Code Reed-Solomon (127, *k*, *d*) avec effacements : simulation et conception sur réseaux de circuits programmables (FPGA)

(127, *k*, *d*) Reed-Solomon code with erasures : simulation and Field Programmable Gate Arrays (FPGA) design

par Abbas DANDACHE*, Thierry VALLINO*, Fabrice MONTEIRO* et Jean-Pierre DELAHAYE**

* LICM/CLOES/SUPELEC, Université de Metz,
2, rue E. Belin, 57078 Metz Cedex 03, France
Tel : +33 (0)3 87 54 73 00, Fax : +33 (0)3 87 20 33 87
Email : dandache@ese-metz.fr
** TDF-C2R
Département REseaux et techniques Informatiques (RETI)
Laboratoire Techniques et Architectures Informatiques (TAI)
Technopole Metz 2000
1 Rue Marconi, 57078 Metz Cedex 03
Tel : +33(0)3 87 20 75 22, Fax : +33 (0)3 87 74 54 30
Email : jean-pierre.delahaye@c2R.tdf.fr

résumé et mots clés

Les applications actuelles de télécommunications nécessitent la transmission de données aussi diverses que le son, la vidéo, la messagerie et les données de mesures, de signalisations et d'assistance. Cela entraîne une complexité croissante des systèmes de transmission et un débit de plus en plus élevé. A la réception, le système doit pouvoir détecter et corriger rapidement les éventuelles erreurs dues au bruit de canal (diminution du taux d'erreurs).

Une des techniques pour diminuer ce taux est d'utiliser un code détecteur correcteur d'erreurs adapté à l'application (codes cycliques, code convolutif, ..). Plus spécifiquement, cet article concerne un code détecteur correcteur d'erreurs Reed-Solomon (127, k, d) avec la description complète d'une technique de marquage des symboles pour la mise en œuvre des effacements. L'algorithme de codage calcule les mots de code et marque les symboles. L'algorithme de décodage opère soit sur les erreurs $t' = t$, soit sur les effacements $e' = 2*t$, soit sur un panachage des deux ($e' + 2*t' \leq d-1$), t étant le nombre maximum d'erreurs corrigibles. En plus la détection des erreurs est possible pour un nombre d'effacements supérieur à $2*t$.

Dans le cadre d'une étude menée conjointement entre le laboratoire LICM et TDF-C2R, plusieurs distances Hamming du code Reed-Solomon (127, k, d) ont été simulées (entre autres à partir de mesures réelles). Les résultats de simulation permettent de quantifier la valeur ajoutée concernant les effacements. De plus, la conception sur FPGA d'un code de Reed-Solomon (127, 121, 7) est étudiée afin d'implanter une fonction « codeur/décodeur avec effacements », pouvant être réutilisée lors de la synthèse d'autres applications traitant des flots de données en continu.

Transmission numérique, codes détecteurs correcteurs d'erreurs, Reed-Solomon, effacements, FPGA.

abstract and key words

Telecommunication applications require transmitting data with different format such as sound, video, email, measures, signalling and help contents. This leads to a growing complexity of transmitting systems and to higher and higher data rates. On reception, the system must be able to quickly detect and correct errors due to the transmission channel noise (decreasing error rate).

Error detecting-correcting codes suited to applications reduce the error rate (cyclic codes, convolutional code...). This paper presents an overview of the implementation of a (127, k, d) Reed-Solomon error-correcting code with erasures. The technology used to mark on symbols is described in details here.
The coding algorithm computes the codewords and marks the symbols. The decoding algorithm detects and corrects either the errors $t' = t$, or the erasures $e' = 2*t$, or a combination of the two ($e' + 2*t' \leq d-1$). The error detection is possible for a number of erasures exceeding $2*t$. The number of rectifiable errors is t. This work is the result of the collaboration between the LICM laboratory and TDF-C2R company. Many Hamming distances of a (127, k, d) Reed-Solomon error-correcting code with erasure have been tested with measure files, simulating different real environments. Results obtained from computer simulations using diversified environment models are in good agreement with analytical results. Moreover, the core of the «(127, 121, 7) Reed-Solomon code with erasures» coder/decoder has been implemented on an ALTERA/FLEX10K family FPGA from a VHDL specification. This core can be used to design applications with continuous data streams.

Error detecting correcting codes, Reed-solomon, Erasures, FPGA.

1. introduction

Les techniques numériques sont à nos portes, leur utilisation s'étend et touche les domaines du son, de la vidéo et des données. Il n'y a plus de supports dédiés, l'acheminement du son, de la vidéo et des données en général se fait en empruntant des médiums divers (câble, hertzien, RTC, satellite, ...). Ces médias pouvant être considérés comme imparfaits, cela peut entraîner une modification du message émis. L'imprévisibilité du message émis par la source impose alors au récepteur l'utilisation de techniques lui permettant de vérifier à la fois l'exactitude et la certitude de l'information reçue.

Afin de diminuer le taux d'erreurs dans le message, des symboles sont rajoutés au message suivant une loi connue à la fois de l'émetteur et du récepteur. Deux types de techniques existent :

• une technique qui détermine uniquement si le message reçu est entaché d'erreurs. On parle alors de codes détecteurs d'erreurs. Comme il n'y a que détection des erreurs, cela entraîne une retransmission du code reçu faux détecté comme tel. Cette stratégie nécessite cependant un canal de retour du type half ou full duplex. La répétition se fait à la requête du récepteur,

• une technique qui permet de détecter et corriger (dans une certaine mesure) les erreurs présentes dans le message. On parle de codes détecteurs correcteurs d'erreurs.

Les codes Reed-Solomon [1], [2], [3], [4] sont des codes détecteurs correcteurs d'erreurs pouvant corriger les erreurs indépendantes, les symboles erronés contigus dans la mesure où le nombre d'erreurs ne dépasse pas la capacité de correction. Le décodage en dehors de la détection d'erreurs nécessite une unité de calculs opérant sur le corps de Galois.

