RS (127,k,d) avec effacement, une implantation de cette technique a été étudiée afin de valider la faisabilité en ASIC d'un tel circuit (RS + technique de marquage).

Des circuits de codage/décodage de Reed-Solomon avec effacements ont déjà été étudiés et sont disponibles dans l'industrie [37], [38]. Mais ces circuits n'implantent pas une technique de marquage.

De plus, cette étude étant une étude de faisabilité, le code VHDL produit n'est pas optimisé pour les performances. Il existe des architectures parallèles pour les décodeurs RS permettant d'obtenir un débit important [39] mais qui ne traitent pas les effacements (à cause de l'algorithme utilisé : l'algorithme d'Euclide).

Avant de développer l'architecture choisie pour l'implantation, nous allons tout d'abord nous intéresser au choix du multiplieur dans les corps de Galois. En effet, l'algorithme de Berlekamp nécessite le calcul du produit de plusieurs polynômes. Or si la somme de deux éléments dans un corps de Galois est simple (addition bit à bit modulo 2), il n'en est pas de même pour la multiplication.

III.2.3.1. Choix du multiplieur.

Le multiplieur dans un corps de Galois est un des composants principaux utilisés dans l'algorithme de Berlekamp-Massey. Il nécessite donc une attention particulière. Comme toutes les opérations se font sur des symboles de 7 bits, il paraît judicieux d'utiliser un multiplieur parallèle qui permet d'obtenir le résultat de la multiplication de deux éléments d'un corps de Galois en un seul cycle d'horloge.

Le problème est alors de calculer la valeur C codée sur 7 bits connaissant A et B et telle que

$$C(x) = A(x)B(x)\ [P(x)].$$

Matriciellement, cette relation peut s'écrire comme [37] :

$$C = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{p-1} \end{pmatrix} = \begin{pmatrix} f_{0,0} & \cdots & f_{0,p-1} \\ \vdots & \ddots & \vdots \\ f_{p-1,0} & \cdots & f_{p-1,p-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{p-1} \end{pmatrix} \text{ avec}$$

$$f_{i,j} = \begin{cases} a_i & j=0\,;i=0,\ldots,p-1 \\ u(i-j)a_{i-j} + \sum_0^{j-1} q_{j-1-t,i}\, a_{p-1-t} & j=1,\ldots,p-1 \;\; i=0,\ldots,p-1 \end{cases}$$

La fonction u(t) est définie pour ne prendre que deux valeurs 0 (si $t<0$) ou 1 (si $t \geq 0$).

Les données $q_{i,j}$ sont données par la relation matricielle :

$$\begin{pmatrix} x^{p} \\ x^{p+1} \\ \vdots \\ x^{2p-2} \end{pmatrix} = \begin{pmatrix} q_{0,0} & \cdots & q_{0,p-1} \\ \vdots & \ddots & \vdots \\ q_{p-2,0} & \cdots & q_{p-2,p-1} \end{pmatrix} \begin{pmatrix} x \\ x^{1} \\ \vdots \\ x^{p-1} \end{pmatrix} [P(x)]$$

Dans notre cas p vaut 7 et P(x) est égal à $x^7 + x^3 + 1$. La matrice Q est alors la suivante

$$Q = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Par contre, le codeur utilisera directement 6 multiplieurs par une constante (vecteur A fixe) optimisés en utilisant l'algorithme décrit dans [37]. Cette solution a été retenue car lors de la synthèse, il y a optimisation des équations logiques (A est fixe et B variable). Par contre, dans le cas général, il n'y a pas optimisation de l'emploi des portes logiques (A et B sont variables). Il en découle que les temps de propagation dans le multiplieur général sont plus élevés que dans le multiplieur par une constante.

III.2.3.2. Implantation du codeur

Deux fonctions doivent être réalisées par le codeur :

- Effectuer le codage initial du code Reed-Solomon (127,121,6)
- Ajouter le bit de parité à chaque symbole du mot code

D'après la définition du code de Reed-Solomon, le mot de code doit être un multiple du polynôme générateur g(x) donné par la formule (3.1). Pour le cas étudié, ce polynôme est égal à :

$$g(X)=X^6+\alpha^{110}X^5+\alpha^4X^4+\alpha^{123}X^3+\alpha^9X^2+\alpha^{120}X+\alpha^{15}$$

Le code de Reed-Solomon étant un code linéaire cyclique, une technique généralement utilisée pour obtenir le mot de code est de concaténer la donnée M(x) avec

le reste de la division de $x^7M(x)$ par g(x). Le circuit de calcul du mot code est alors similaire à celui décrit dans le chapitre I (fig I-15b). Il existe cependant une différence. Les codes considérés dans le chapitre I travaillent avec un symbole de taille 1 (1 bit). La multiplication par (g_i) revient alors à implanter une liaison (g_i =1) ou non (g_i =0) entre la boucle de retour et la porte XOR considéré (fig III-7 a et b).

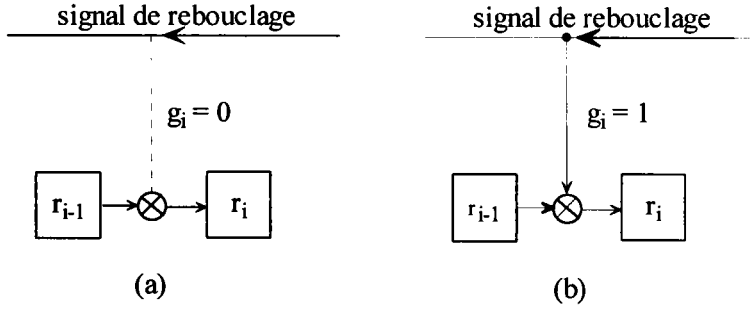


Fig III-7. Principe de Multiplication

Dans le cas du Reed-Solomon (127,k,d), la multiplication par g_i se fait en utilisant un multiplicateur dans un champ de Galois (le rétrobouclage se fait alors via un bus de 7 bits). Le schéma de principe du diviseur polynomial pour le code RS(127,121,6) est représenté à la figure III-8

