Comparaison performances / complexité de décodeurs de codes BCH utilisés en turbo-décodage.

Sylvie Kerouédan, Patrick Adde, Pascale Ferry

ENST Bretagne
BP832, 29285 Brest Cedex, France

Sylvie.Kerouedan@enst-bretagne.fr, Patrick.Adde@enst-bretagne.fr
Pascale.Ferry@enst-bretagne.fr

Résumé – Cet article propose une étude comparée entre algorithme et architecture en vue de l'implantation sur silicium et plus particulièrement sur FPGA d'un circuit de turbo-décodage de codes BCH(128,120,4). L'utilisation du langage C — pour les simulations — et du VHDL — pour la synthèse — permettent de comparer les performances et la complexité du circuit en fonction de quelques paramètres essentiels au déroulement de l'algorithme de décodage comme le nombre de bits de quantification, le nombre de concurrents et le nombre de vecteurs de tests.

Abstract – This paper presents a comparison between algorithms and architectures in order to implement a turbo decoding integrated circuit for BCH(128,120,4) codes on silicon, especially on FPGA. The use of C language (as the simulation language) and VHDL language (to perform the synthesis of the circuit) allow performances and complexity to be compared according to some main algorithm parameters, such as the number of quantization bits, the number of concurrent words and the number of test vectors.

1. Introduction

Les turbo codes — codes correcteurs d'erreurs obtenus à partir d'une concaténation de plusieurs codes correcteurs classiques — ont, depuis leur invention en 1993 à l'ENST de Bretagne [1], montré leur puissance. Ils permettent, en effet, de s'approcher de la limite théorique fixée par Shannon, en mettant en œuvre une complexité moyenne. Il est possible d'utiliser différents types de codes simples pour réaliser des turbo codes. Nous étudierons plus particulièrement le décodage de codes produits utilisés dans un dispositif de turbo décodage, i.e. dans un processus de décodage itératif. Après avoir rappelé le principe de l'algorithme « optimal » de décodage [2], nous verrons la simplification [3,4] de cet algorithme en vue d'une implantation sur silicium. Enfin grâce à la description de l'algorithme en C, et à celle du décodeur élémentaire en VHDL, nous mettrons en parallèle l'amélioration des performances avec l'augmentation de la complexité des circuits.

2. Présentation de l'algorithme optimal de décodage de codes produits

2.1. Les codes produits

Les codes produits sont obtenus par concaténation de codes en blocs linéaires. Les propriétés du code produit sont héritées des codes qui le composent. Considérons deux codes C1 et C2 caractérisés par les paramètres (n_1,k_1,d_1) [1] et (n_2,k_2,d_2), les paramètres du code produit C1⊗C2 sont (n,k,d) tels que $n=n_1\times n_2$, $k=k_1\times k_2$, et $d=d_1\times d_2$. Le rendement de ce code est aussi égal au produit du rendement des codes C1 et C2. Pour cette étude, on choisit des codes BCH[2] étendus pour effectuer le produit BCH(128,120,4)⊗BCH(128,120,4), où les codes C1 et C2 sont égaux.

2.2. Le décodage

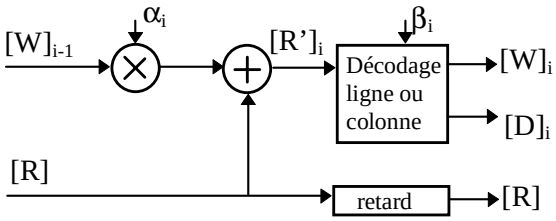

FIG. 1 : cellule élémentaire du décodage

Le décodage des codes produits proposé par R. Pyndiah et coauteurs dans [2] associe un décodage à entrée pondérée (type Chase) et sortie pondérée. L'algorithme itératif utilise des cellules élémentaires mises en cascade. Ce décodeur élémentaire est représenté sur la figure 1. Les différents paramètres et données sont :

- [**R**], le vecteur reçu (entrée pondérée),
- $[\mathbf{W}]_{i-1}$, l'information extrinsèque livrée par le décodage précédent (sortie pondérée),

[1] n_i représente la longueur du code, k_i, la longueur du message et d_i, la distance minimale de Hamming

[2] BCH pour Bose-Chaudhuri-Hocquenghem

- $[\mathbf{R'}]_i=[\mathbf{R}]+\alpha_i[\mathbf{W}]_{i-1}$
- $[\mathbf{D}]_i$, le résultat binaire du décodage courant

α_i et β_i sont des constantes, fonctions de la demi-itération courante.