L'article présente le code détecteur correcteur d'erreurs Reed-Solomon $(127, k, d)$, la technique utilisée pour marquer les symboles, les algorithmes de codage et de décodage, les simulations entreprises et la conception sur FPGA d'un code Reed-Solomon $(127, 121, 7)$.

2. code reed salomon avec effacement

2.1. code Reed-Solomon (127, k, d)

Ce sont des codes en bloc possédant toutes les propriétés des codes cycliques. Un polynôme irréductible et primitif $P(x)$ de degré p sert à générer les éléments du corps de Galois $(2p)$ utilisés dans tous les calculs. Ces codes travaillent sur des symboles de p bits. La distance minimum d renseigne sur la capacité de correction du code [1], [2], [3].

Le code Reed-Solomon $(127, k, d)$ est entièrement défini par le polynôme générateur $g(x)$. Le polynôme irréductible et primitif est de la forme $P(x) = x^7 + x^3 + 1$. Le corps de Galois (2^7) contient 127 éléments et α est une racine de $P(x)$. Le polynôme générateur $g(x)$ caractérise entièrement les propriétés de ce code en matière de détection et de correction. La taille des symboles est de 7 bits.

2.2. codage

La distance Hamming d permet de déterminer la capacité de correction du code détecteur correcteur d'erreurs. Les paramètres n, k et d sont définis ci-dessous :

– $d = 2*t+1$ ou t représente le nombre d'erreurs corrigibles,

– taille du message $k = 2p - 1 - 2*t$,

– longueur du code $n = 2p - 1 = 127$, p étant égal à 7.

Selon la distance d, l'algorithme de codage calcule le polynôme générateur $g(x)$.

$$g(x) = \prod_{i=0}^{(d-2)} (x - \alpha^i) = (x - \alpha^0)(x - \alpha^1)\dots(x - \alpha^{(d-2)}) \quad (1)$$

Pour le code Reed-Solomon $(127, k, d)$, les éléments d'informations $M(x)$ peuvent se mettre sous la forme suivante :

$$M(x) = \sum_{i=k=1}^{0} a_i x^i = a_{(k-1)}x^{(k-1)} + \ldots + a_1 x^1 + a_0 x^0 \quad (2)$$

avec $a_i \in \{0, 127\}$.

La redondance est le reste de la division de $X^{(n-k)} * M(X)$ par le polynôme $g(X)$.

L'addition des coefficients est une arithmétique modulo deux. On peut alors écrire le reste sous forme de somme :

$$R(x) = \sum_{j=n-k-1}^{0} r_j x^j = r(n-k-1)x^{(n-k-1)} + \ldots + r_1 x^1 + r_0 x^0 \quad (3)$$

avec $r_j \in \{0, 127\}$.

Le reste $R(x)$ ainsi obtenu complète le message pour former le mot de code $C(x)$, ainsi l'expression littérale de $C(x)$ est donnée ci-dessous :

$$C(x) = x^{(n-k)} \sum_{i=k=-1}^{0} a_i x^i + \sum_{j=n-k-1}^{0} r_j x^j \quad (4)$$

Le codage est systématique. Les coefficients des polynômes $M(x)$, $R(x)$ et $C(x)$ peuvent être représentés soit sous forme de valeurs discrètes comprises entre 0 et 127, soit sous forme de puissance de α.

2.3. technique de marquage des symboles

Les codes Reed-Solomon utilisent des symboles. Chaque symbole est composé de 7 bits. Pour une meilleure compréhension de la suite de l'article, nous allons donner quelques définitions.

Une **erreur** est un symbole erroné à l'intérieur d'un mot de code reçu $C'(x)$ dont la position et la valeur corrective de son coefficient sont inconnues.

Un **effacement** est un symbole erroné à l'intérieur d'un mot de code reçu $C'(x)$ dont la position est clairement repérée. La valeur corrective de son coefficient n'est pas connue.

La **correction d'erreurs et des effacements** consiste à restituer la valeur d'origine de chacun des symboles erronés d'un mot de code reçu $C'(x)$

La **détection des erreurs** consiste à déceler l'altération des symboles à l'intérieur d'un mot de code reçu $C'(x)$.

La mise en œuvre des effacements permet d'améliorer de manière significative la capacité de correction du code Reed-Solomon $(127, k, d)$ d'origine [4]. Pour pouvoir traiter les effacements, il faut mettre en œuvre un moyen de marquage de chaque symbole. Dans notre cas, nous avons utilisé :

- sept bits comme taille de symbole. L'amplitude du symbole ou coefficient appartient donc à l'ensemble des valeurs discrètes comprises entre 0 à 127,
- un bit de parité comme moyen de marquage de chaque symbole (on travaille alors avec des octets).

Le bit de parité permet de séparer les erreurs des effacements. Notons au passage que le bit de parité permet de détecter toutes les erreurs dont le nombre est impair.

2.4. code Reed-Solomon (127, k, d) modifié

Pour prendre en compte les effacements, le code Reed-Solomon $(127, k, d)$ est modifié comme indiqué ci-dessus. Une fois que le mot de code $C(x)$ est formé, tous les symboles sont de 8 bits. Le bit de parité est calculé symbole par symbole tant pour le message que pour la redondance. Pour cela l'algorithme de codage divise les symboles pris individuellement par $P_1(x) = x + 1$, rajoute le reste d'un bit au symbole de 7 bits. Les caractéristiques du code Reed-Solomon $(127, k, d)$ ainsi formé sont :

– longueur n est de 127 symboles,

– taille du message est de k symboles,

– distance minimum est d,

– nombre d'erreurs corrigibles est t,

– nombre maximal d'effacements est $2 * t$,

– correction conjointe erreurs t' et effacements e' avec $2*t'+e' \leq d-1 = 2*t$,

– détection d'erreurs pour un nombre d'effacements supérieur à $2 * t$,

– 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 Reed-Solomon $(127, k, d)$ d'origine.