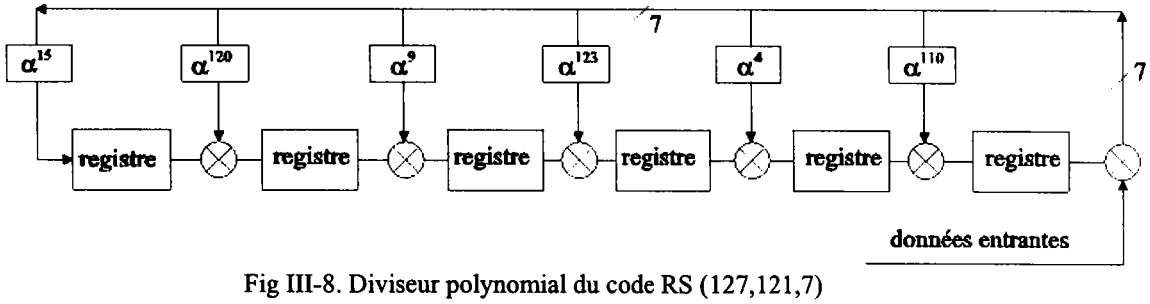


Fig III-8. Diviseur polynomial du code RS (127,121,7)

La deuxième opération réalisée par le codeur est le calcul et le rajout du bit de parité. Elle est effectuée très simplement via un réseau de porte XOR.

Finalement, le schéma de principe du codeur (division + bit de parité) est représenté à la figure III-9

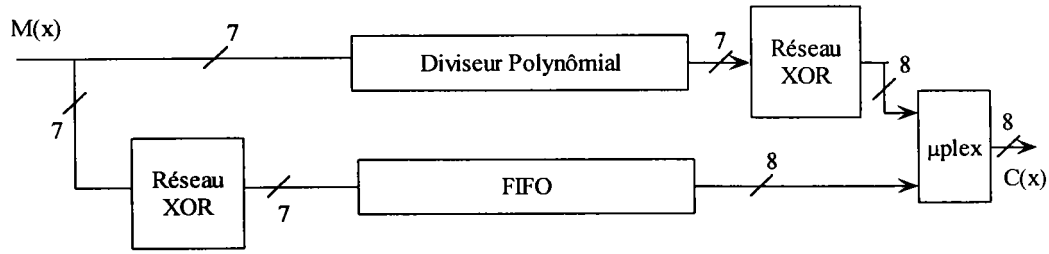


Fig III-9. Architecture du codeur

Comme pour les codes à logique majoritaire, le codage série (par rapport au symbole) ne pose pas de problème d'implantation dès que le choix crucial du multiplieur a été effectué.

Par contre, le décodage est nettement plus compliqué à implanter à cause des opérations polynomiales dans les corps de Galois (multiplication de polynôme, dérivée, Algorithme de Berlekamp-Massey).

III.2.3.3. Implantation du décodeur

III.2.3.3.1. Principe de correction

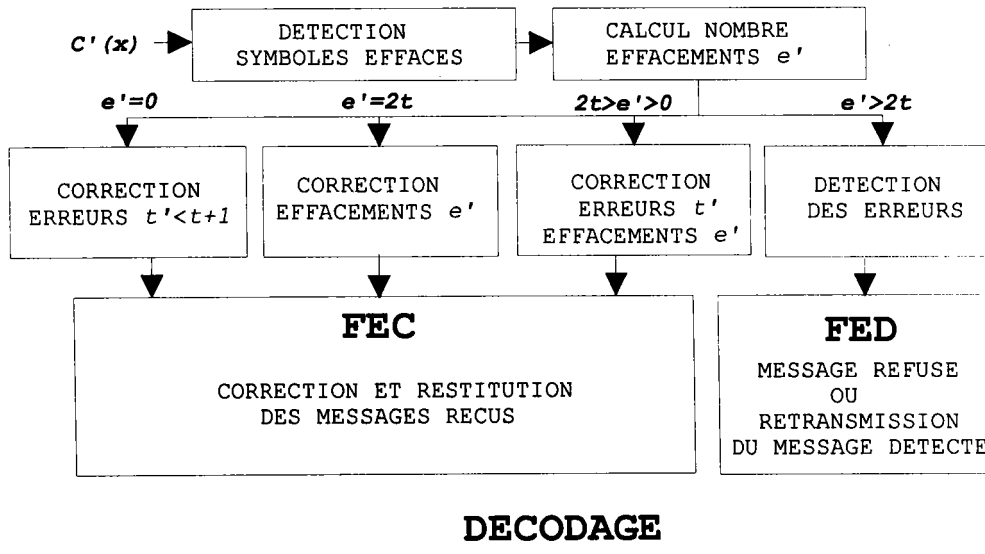


Fig III-10. Principe de décodage

Pour le décodage, plusieurs cas peuvent se produire (fig III-10) ;

Si le nombre d'effacement est supérieur à 2*t, alors le système peut immédiatement décider que le mot n'est pas corrigible. L'algorithme de Berlekamp-Massey n'est alors pas exécuté.

Dans le cas contraire, cet algorithme doit être activé. Ce n'est qu'à la fin de celui-ci que le système pourra décider si le mot n'est pas corrigible (e'+2*t' > 2*t) ou si un des trois autres cas de la figure 3 s'applique. Cette adaptation automatique de l'algorithme de décodage selon le cas (erreurs, erreurs et effacements, effacements, détection des erreurs) constitue la solution optimale pour une implantation logicielle.

Dans le cadre de cette étude, nous nous sommes placés dans le cas le plus général, c'est à dire que nous supposons qu'il y a toujours des erreurs et des effacements.