Voici brièvement décrites, les étapes du décodage de la demi-itération *i* :

a) repérage des positions des *m* composantes les moins fiables (de I_1 à I_m) du mot reçu ;
b) génération de *t* séquences test $[\mathbf{T}]^j$ et calcul du mot binaire $[\mathbf{Z}]^j = [\mathbf{T}]^j \otimes \text{signe}[\mathbf{R}]$;
c) décodage algébrique des *t* mots $[\mathbf{Z}]^j$ par l'algorithme de Berlekamp (obtention du mot $[\mathbf{C}]^j$ et vérification du fait que ce résultat est un mot de code) ;
d) calcul de la distance euclidienne entre $[\mathbf{C}]^j$ et $[\mathbf{R}]$, $M^j=\|[\mathbf{R}]- [\mathbf{C}]^j\|^2$;
e) sélection du mot de code à distance minimale Md, c'est la décision binaire $[\mathbf{D}]$;
f) association d'une fiabilité à chaque bit d_j de $[\mathbf{D}]$, dont le calcul est basé sur l'estimation de Logarithme du Rapport de Vraisemblance (LRV),
   en fait pour chaque bit d_j, on cherche le mot de code concurrent $[\mathbf{C}^c]$, i.e. le mot à distance minimale Mc de la décision binaire $[\mathbf{D}]$ et tel que $C^c_j \neq d_j$, alors,
   – si ce mot existe, la fiabilité est $F_j=(Mc-Md)/4$
   – sinon, cette fiabilité est $F_j=\beta_i$
g) calcul de l'information extrinsèque à fournir aux itérations suivantes, $W_j=(F_j-C^d_j.R'_j).C^d_j$

L'itération de cet algorithme élémentaire permet d'obtenir l'effet « turbo » et donne donc de très bons résultats. Afin de tester et de valider cet algorithme, nous avons chercher à l'intégrer sur FPGA selon la démarche décrite dans [3]. Pour des codes assez long comme le BCH(128,120,4), le choix de paramètres comme le nombre de vecteurs de test semble améliorer sensiblement les performances sans augmenter de beaucoup la complexité, c'est pourquoi, nous avons choisi ce code pour étudier l'impact du choix de certains paramètres sur la complexité du circuit.

3. Architecture du décodeur

Pour implanter cet algorithme élémentaire, nous avons transcrit les étapes du décodage (pour une demi-itération) en blocs simples décrits en VHDL et assemblés comme le montre la figure 2 :

- Les fonctions d'entrée travaillent en parallèle et au fil de l'arrivée des données ; ces opérations sont le calcul du syndrome, le calcul de la parité, le tri des composantes les moins fiables (i.e. la mémorisation des symboles dont la fiabilité est la plus faible). 128 périodes d'horloges sont nécessaires à l'obtention de toutes les données.
- 128 nouvelles périodes d'horloge permettent de traiter chaque vecteur de test : calcul des métriques et tri.
- À l'issu de ces traitements, le mot « décidé » et le (ou les) mot(s) « concurrent(s) » sont connus, le calcul de la fiabilité des 128 symboles peut se faire. Ces fiabilités sont émises au fur et à mesure de leur calcul pendant à nouveau 128 périodes d'horloges.