2.5. décodage

Le mot de code $C(x)$ diffusé ou transmis peut subir des altérations dues à l'environnement. Le mot de code reçu $C'(x)$ est égal à :

$$C'(x) = [C(x) + E(x)] \text{ Mod } 2. \quad (5)$$

$E(x)$ représente l'expression polynomiale des erreurs. Les coefficients b_j ont une amplitude comprise entre 0 et 127. Le coefficient dont la valeur est égale à 0 ne sera pas écrit dans $E(x)$.

$$E(x) = \sum_{j=n-1}^{0} b_j x^j = b_{(n-1)}x^{(n-1)} + \ldots + b_1 x^1 + b_0 x^0 \quad (6)$$

avec $b_j \in \{0, 127\}$

Dans la mesure où le code Reed-Solomon $(127, k, d)$ n'est pas en dépassement, nous avons deux scenarii possibles :

– $E(x) = 0$ donc $C'(x)$ est identique à $C(x)$,

– $E(x) \neq 0 \quad C'(x) = [C(x) + E(x)] \text{Mod } 2$.

L'algorithme de décodage permet de corriger les erreurs, traiter ou non les effacements [4]. La prise en charge des effacements par le décodeur donne une valeur ajoutée car on améliore notablement la capacité de correction. A cela il faut ajouter aussi la possibilité de détecter les erreurs lorsque le nombre d'effacements dépasse $2 * t$.

Le décodage du code Reed-Solomon $(127, k, d)$ modifié demande plusieurs étapes de calculs et les polynômes définis ci-dessous sont utilisés :

– calcul du nombre e' et des positions des effacements à l'intérieur du mot de code reçu $C'(x)$,

– si $e' > 2 * t$, la détection d'erreurs est activée. Autrement la procédure correction d'erreurs et d'effacements est mise en œuvre.

Polynôme localisateur d'effacements $\sigma_e(x)$

$$\sigma_e(x) = (1 + \alpha^{i_1}x)(1 + \alpha^{i_2}x)\ldots(1 + \alpha^{i_J}x) \tag{7}$$

avec $j \in \{1, 2, 3, \ldots, e'\}$,

j étant le rang des effacements,

e' étant le nombre d'effacements calculé à la réception avec $e' \leq 2 * t$,

$i_1, i_2, \ldots, ij \in \{0, 126\}$.

Syndrome $S(x)$

$$S(x) = \sum_{i=1}^{(d-1)} S_i x^{(i-1)} = S_1 + S_2 x + S_3 x^2 + \ldots + S(d-1)x^{d-2}. \tag{8}$$

$$S_i = C'(x) \quad \text{avec} \quad x = \alpha^{(i-1)} \quad \text{et} \quad i \in \{1, d-1\} \tag{9}$$

d étant la distance de Hamming.

Polynôme $U(x)$

Le polynôme $U(x)$ est obtenu à partir de $\sigma_e(x)$ et de $S(x)$.

$$U(x) = \sigma_e(x)S(x) \tag{10}$$

Polynômes de Forney $\varepsilon(x)$ et $T(x)$

L'algorithme de Berlekamp-Massey a été légèrement modifié pour pouvoir utiliser la méthode de Forney. Ainsi les effacements sont pris en compte. Forney déduit deux polynômes $\varepsilon(x)$ et $T(x)$.

$$U(x) = \varepsilon(x) + T(x)x^{e'}. \tag{11}$$

le degré de $\varepsilon(x) < e'$, e' étant le nombre d'effacements.

Polynôme localisateur d'erreurs et d'effacements

L'algorithme de Berlekamp-Massey modifié permet de trouver le polynôme localisateur d'erreurs et d'effacements $\sigma(x)$ et de déduire le polynôme localisateur d'erreurs $\beta(x)$.

$$\sigma(x) = \sigma_e\beta(x) \tag{12}$$

Polynôme localisateur d'erreurs $\beta(x)$

Les racines du polynôme localisateur $\beta(x)$ sous forme de puissance α permettent de déterminer les positions des erreurs à l'intérieur du mot de code reçu $C'(x)$.

Polynôme évaluateur d'erreurs $g(x)$

$$\gamma(x) = \beta(x))T(x) \mod_{x^{(2t-e')}} \tag{13}$$

Valeurs correctives des erreurs t' et des effacements e'

$$e_i = \frac{-\alpha^{i(1-e')}\gamma(\alpha^{-i})}{\sigma e(\alpha^{-i})\beta'(\alpha^{-i})} \tag{14}$$

pour une erreur à la position i.

$$e_j = \frac{-\alpha^{j(1-e')}\gamma(\alpha^{-j})}{\sigma' e(\alpha^{-j})\beta(\alpha^{-j})} - \alpha^j \frac{\varepsilon(\alpha^j)}{\sigma'_e(\alpha^{-j})} \tag{15}$$

pour un effacement à la position j.

$\beta'(x)$ et $\sigma'_e(x)$ sont les dérivées de $\beta(x)$ et de $\sigma_e(x)$, avec i et $j \in 0, 1, 2, 3, \ldots, 2^n - 2$.

3. simulation logicielle

La simulation logicielle joue un rôle important, c'est un passage incontournable avant la mise en œuvre :

- soit logicielle sur des applications déjà en place,
- soit matérielle pour des applications où le temps réel est exigé.

La simulation logicielle permet entre autres de réaliser et de tester les algorithmes de codage et décodage, d'améliorer et d'évaluer les performances des algorithmes réalisés par l'utilisation des différents modèles d'environnement. Nous définissons le modèle d'environnement comme un ou plusieurs fichiers d'erreurs de format binaire de taille pouvant atteindre plusieurs Mega-octets.