III.2.3.3.2. Organisation générale

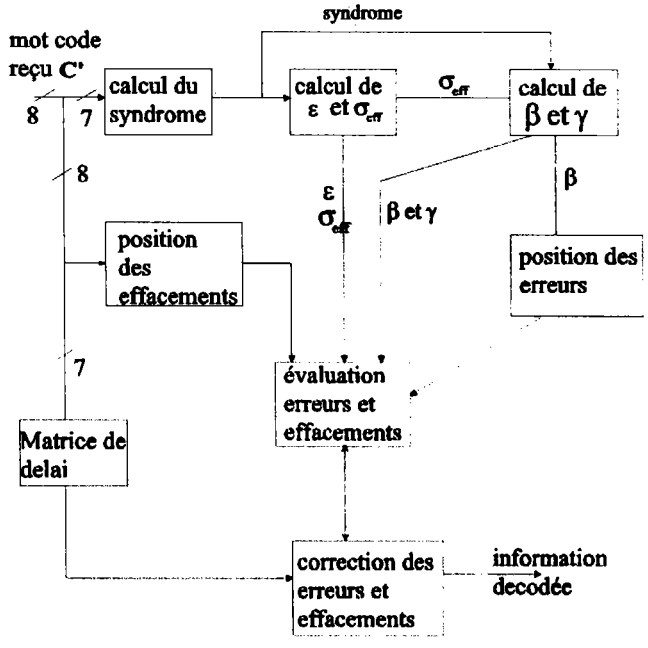


Fig III-11. Architecture utilisée pour le décodeur du Reed-Solomon *(127,k,d)*.

L'architecture classique [39] utilisée pour le décodeur est celle présentée à la figure III-11. Elle fonctionne de la façon suivante :

Les symboles du mot code (trame de 127 symboles) sont envoyés dans un réseau de portes XOR pour détecter les effacements. Quand un effacement est localisé, sa position et la valeur (α^i) correspondante sont mémorisées dans des registres. Parallèlement à la localisation des effacements, la trame de donnée est utilisée pour le calcul du syndrome.

Connaissant la position des effacements et le syndrome, on peut débuter le calcul des polynômes ε et $\sigma_{eff.}$ γ et β (algorithme de Berlekamp). La donnée $\beta(x)$ nous permet, grâce à un test sur $\beta(\alpha^{-i})$, de localiser les erreurs (bloc localisateur).

A ce stade, les erreurs et les effacements sont localisés. Tout les polynômes sont connus. On peut donc évaluer les errata et corriger les données préalablement stockées dans une FIFO.

Dans les paragraphes suivants, nous étudierons successivement les étages de calcul :

- du syndrome
- des polynômes ε, σ_{eff}, γ et β.

III.2.3.3.3. Calcul du syndrome

Le syndrome S(x) est égal à :

$$S(x) = \sum_0^5 s_i x^i \text{ avec } s_i = C'(\alpha^i)$$

On a alors

$$C'(\alpha^i) = c'_{n-1}(\alpha^i)^{n-1} + c'_{n-2}(\alpha^i)^{n-2} + \cdots + c'_1(\alpha^i)^1 + c'_0(\alpha^i)^0$$

En mettant en facteur (α^i), on a

$$C'(\alpha^i) = c'_0 + (\alpha^i)\left[c'_{n-1}(\alpha^i)^{n-2} + c'_{n-2}(\alpha^i)^{n-3} + \cdots + c'_1\right] = c'_0 + (\alpha^i)C'_1(x)$$

Comme précédemment, on peut mettre (α^i) en facteur dans $C'_1(x)$. On peut alors utiliser un système récursif pour le calcul de C'(α^i) :

$$C'(\alpha^i) = c'_0 + (\alpha^i)\left[c'_1 + (\alpha^i)\left[c'_2 + (\alpha^i)\left[c'_3 \cdots\cdots \left[c'_{n-2} + (\alpha^i)c'_{n-1}\right]\right]\right]\right]$$

L'équation précédente montre que C'(αi) peut être obtenu après le calcul successif des valeurs suivantes :

$$\begin{cases} A_1 = c'_{n-2} + (\alpha^i)c'_{n-1} \\ A_2 = c'_{n-3} + (\alpha^i)A_1 \\ \vdots \\ A_n = c'_0 + (\alpha^i)A_{n-1} \end{cases}$$

Le calcul de C'(α^i) peut alors se faire via un processus itératif en utilisant un système (fig III-12) qui va calculer successivement les valeurs A_j.

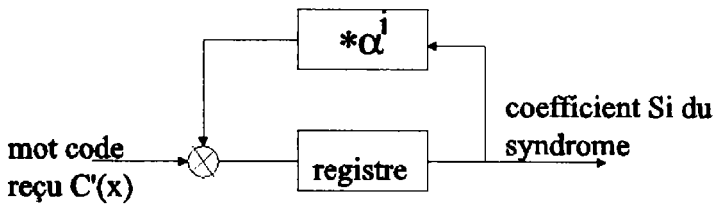


Fig III-12. Circuit de calcul des coefficients α^i

Les différents coefficients S_i du syndrome sont alors calculés en utilisant (d) systèmes de ce type en parallèle. Les coefficients du syndrome sont alors obtenus après n cycle d'horloges.

III.2.3.3.4. Implantation de l'algorithme de Berlekamp

Les coefficients du syndrome et les positions des effacements étant connus, on peut calculer les coefficients des polynômes $\sigma_{eff}(x)$ et $\varepsilon(x)$. Ces coefficients sont générés par deux systèmes d'équations récursifs (première partie de l'algorithme de Berlekamp modifié) :

$$\sigma_{j+1} = \sigma_j - \alpha_i\sigma'_j \qquad \omega_{j+1} = \omega_j - \alpha_i\omega'_j + \Delta_j x^j \qquad (3.24)$$

$$\sigma'_{j+1} = x\sigma_{j+1} \qquad \omega'_{j+1} = x\omega_{j+1} \qquad (3.25)$$

avec j qui désigne la position d'un effacement

Δ_j le coefficient en x^j du produit $\sigma_j S$.

On peut remarquer que ces deux systèmes sont très similaires :

$$\sigma_{j+1} = \sigma_j - \alpha_i \sigma'_j + (\Delta_j = 0) x^j \quad \omega_{j+1} = \omega_j - \alpha_i \omega'_j + \Delta_j x^j \qquad (3.26)$$

$$\sigma'_{j+1} = x\sigma_{j+1} \qquad \omega'_{j+1} = x\omega_{j+1} \qquad (3.27)$$

Le calcul des coefficients de $\sigma_{eff}(x)$ et $\varepsilon(x)$ peut donc se faire en employant deux systèmes en parallèle et initialisés de façon différente. Le système retenu est celui décrit par les équations suivantes (fig III-13) :

$$u_{j+1} = u_j - a u'_j + b X^j \qquad (3.28)$$

$$u'_{j+1} = X u_{j+1} \qquad (3.29)$$

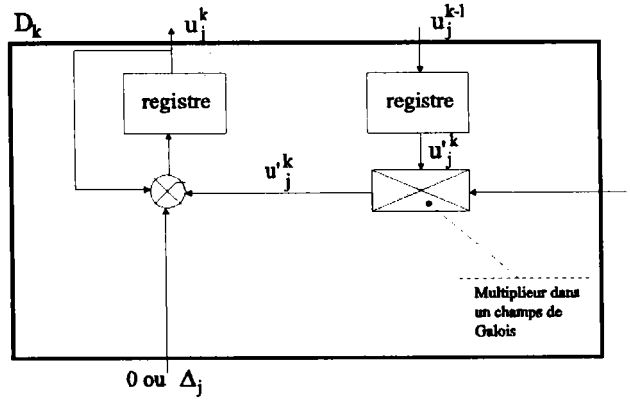


Fig III-13. Schéma du système implantant l' équation (3.28)

On remarque immédiatement que pour le calcul de σ_{j+1} on a (a,b) qui est égal à $(\alpha_i, 0)$. Par contre, le calcul de ω nécessite d'avoir (a,b) égal à (α_i, Δ_j).

Le calcul des polynômes $\sigma_{eff}(x)$ et $\varepsilon(x)$ va donc utiliser deux circuits C1 et C2 implantant les équations (3.28) et (3.29) avec des entrées différentes. L'architecture de ces deux circuits sont identiques mais diffèrent au niveau des signaux qui leurs sont appliqués (pour les circuits C1 et C2, on a respectivement a $=\alpha_i$ et b=0 ou Δ_i) (fig III-14).

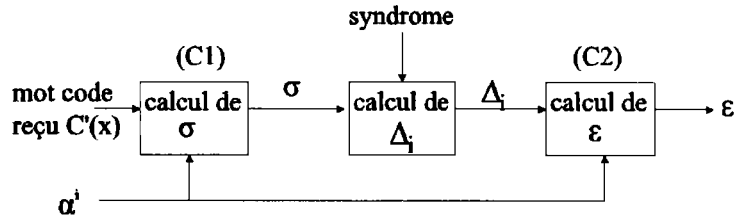


Fig III-14. Schéma du circuit de calcul des polynômes ε et σ_{eff}

De plus, u_j et u'_j représentent chacun les coefficients d'un polynôme de degré égal au maximum à 6. Les équations peuvent donc s'écrire respectivement comme :

$u'^{0}_{j+1} = 0$ et $u'^{m}_{j+1} = u^{m-1}_{j+1}$ pour $m \in \{1,2,..,5\}$ (3.30)

Ce qui revient à effectuer un décalage à droite entre u'_{j+1} et u_{j+1} (fig III.15).

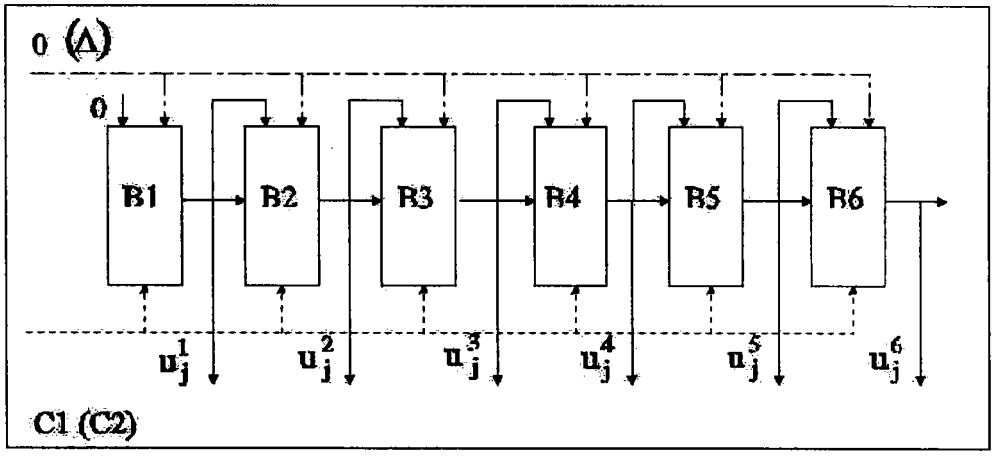


Fig III-15. Principe du décalage à droite

Au bout de s cycles de calcul (s étant le nombre d'effacements), on a les égalités :

$\sigma = \sigma_{eff}$

$\omega = \varepsilon$

A ce stade, deux polynômes sont inconnus (β et γ). Pour les calculer, on utilise l'algorithme de Berlekamp-Massey en implantant les équations suivantes :

$\sigma_{j+1} = \sigma_j - \Delta_j \sigma'_j$ $\omega_{j+1} = \omega_j - \Delta_j \omega'_j$

$\sigma'_{j+1} = X(C.\sigma'_j + C. \Delta^{-1}_j \sigma'_j)$ $\omega'_{j+1} = X(C. \omega'_j + C. \Delta^{-1}_j \omega'_j)$

$d_j = d_j.C+(j+1+s-d_j).C$

où C est le résultat du test (Δ_j=0 ou $2d_j > j+s$)

Pour calculer ces deux polynômes, on utilise la méthode précédente. Un circuit Di (fig III-16) implante les équations précédentes.