Les modèles aléatoires sont des fichiers d'erreurs de format binaire obtenus par un utilitaire logiciel distribuant aléatoirement les erreurs selon la taille en octets du fichier souhaitée et un taux d'erreurs bit donné.

Les fichiers de mesures (diffusion hertzienne en bande II, mesures en rétrodiffusion) permettent l'approche d'un environnement réel. C'est la méthode couramment utilisée à TDF (Télédiffusion De France). Les fichiers de mesures de format binaire sont très fiables et reproductibles, par conséquent ils permettent de tester, de valider et de comparer avec le même modèle d'environnement.

Tableau 1. – Résultats des simulations.

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 %

Les mesures ont été réalisées sur le terrain avec tout l'environnement analogique (modulateur, émetteur, récepteur, ...). Le principe consiste à comparer à la réception les séquences pseudo-aléatoires émises avec celles synchronisées et générées localement. A l'issue de cette comparaison, les éléments binaires de même nature sont représentés par un élément binaire « 0 » et ceux de nature différente sont représentés par un élément binaire« 1 ». Les éléments binaires issus de cette comparaison sont concaténés par segment de 8 bits, ensuite l'octet ainsi formé est enregistré successivement sous forme de fichier d'erreurs de format binaire. L'octet est l'élément caractéristique de base des fichiers d'erreurs et chaque bit de poids allant de 1 à 128 peut prendre deux valeurs :

- un 0 représente la transparence du support utilisé,
- un 1 signale une altération du bit à cette position.

3.1. principe d'une simulation logicielle

La figure 1 présente le principe d'une simulation logicielle. Le codage consiste d'une part à former le mot de code $C(x)$ du code Reed-Solomon $(127, k, d)$ d'origine et d'autre part à calculer le bit de parité de chaque symbole. L'injection d'erreurs consiste à faire une addition modulo 2 entre les octets des mots de code $C(x)$ et ceux du fichier d'erreurs représentant un modèle d'environnement. Nous obtenons l'équivalent d'un message codé reçu $C'(x)$ qui sera décodé. L'algorithme de décodage décèle les effacements e' et les comptabilisent. La détection des erreurs est activée si le nombre d'effacements e' dépasse $2 * t$. Autrement les

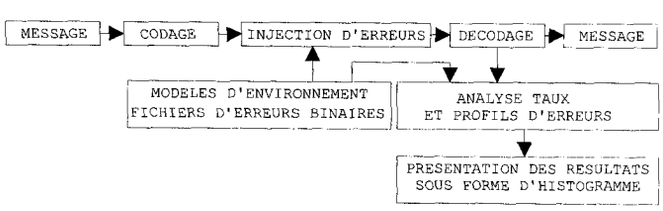


Figure 1. – Principe d'une simulation logicielle.

erreurs t' et les effacements e' sont conjointement prises en charge et enfin le message est restitué. Les erreurs induites par le modèle d'environnement sont analysées et les résultats se présentent sous forme de taux et profils d'erreurs.

Le tableau 1 donne un aperçu des simulations entreprises sur des modèles aléatoire et hertzien. Pour la simulation du code Reed-Solomon $(127, k, d)$ pour un modèle aléatoire, le fichier d'erreurs de format binaire contient uniquement des erreurs isolées et le taux d'erreurs bits est de 1, 12.10^{-2}. Dans le cas d'un code Reed-Solomon (127, 105, 23) sans effacements, le taux d'erreurs résiduelles est de 1, 82.10^{-3}. Le code est en dépassement car non seulement il n'a pas pu corriger toutes les erreurs, il en a rajouté d'autres. Pour le même modèle aléatoire, deux longueurs de code Reed-Solomon $(127, k, d)$ résolvent le problème :

– code Reed-Solomon (127, 105, 23) avec effacements,

– code Reed-Solomon (127, 89, 39) sans effacements.

Dans le cas de la simulation du code Reed-Solomon $(127, k, d)$ pour un support hertzien en bande II (ce modèle est 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. Pour des taux d'erreurs inférieurs à 1, 00.10^{-2}, l'écart du rendement entre un code Reed-Solomon $(127, k, d)$ sans effacements et celui avec effacements est faible. Par contre la distance Hamming se fait plus sentir :

– code Reed-Solomon (127, 65, 63) avec effacements nécessite une capacité de correction t = 31 pour un taux d'erreurs résiduelles nul. Il n'y a pas de fausses corrections en cas de dépassement du code. Au-delà de $2 * t$, les erreurs sont détectées,

Tableau 2. – Comparaison des performances.

Code Reed-Solomon *(127,k,d)*		
	DECODAGE	
Paramètres du code	Sans effacements	Avec effacements
Rendement (*k/n*) en %	*k*/127	7*k*/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'≤ d-1	Non	Oui
Distance Hamming d	2*t + 1	2*t + 1

Tableau 3. – Comparaison avec d'autres codes.

Code détecteur correcteur d'erreurs	Rendement	Profils d'erreurs corrigibles
Code de Fire (35,27)*	77,14 %	Correction d'un paquet d'erreurs de taille ≤ 3.
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.

– code Reed-Solomon (127, 57, 71) sans effacements nécessite une capacité de correction $t = 35$ pour un taux d'erreurs résiduelles nul. De fausses corrections apparaissent en cas de dépassement du code. La détection des erreurs n'est pas prévue.

3.2. comparaison des performance

Le tableau 2 permet de comparer les performances du décodage du code Reed-Solomon $(127, k, d)$ selon l'option « avec ou sans effacements ». Les cinq avantages l'emportent largement sur le seul inconvénient (rendement légèrement plus faible).