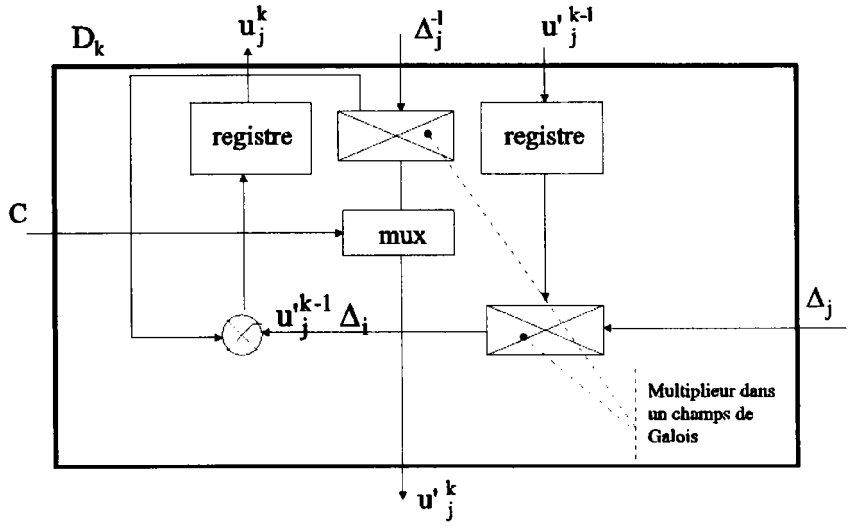


Fig III-16. Schéma de principe de D_k

La réalisation de la multiplication par X se fait par décalage à droite comme précédemment (fig III-17). Au bout de (t-s) cycles de calcul, le polynôme σ est égal à $\sigma_{err+eff}$.

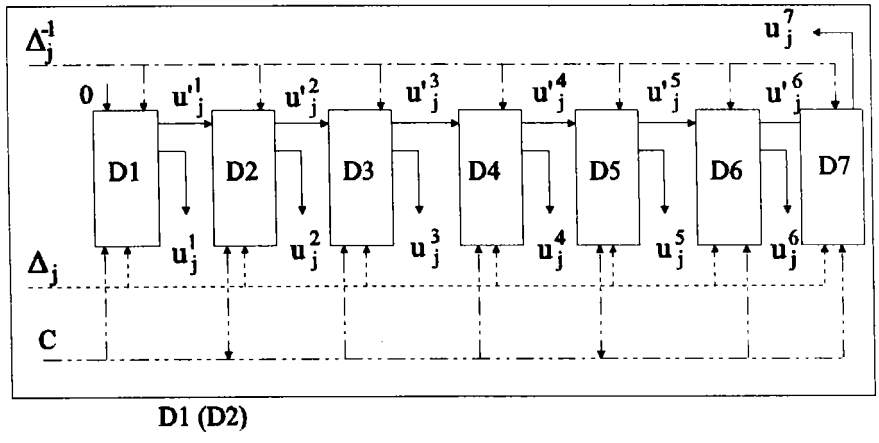


Fig III-17. Circuit utilisé pour le calcul des polynômes β et γ.

III.2.3.3.5. Localisation et évaluation des erreurs

Le polynôme β étant connu, un calcul par une architecture pipeline de $\beta(\alpha^{-i})$ (fig III-18) avec i variant 0 à 126 (taille de la donnée) permet grâce à un test sur zéro de connaître les racines du polynôme β (position des erreurs).

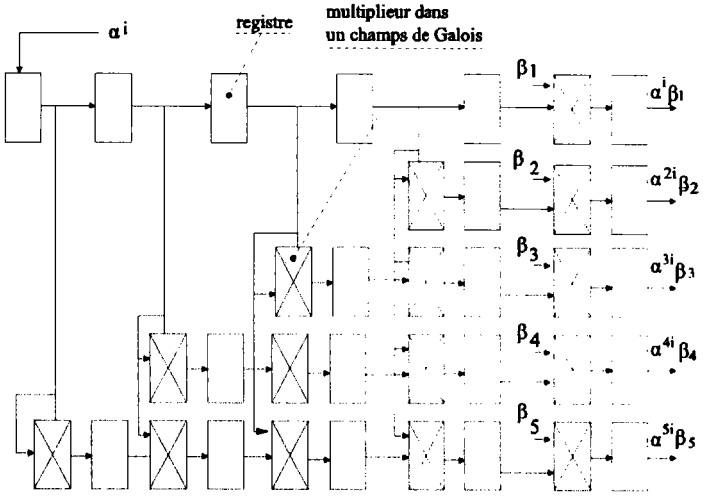


Fig III-18. Circuit d'évaluation de $\beta(\alpha^{-i})$

Quand les positions des effacements et des erreurs sont connues, on peut calculer les valeurs correctives.

D'après les formules (3.24) et (3.25), il est nécessaire de calculer les dérivées des polynômes β et γ. Cependant, comme nous travaillons dans des corps de Galois, la relation suivante est toujours vérifiée :

$$\alpha^i + \alpha^i = \alpha^i - \alpha^i = 0$$

Par conséquent, on a toujours :

$$M'(x) = \left(\sum_{i=0}^{\deg(m)} m^i x^i\right)^{(1)} = \left(\sum_{i=1}^{\deg(m)-1} i \times m^{i-1} x^i\right) = \left(\sum_{\substack{i=1 \\ i \text{ impair}}}^{\deg(m)-1} i \times m^{i-1} x^i\right)$$

Le dernier terme de l'équation précédente est obtenu en utilisant la propriété suivante des corps de Galois :

$$\frac{d}{dx}\left(\alpha^{j}x^{2i}\right)=(2i)\alpha^{j}x^{2i-1}=0$$

Le calcul de la dérivée des polynômes consiste donc à ne conserver que les coefficients des termes impairs, remplacer les termes pairs par zéro et décaler d'un rang ces coefficients. Le calcul des valeurs $\beta(\alpha^i)$, $\gamma(\alpha^i)$, $\sigma_{eff}(\alpha^i)$, $\varepsilon(\alpha^i)$, $\beta'(\alpha^i)$, $\gamma'(\alpha^i)$, $\sigma'_{eff}(\alpha^i)$, $\varepsilon'(\alpha^i)$ peut alors s'effectuer en utilisant un circuit du type précédent.

III.2.3.3.5. Résultats

L'architecture du codeur/décodeur qui a été présentée dans les paragraphes précédents a été décrite au niveau RTL en VHDL et synthétisée sur FPGA (type FLEX10K30) en utilisant le logiciel MaxplusII de la société Altera. Cela nous a permis d'avoir les résultats suivants :

- La fréquence maximum de fonctionnement est de 20 Mhz (la fréquence maximum des circuits Flex10K est de 125 Mhz), ce qui nous autorise un débit de 140 Mbits/s,
- La surface occupée est de l'ordre de 6547 Logic Cells (LC's). Cette surface est comparable aux circuits industriels (la surface occupée par le décodeur réalisé par la société Hammer Cores est de 5117 LC's).

De part l'architecture choisie, cette implantation est limitée par le nombre de cycles N nécessaires pour avoir une donnée décodée:

N = N1+N2+N3+N4 où

N1 : nombre de cycles d'horloge nécessaires au calcul du syndrome (127 dans notre cas),

N2 : nombre de cycles nécessaires au calcul des différents polynômes (6 cycles de calcul maximum) soit 60 cycles d'horloge dans le cas le plus défavorable (nombre maximum d'erreurs ou d'effacements),

N3 : nombre de cycles d'horloge nécessaires pour déterminer la position des erreurs (127 cycles d'horloge),

N4 : nombre de cycles d'horloge nécessaires à la correction des erreurs et effacements (254 cycles d'horloge).

Dans le cas du Reed-Solomon (127,121,7), les opérations nécessitent un temps de latence de 568 cycles d'horloge (cas le plus défavorable). Ce temps de latence est comparable à celui du circuit développé par la société Hammer Cores [41]. Ce nombre peut être amélioré par le développement d'une architecture dans laquelle chaque bloc sera pipeliné et/ou parallèlisé.

III.3. Conclusions

La validation par simulation basée sur des modèles d'environnement réel (fichiers d'erreurs de format binaire) confirme la possibilité de corriger à la fois les erreurs et les effacements quand le code n'est pas en dépassement et détecter les erreurs quand celui est en dépassement (panachage du "FEC" (Forward Error Correction) et du "FED" (Forward Error Detection)).

Les effacements relayent la méthode classique de correction d'erreurs en ce sens :

- qu'ils assurent la détection des erreurs au-delà de 2*t,
- qu'ils évitent les fausses corrections en cas de dépassement du code Reed-Solomon,
- qu'ils améliorent la capacité de correction des codes Reed-Solomon (correction conjointe des erreurs et effacements dans la limite de $e'+2*t \leq d-1$).

La conception d'un codeur/décodeur avec effacements a été décrite en VHDL et validée sur FPGA avec pour objectif de traiter des flots de données en continu. Les résultats obtenus sont comparables, en terme de complexité, à ceux des circuits existants. La mise en œuvre de la détection, en plus du traitement des effacements, apporte également une valeur ajoutée aux circuits traitant les effacements (possibilité rarement mise en œuvre).

Conclusion générale.

Actuellement, nous transmettons de plus en plus de données entre deux points A et B que ce soit dans l'industrie (contrôle à distance d'appareil en milieu bruité, donnée provenant de mesure) ou dans les loisirs (vidéo à la demande, télévision numérique, réseau internet). Un des problèmes qui se pose alors est d'assurer la fidélité du message transmis (i.e. le message reçu est bien le message envoyé). Plusieurs techniques ont été mise au point pour détecter et/ou corriger les erreurs (ARQ, ARQ-Hybride, Code détecteur/correcteur d'erreurs). Des trois techniques précitées, seul les codes correcteurs d'erreurs permettent (dans une certaine mesure) la détection et la correction des erreurs. Le débit des applications le permettant, la voie logicielle a souvent été utilisée pour implanter les codes correcteurs. Ensuite, grâce à l'avènement du circuit intégré, il a été possible d'implanter dans une seule puce la fonction de détection et de correction des erreurs avec un meilleur débit que la version logicielle (mais en sacrifiant la souplesse permise par le logiciel). Le volume et donc le débit des données augmentant, la version matérielle traditionnelle montre ses limites. Il est alors nécessaire de développer des circuits spécifiques à haut débit permettant de répondre à ces nouvelles contraintes.

De plus le débit augmentant, le risque d'erreurs s'accroît (durée des données plus faible pour une durée de l'erreur identique). Le nombre d'erreurs peut alors augmenter. La

taille de la redondance (symboles rajoutés à la donnée par le circuit de correction) étant liée au nombre d'erreurs pouvant être corrigées, cette augmentation des erreurs va avoir un impact négatif sur le débit utile des applications. Il faut donc trouver des algorithmes permettant d'augmenter la capacité de correction des codes correcteurs d'erreurs.
Ces deux problèmes (circuit à haut débit et problème de la capacité de correction) ont été traités dans ce mémoire. La famille des codes à logique majoritaire a été utilisée pour une étude portant sur l'amélioration de l'architecture du couple codeur/décodeur en vue d'un traitement à haut débit de flots continus de données (circuit à haut débit). Par contre l'étude de l'augmentation de la capacité de correction des codes s'est faite sur la famille des codes de Reed-Solomon en mettant en œuvre une technique de marquage.

Après un bref rappel sur les différentes techniques permettant de détecter et/ou corriger les erreurs de transmission entre deux points A et B, la théorie des codes correcteurs d'erreurs et plus particulièrement des codes linéaires cycliques a été développée (la propriété marquante de ce type de code est que le mot code est un multiple d'un polynôme générateur caractéristique du code utilisé). Nous nous sommes alors intéressés, d'un point de vue général, aux différentes fonctions nécessaires à :

- la création d'un mot de code répondant à la propriété citée précédemment,
- la détection, la localisation et la correction des erreurs.