Les atouts majeurs sont la détection des erreurs pour un nombre d'effacements $e' > 2 * t$, la correction conjointe des erreurs t' et des effacements e' et l'absence de fausses corrections en cas de dépassement de la capacité de correction du code détecteur correcteur d'erreurs.

Le tableau 3 permet de comparer le code Reed-Solomon $(127, k, d)$ à d'autres codes détecteurs correcteurs d'erreurs ayant une capacité de correction équivalente. 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. Le code de Fire $(35, 27)*$ corrige qu'un seul paquet d'erreurs de taille ≤ 3 et peut détecter dans certains cas des paquets d'erreurs. Sa présence dans le tableau ci-dessous est à titre indicatif.

4. conception sur FPGA

Plusieurs études sur des « codeurs/décodeurs », de Reed-Solomon ont déjà été réalisées tant dans le domaine universitaire [5], [6] qu'industriel [7], [8]. Chacune de ces études autorise l'utilisation des effacements (en général, une broche est dédiée à cette opération) mais aucun n'implante réellement une méthode de discrimination des erreurs des effacements. Nous avons donc étudié l'implantation sur FGPA d'une fonction « codeur/décodeur de Reed-Solomon avec effacement » qui puisse être réutilisée comme composant pour la synthèse d'autres applications traitant des flots de données en continu. Pour cela, l'ensemble des blocs fonctionnels a été étudié avec pour objectif une architecture pipeline.

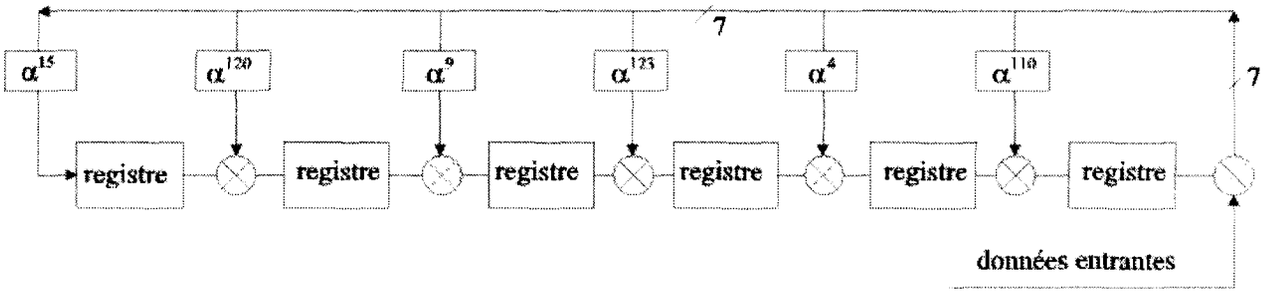


Figure 2. – Circuit de calcul du reste.

4.1. multiplieur dans un corps de GALOIS

Le multiplieur dans un corps de Galois est un des composants principaux utilisé 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. Nous avons donc utilisé un multiplieur de Mastrovito qui utilise une matrice pour le calcul du produit de deux vecteurs.

Pour le calcul du syndrome, nous avons alors utilisé 6 multiplieurs par une constante qui ont été optimisés en utilisant l'algorithme décrit dans [9]. Cette solution est préférable à l'utilisation d'un multiplieur « classique » qui présente des temps de propagation plus importants [6].

4.2. le codage

Dans cette étude, une seule distance de Hamming ($d = 7$) est utilisée pour la validation de l'architecture des codeurs/décodeurs sur FPGA.

Pour le codeur Reed-Solomon $(127, 121, 7)$ considéré, deux circuits sont nécessaires :

• un circuit qui calcule le reste de la division de $X^7 * M(X)$ par le polynôme $g(X)$:

$$g(X) = X^6 + \alpha^{110} X^5 + \alpha^4 X^4 + \alpha^{123} X^3 + \alpha^9 X^2 + \alpha^{120} X + a^{15}.$$

Le circuit utilisé (figure 2) est constitué uniquement de registres de 7 bits et de portes XOR. Les multiplicateurs dans le corps de Galois (généré par le polynôme $P(x) = x^7 + x^3 + 1$) utilise l'architecture décrite dans [9].

• un circuit qui implante le marquage des symboles : à chaque symbole, on rajoute le bit de parité obtenu par l'utilisation d'un réseau de portes XOR.

4.3. le décodage

Pour le décodage, plusieurs cas peuvent se produire. 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 cas de la prise en charge des erreurs et des effacements, le temps (nombre de cycles) de traitement a fait l'objet de la conception d'une architecture sur FPGA. L'architecture étudiée pour le décodeur est celle présentée à la figure 4 et fonctionne de la façon suivante : les symboles de 8 bits constituant la donnée (trame de 127 symboles) sont envoyés simultanément dans deux circuits :

• un circuit calcule le bit de parité de chaque symbole reçu et détermine la présence ou non d'effacements. Les positions des effacements, le nombre total de ces effacements ainsi que les α^i (i étant le rang d'un effacement) sont stockés dans des registres de 6 bits.

• un circuit de calcul du syndrome qui implante la formule (8) (paragraphe 2.5) en utilisant un processus récursif [3] pour le cal-

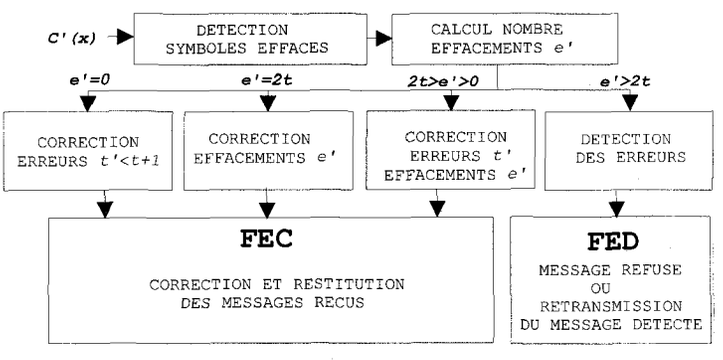


Figure 3. – Décodage optimisé du code Reed-Solomon $(127, k, d)$.