Dans le deuxième chapitre, nous nous sommes focalisés sur les architectures du codeur et du décodeur pour une famille de code linéaire cyclique : les codes à logique majoritaire. Après une présentation mathématique de la famille de code étudiée, un état de l'art sur les différentes architectures possibles pour implanter le couple codeur/décodeur a été réalisé. La conclusion de cette étude a été que la seule

architecture permettant de répondre aux contraintes fixées (haut débit et traitement de flot continu de données) est l'architecture parallèle. L'architecture globale des codeurs et décodeurs a alors été définie.

Ces deux architectures (codeur et décodeur) ont été décrites au niveau RTL en VHDL pour un code particulier : le code DSCC (72,44). Après validation de ces architectures, une version générique de ces circuits a été réalisée permettant le choix du code (DSCC, DTI, EG) et du degré de parallélisme. Grâce à cette version générique des codeurs et décodeurs, une étude en surface et en débit a pu être effectuée. On a ainsi pu montrer que l'architecture parallèle permet une augmentation notable du débit par rapport à l'architecture série alors que conjointement l'augmentation de la surface est limitée. De plus, alors que le débit du circuit série est limité par le circuit de division polynomiale (cœur du circuit de codage/décodage), le circuit parallèle est limité quant à lui par les étages d'entrées et de sorties (conversion parallèle/série et série/parallèle). Il reste maintenant à valider ces résultats sur ASIC pour pouvoir affiner l'étude de la surface et du débit des circuits de codage et décodage et à étendre la démarche à d'autres codes.

Le troisième chapitre est consacré à la mise en œuvre d'une technique de marquage pour les codes de Reed-Solomon permettant d'augmenter la capacité de correction du code utilisé. Après une présentation mathématique de cette famille, le choix de la technique de marquage est expliqué. Ce marquage est alors validé via des simulations (simulation d'environnement réel, d'erreurs aléatoires) effectuée par la société TDF-C2R.

Les codeurs/décodeur de Reed-Solomon avec effacements présents dans le commerce n'implantent pas de technique de marquage. Aussi avons nous décidé de faire une étude

de faisabilité concernant l'implantation en ASIC sur un circuit d'un codeur Reed-Solomon (127,121,7) et de sa technique de marquage. Cette étude a permis de montrer que cette technique de marquage pouvait être implantée conjointement avec le décodeur Reed-Solomon sans trop pénaliser la surface totale du circuit. Les résultats obtenus devraient être améliorés par une future étude permettant d'améliorer et d'optimiser les différents blocs constitutifs des circuits de codage et de décodage.

Annexe A. Les corps de Galois.

I. Définition des corps de Galois

I.1. Définition

Soit p un nombre premier. On considère l'ensemble F constitué des p premiers entiers positif $F = \{0,1,2,\ldots,p-1\}$.

On munit cet ensemble de deux lois $\oplus$ et Θ définie par :

$$\forall (i,j) \in F^2,\ i \oplus j = s$$

$$\forall (i,j) \in F^2,\ i \Theta j = r$$

où (s) et (r) représentent respectivement les restes des divisions de (i+j) et de (i×j) par p.

On peut démontrer que

- $(F, \oplus)$ et (F^*, Θ) sont des groupes commutatifs
- $\forall (i,j,k) \in F^3,\ i\Theta(j\oplus k) = (i\Theta j)\oplus(i\Theta k)$

Par conséquent, $(F, \oplus, \Theta)$ est un corps. Dans le cas (p =2), l'ensemble F est constitué de deux éléments (0 et 1). Les opérations Θ et $\oplus$ sont respectivement équivalentes à un ET et un XOR logique.

Il est possible pour n'importe quel m d'étendre le corps GF(p) à un corps de p^m éléments. Ce corps appelé corps de Galois est noté GF(p^m).

I.2. Caractéristique du corps de Galois

On considère le corps de Galois GF(q). On forme alors la somme S suivante :

$$S_k = \sum_{i=0}^{k} 1$$

L'opération $\oplus$ étant interne à GF(q), il existe deux éléments m et n tels que :

$$S_m = S_n$$

Ce qui implique que $\sum_{i=0}^{m-n} 1 = 0$. Il existe donc un nombre λ tel que :

1- $\sum_{i=0}^{\lambda} 1 = 0$ et

2-il n'existe aucun nombre plus petit vérifiant la propriété précédente.

Ce nombre λ est appelé *caractéristique* de GF(q). Quand q est premier, la caractéristique de GF(q) est égale à q.

I.3. ordre de GF(q)

Soit a un élément non nul de GF(q). De façon analogue à ce qui précède, on peut former le produit P_k :

$$P_k = \prod_{1}^{k} a$$

Comme le corps possède un nombre fini d'élément et que l'opération de multiplication est interne au corps, tout les produits P_k ne sont pas distincts, c'est dire :

$$\exists (\alpha, \beta) \text{ tels que } P_\alpha = P_\beta$$

Comme précédemment, il existe une valeur *n* telle que :

$P_n = 1$

n est le plus petit entier tels que $P_n = 1$

Cet entier est appelé *ordre de l'élément a*

I.4. Autres propriété de GF(q)

Soit $(q_1,q_2,\ldots q_{q-1})$ les (q-1) éléments non nuls distincts de GF(q). Soit a un élément non nul de cet ensemble. Les (q-1) éléments $(a.q_1,a.q_2,\ldots,aq_{q-1})$ sont non nuls et distincts. On a alors :

$$\prod_{i=1}^{q-1}(aq_i)=(a^{q-1})\prod_{i=1}^{q-1}q^i=\prod_{i=1}^{q-1}q^i$$

Comme les valeurs a et $\prod_{i=1}^{q-1}q^i$ sont non nuls, on doit avoir $a^{q-1}=1$

D'où le théorème :

Soit a un élément non nul de GF(q) alors $a^{q-1}=1$

De plus si n est l'ordre d'un élément non nul de GF(q) alors on peut prouver que n divise (q-1)

II. Le corps de Galois GF(2^m)

Avant de détailler la construction et les propriétés de GF(2^m), nous allons donner quelques définitions utiles sur les polynômes dont les coefficients appartiennent à GF(2).

II.1. Définitions

II.1.1. Polynôme irréductible

Un polynôme P(x) de degré m à coefficients dans GF(2) est irréductible si p(x) n'est divisible par aucun polynôme (à coefficients dans GF(2)) de degré inférieur à m (mais positif).

Par les trois polynômes de degré 2 (X^2,X^2+1,X^2+X) ne sont pas irréductibles car ils peuvent être divisés par X ou X+1. Par contre X^2+X+1 est irréductible.

II.1.2. Polynôme primitif

Un polynôme irréductible de degré m est dit primitif si le plus petit entier n tel que P(x) divise (x^n+1) est n = 2^m-1.

Par exemple X^4+X+1 divise X^{15}+1 mais ne divise aucun polynôme de degré inférieur à 15. Par conséquent, le polynôme X^4+X+1 est primitif.

II.2. Le corps GF(2^m)

II.2.1. Construction du corps GF(2^m)

Soit P un polynôme primitif sur GF(2) de degré m. Soit α tels que p(α) soit nul. On définit l'opération . par :

$0.0 = 0$
$0.1 = 1.0 = 0$
$1.1 = 1$
$0.\alpha = \alpha.0 = 0$
$1.\alpha = \alpha.1 = \alpha$
$\alpha^j = \alpha.\alpha.\cdots.\alpha$ (j fois)

Comme P(x) divise (X^{2^m-1}+1), on peut écrire :

$$\alpha^{2^m-1}+1=q(\alpha)p(\alpha)=0 \Rightarrow \alpha^{2^m-1}=1$$

Par conséquent, l'ensemble F pour lequel la multiplication . est définie est l'ensemble :

$$F=\left\{\alpha^0,\alpha^1,\cdots,\alpha^{2^m-2}\right\}$$

De plus, on a :

$$\alpha^i = q_i(\alpha)p(\alpha)+a_i(\alpha) = \sum_{j=0}^{m-1} a_{ij}x^j$$

Comme P est un polynôme primitif, il ne divise aucun polynôme de degré inférieur à m. Par conséquent, les α^i peuvent être représentés par des polynômes non nuls distincts de degré m-1.

La représentation polynomiale permet de définir facilement une opération d'addition par :

$$\alpha^i + \alpha^k = \sum_{j=0}^{m-1} a_{ij}x^j + \sum_{j=0}^{m-1} a_{kj}x^j = \sum_{j=0}^{m-1} (a_{ij} + a_{kj})x^j$$

L'ensemble F (défini précédemment) muni de cette loi est un groupe commutatif. De plus, il est aisé de démontrer que la multiplication est distributive par rapport à l'addition. Donc l'ensemble F est un corps à 2^m éléments, GF (2^m), qui peuvent être représenté de deux manières différentes :

Une représentation sous forme d'une puissance de α très utile pour la multiplication,

Une représentation sous forme polynomiale intéressante pour l'addition.

Representation puissance	polynomiale			
	x^0	x^1	x^2	x^3
0	0			
1	1			
α^1		α		
α^2			α^2	
α^3				α^3
α^4	1	α		
α^5		α	α^2	
α^6			α^2	α^3
α^7	1	α		
α^8	1		α^2	
α^9		α		α^3
α^{10}	1	α	α^2	
α^{11}		α	α^2	α^3
α^{12}	1	α	α^2	α^3
α^{13}	1		α^2	α^3
α^{14}	1			α^3

Représentation des éléments de GF(2^4) générés par $1+X+X^4$

II.3. Propriétés de GF(2^m)

II.3.1. Conjugué d'un élément de GF(2^m) et propriétes

Soit F un polynôme dont les coefficients appartiennent à GF(2). On pose α un élément non nul de GF(2^m). Si α est un racine de F (i.e. F(α) = 0) alors pour tout l supérieur ou égal à zéro α^{2^l} est aussi racine de F. α^{2^l} est appelé le *conjugué* de α

On a $F^2(x) = \left(\sum_{i=0}^{m-1} f_i x^i\right)^2 = f_0^2 + 2f_0\left(\sum_{i=0}^{m-2} f_i x^i\right) + \left(\sum_{i=0}^{m-2} f_i x^i\right)^2$

Or les coefficients f_i appartiennent à GF(2). Par conséquent, $2f_0$ est égal soit à (0+0) soit (1+1) c'est à dire que l'on a $2f_0 = 0$ (addition modulo 2).

D'où $F^2(x) = \left(\sum_{i=0}^{m-1} f_i x^i\right)^2 = f_0^2 + \left(\sum_{i=0}^{m-2} f_i x^i\right)^2$. On peut appliquer le même raisonnement pour la somme $\sum_{i=0}^{m-2} f_i x^i$. De proche en proche, on arrive alors au résultats suivant : $F^2(x) = \sum_{i=0}^{m-1} (f_i x^i)^2$.

Or les coefficients f_i appartiennent à GF(2), on a donc $(f_i)^2=f_i$. Par conséquent :

$$F^2(x) = \sum_{i=0}^{m-1} f_i (x^2)^i = F(x^2)$$

On en déduit alors que :

$$\forall k \geq 0, [F(x)]^{2^k} = F(x^{2^k})$$

D'où si α est une racine de F(x), alors α^{2^k} est aussi une racine de F(x).

On peut alors prouver les deux propriétés suivantes :

Les 2^m-1 éléments non nuls de GF(2^m) forment toutes les racines de x^{2^m-1}.

Les éléments de GF(2m) constituent toutes les racines de $x^{2^m} + x$

II.3.2. Polynôme minimal et propriétés

Comme chaque éléments α de GF(2^m) est racine du polynôme $x^{2^m} + x$, il peut être racine d'un polynôme de degré inférieur à 2^m. On appelle *polynôme minimal* $\phi(x)$ le polynôme de degré minimal sur GF(2) tel que $\phi(\alpha) = 0$. Ce polynôme est irréductible.

On peut alors démontrer le théorème suivant :

Soit $\phi(x)$ le polynôme minimal d'un élément α de GF(2^m). On appelle *e* le plus petit entier tel que $\alpha^{2^e} = \alpha$. Alors on a :

$$\phi(x)=\prod_{i=0}^{e-1}(x+\alpha^{2^i})$$