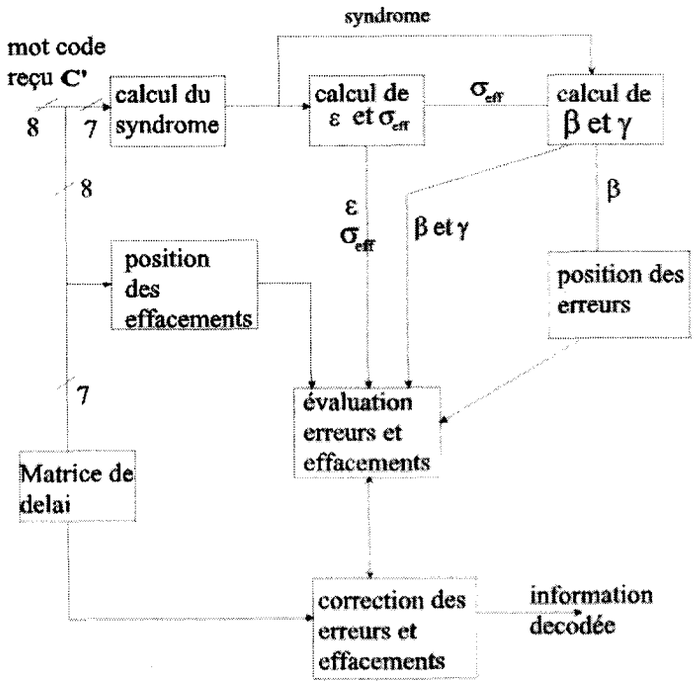


Figure 4. – Architecture utilisée pour le décodeur du Reed-Solomon $(127, k, d)$.

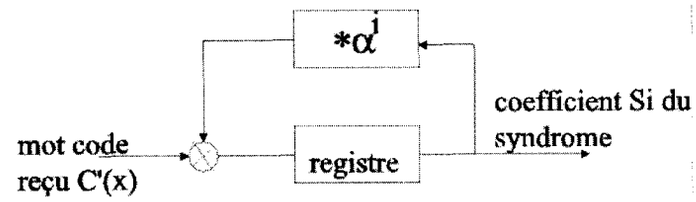


Figure 5. – Circuit de calcul des coefficients αi.

cul des coefficients du syndrome (figure 5). Tous les coefficients sont calculés en parallèle.

Les coefficients du syndrome sont alors utilisés comme entrée du circuit calculant les coefficients des polynômes $\sigma_{eff}(x)$ et $\varepsilon(x)$. Ces coefficients sont générés par deux systèmes d'équations récursifs [4].

$$\sigma_{j+1} = \sigma_j - \alpha_i \sigma'_j \quad \omega_{j+1} = \omega_j - \alpha_i \omega'_j + \Delta_j X^j \tag{16}$$

$$\sigma'_{j+1} = X\sigma_{j+1} \quad \omega'_{j+1} = X\omega_{j+1} \tag{17}$$

avec j qui désigne la position d'un effacement, et Δ_j le coefficient en X^j du produit $\sigma_j S$.

Considérons le système qui implante les formules suivantes :

$$u_{j+1} = u_j - a u'_j + bX^j \tag{18}$$

$$u'_{j+1} = X u_{j+1} \tag{19}$$

On peut remarquer que les équations (16) et (17) se déduisent de (18) et (19) par :

$$u_j = \sigma_j, a = \alpha_i, b = 0. \tag{20}$$

$$uj = \omega_j, a = \alpha_i, b = \Delta_j. \tag{21}$$

Le calcul des polynômes $\sigma_{\text{eff}}(x)$ et $\varepsilon(x)$ va donc utiliser deux circuits C1 et C2 implantant les équations (18) et (19) 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 6).

De plus, u_j et u'_j représentent chacun les coefficients d'un polynôme de degré égal au maximum à 6. Les équations (18) et (19) peuvent donc s'écrire respectivement comme :

$$u^{0'}_{j+1} = 0 \text{ et } u^{m'}_{j+1} = u^{m-1}_{j+1} \quad \text{pour} \quad m \in \{1, 2, \ldots, 5\}. \tag{22}$$

Ce qui revient à effectuer un décalage à droite entre u'_{j+1} et u_{j+1} (fig 7).

$$u^m_{j+1} = u^m_j - a u^{m'}_j + bX^j \quad \text{pour} \quad m \in \{0, 2, \ldots, 5\} \tag{23}$$

Le circuit Bi de la figure 8 implante les deux équations (22) et (23) d'un élément i utilisé dans les circuits $C1$ et $C2$.

Finalement, pour générer les deux polynômes, 12 multiplicateurs dans le corps de Galois et 30 registres de 7 bits ont été utilisés (mémorisation des coefficients de $\sigma, \sigma', \omega, \omega'$ et Δ). Au bout de s cycles de calcul (s étant le nombre d'effacements), on a les égalités :

$$\sigma = \sigma_{\text{eff}} \tag{24}$$

$$\omega = \varepsilon \tag{25}$$

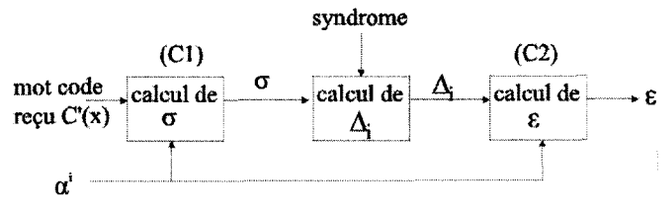


Figure 6. – Circuit de calcul des polynômes σ_{eff} **et** ε.

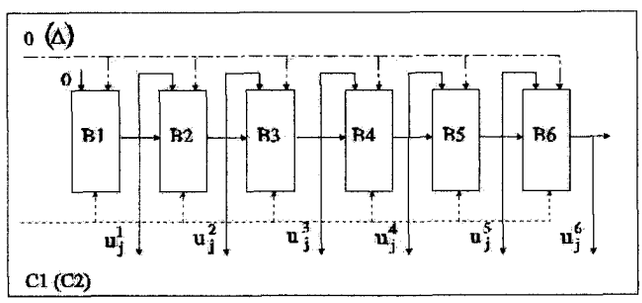


Figure 7. – Schéma d'implantation du système récursif $C1$ **(ou** $C2$**).**

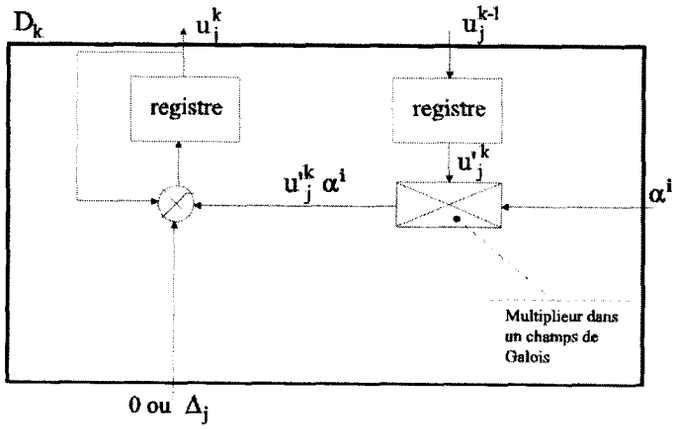


Figure 8. – Circuit Di utilisé pour l'implantation de la formule (24).

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 :

$$\sigma_{j+1} = \sigma_j - \Delta_j \sigma'_j \quad \omega_{j+1} = \omega_j - \Delta_j \omega'_j \tag{26}$$

$$\sigma'_{j+1} = X(C\cdot\sigma'_j + C\cdot\Delta^{-1} j\sigma'_j) \quad \omega'_{j+1} = X(C\cdot\omega'_j + C\cdot\Delta^{-1} j\omega'_j) \tag{27}$$

$$d_j = d_j \cdot C + (j + 1 + s - d_j)\cdot C \tag{28}$$

où C est le résultat du test ($\Delta_j = 0$ ou $2d_j > j + \sigma$)

Pour calculer ces deux polynômes, on utilise la même méthode que précédemment. Un circuit D_i (fig 9) implante ($C \cdot \sigma'_j + C \cdot \Delta^{-1} j\sigma'_j$) et l'équation (28). La réalisation de la multiplication par X se fait par décalage à droite comme précédemment (fig 10). Au bout de $(t - s)$ cycles de calcul, les polynômes σ et ω sont égaux respectivement à β et γ.

Le polynôme β étant connu, un calcul par une architecture pipeline de $\beta(\alpha^i)$ (fig 11) 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).

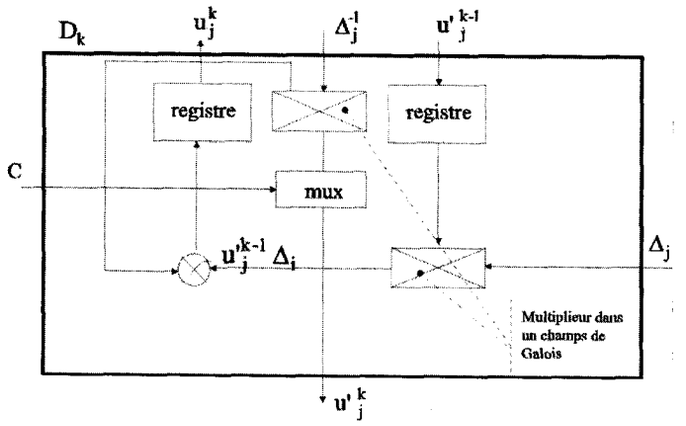


Figure 9. – Circuit utilisé pour le calcul des polynômes β et γ.

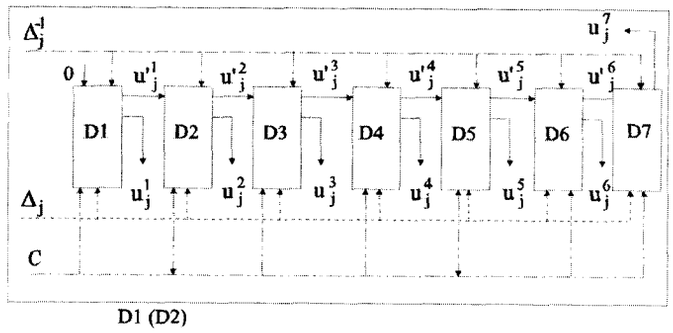


Figure 10. – Circuit utilisé pour le calcul des polynômes β et γ.

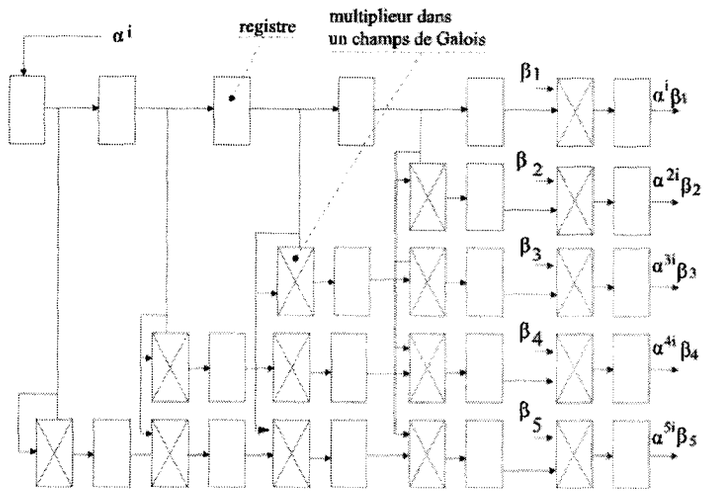


Figure 11. – Circuit de calcul $\beta(\alpha^i)$.

Quand les positions des effacements et des erreurs sont connues, on peut calculer les valeurs correctives. Pour ce faire, on a implanté des circuits de calcul des valeurs $\beta(\alpha^i)$, $\gamma(\alpha^i)$, $\sigma_{\text{eff}}(\alpha^i)$, $\varepsilon(\alpha^i)$, $\beta'(\alpha^i)$, $\gamma'(\alpha^i)$, $\sigma'_{\text{eff}}(\alpha^i)$, $\varepsilon(\alpha^i)$ ainsi que celui des inverses.

4.4. résultats

L'architecture du codeur/décodeur qui a été présentée dans les paragraphes précédents a été décrite en VHDL et implantée 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, 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 [7]. Ce nombre peut être amélioré par le développement d'une architecture dans laquelle chaque bloc sera pipeliné et/ou parallèlisé.

5. conclusions

Les codes Reed-Solomon sont de plus en plus utilisés, des circuits spécifiques permettent le décodage. Les effacements améliorent la capacité de correction. En général, cette possibilité n'est pas mise en œuvre.

La correction conjointe des erreurs et effacements apportent une valeur ajoutée aux codes Reed-Solomon et justifie sa place dans les domaines satellitaire, hertzien et du câble.

La validation par simulation basée sur des modèles d'environnement réel (fichiers d'erreurs de format binaire) couramment utilisés à TDF confirme la possibilité de panacher le « FEC » (Forward Error Correction) et le « 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 $e' + 2 * t' \leq d - 1$).

Les fichiers-source en C du simulateur logiciel sont disponibles et peuvent être réutilisés dans le cadre d'une application nécessitant une protection beaucoup plus grande. Les fichiers source en C permettent d'une part le codage des informations, et d'autre part le décodage complet avec l'option « avec ou sans effacements ». La taille des symboles ne pose pas de problème particulier. La distance minimum d peut être paramétrée.

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. La prochaine étape consiste à optimiser cette architecture.

La suite de l'étude consiste à mettre en œuvre :

- les effacements sur le code Reed-Solomon (204, 188, 17) largement utilisé dans les projets européens et le code Reed-Solomon (15, k, d),
- une stratégie de synchronisation interactive permettant l'option « avec ou sans effacements » suite aux requêtes du récepteur.

BIBLIOGRAPHIE

[1] Poli (A.), Huguet (L.). Codes correcteurs : théorie et applications. *Masson*, Paris (1989).

[2] Peterson (W.), Weldon (E. J. Jr). Error correcting codes. *Second edition* (1972).

[3] Lin (S.), Daniel (J.), Costello (Jr.). Error control coding : fundamentals and applications. *Editor Franklin* (F.), Kuo (1983).

[4] Cohen (G.), Dornstetter (J.-L.), Godlewski (P.). Codes correcteurs d'erreurs : une introduction au codage algébrique. *Masson*, Paris (1992).

[5] Heather (B.), Hui Zhang. Comparison of Reed-Solomon codec implementations. *CS252 Project*, Université de Berkeley.

[6] Dabbagh (A.). Etude et conception d'un circuit de détection et correction d'erreurs en transmission d'informations numériques. *Thèse présentée à l'université de Rennes I*, (1995).

[7] Reed Solomon decoders with erasures. Société *Hammer cores*, (mars 1999).

[8] AHA4011 : 10 Mbytes/sec Reed-Solomon Error correction device. *Product Specification*, Advanced Hardware Architectures.

[9] Paar (C.). A new architecture for a parallel finite field multiplier with low complexity based on composite Fields. *IEEE Transaction on computers*, July 1996, vol. 45, n° 7 pp 856-861

Manuscrit reçu le 26 novembre 1998.

LES AUTEURS

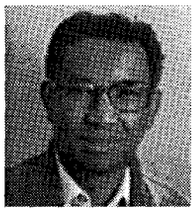
Abbas DANDACHE

Docteur en Micro-électronique de l'INPG de Grenoble (1986). Maître de Conférences à l'Université de Metz et coordinateur du groupe de recherche en micro-électronique au sein du laboratoire LICM/CLOES-SUPELEC. Son activité de recherche porte sur le thème «Conception de circuits et de systèmes pour des applications télécom».

Thierry VALLINO

Thierry Vallino est titulaire en 1995 d'un D.E.A. Instrumentation et Micro-électronique de Nancy-Strasbourg. Doctorant ANRT au LICM/CLOES SUPELEC depuis 1996 sous la direction du Pr B. Lepley et Mr A. Dandache, ses thèmes de recherche concernent l'étude et l'implantation en ASIC d'architecture parallèle pour des codes correcteurs d'erreurs à haut débit.

Jean Pierre DELAHAYE



Jean-Pierre Delahaye est Ingénieur d'études à TDF-C2R (Département RETI, Laboratoire TAI). Il travaille sur différents supports de sécurité (clé physique de protection, cartes à puce) pour renforcer la sécurité globale d'un système d'échanges de fichiers informatiques utilisant des algorithmes cryptographiques. Ses thèmes de recherche portent aussi sur les codes détecteurs correcteurs d'erreurs.

Fabrice MONTEIRO

Docteur en «Composant, Signaux, Systèmes» (LIRMM, Université de Montpellier, 1992) est Maître de Conférences à l'Université de Metz. Son activité de recherche porte sur la conception de circuits et de systèmes pour des applications télécom, et plus particulièrement sur les aspects modélisation et synthèse comportementale.