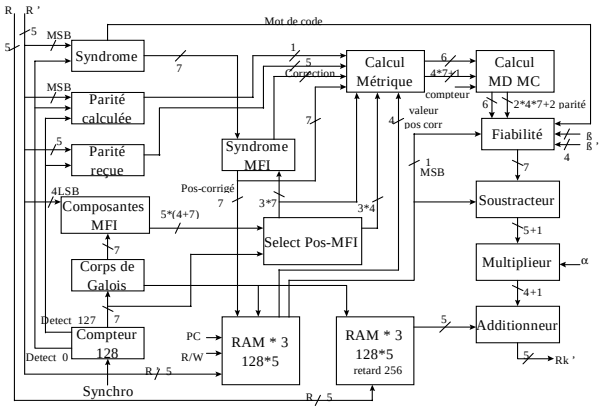

FIG. 2 : Schéma bloc du décodeur élémentaire

L'architecture choisie impose donc une latence[3] de 128×2 périodes d'horloge. Nous avons retenu 3 paramètres (nombre de bits de quantification, nombre de vecteurs de test et nombre de mots « concurrents » à retenir pour comparer performances et complexité. La mesure de surface sur FPGA (Xilinx) se fait en CLB[4], ce sera donc notre unité de comparaison de surface. Notons ici, que chaque CLB du composant retenu, famille Xilinx XC4000, contient deux générateurs de fonctions logiques à 4 entrées F et G, 1 générateur de fonction permettant de combiner les sorties des blocs F et G et une autre entrée, 2 bascules D à Set ou à Reset asynchrone.

4. Simplification de l'algorithme

4.1. Le nombre de concurrents

Dans l'algorithme de Chase-Pyndiah ou algorithme non-simplifié [2], s'il y a 16 vecteurs de test, il y a 16 mots concurrents à retenir : celui qui a la métrique la plus faible sera le mot « décidé » et les autres seront classés par ordre de métrique croissante. Il faut donc mémoriser ces 16 mots de 128 bits et leur métrique, ce qui fait plus de 150 bascules D rien que pour la mémorisation, il faut ensuite traiter ces différents concurrents... L'intégration de cet algorithme nécessite donc une augmentation en surface par rapport à l'intégration d'un algorithme simplifié à un seul concurrent qui peut dépasser les 60% [4], le gain en performance est lui de 0,13dB pour un TEB de 10^{-6} (figure 3).
Les blocs logiques qui vont le plus jouer dans l'augmentation de surface entre les 2 solutions qui sont de retenir 1 ou 3 concurrents sont « calcul_MD_MC » et « fiabilité » (figure 2).
Dans le tableau 1, on note que le passage de 1 à 3 concurrents se fait avec une augmentation de surface de 13,5% environ — le nombre de bascules est bien plus important lorsqu'il y a 3 concurrents, car cela correspond à la mémorisation de 4 mots

[3] i.e. la durée entre la réception et l'émission d'un symbole
[4] Configurable Logic Block ou cellule logique programmable

au lieu de 2 — et le gain en performance est de 0,07dB pour un TEB de 10^{-6} (figure 3).

TAB. 1 : Comparaison de surface entre les algorithmes à 1 concurrent et à 3 concurrents , avec 16 vecteurs de test et une quantification sur 5 bits.

	RAM	Reste du circuit	total	Nb de bascules
1 concurrent	240 CLB	545 CLB	785 CLB	275
3 concurrents	240 CLB	650 CLB	890 CLB	399

NB : Toutes les simulations sont effectuées pour un bruit blanc gaussien sur une modulation MDP4.

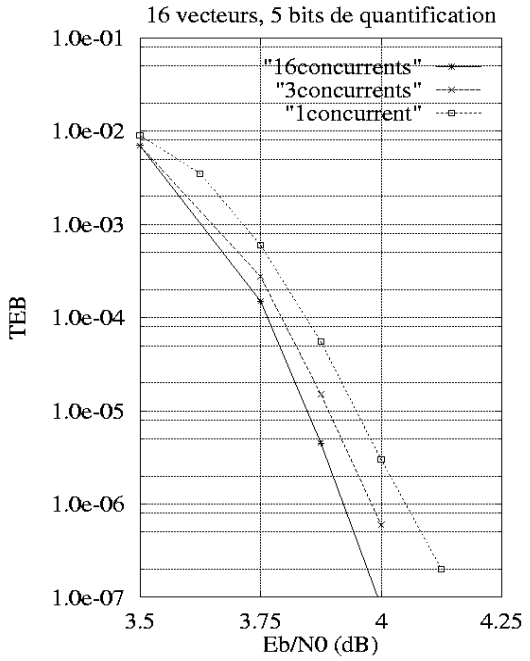

FIG. 3 : Nombre de concurrents et performances

4.2. Le nombre de bits de quantification

TAB. 2 : Comparaison de surface entre une quantification sur 4 ou 5 bits, avec 16 vecteurs de test et 1 concurrent.

	RAM	Reste du circuit	total	Nb de bascules
5 bits	240 CLB	545 CLB	785 CLB	275
4 bits	192 CLB	492 CLB	684 CLB	264

Le nombre de bits de quantification, correspond au nombre de bits permettant de coder les symboles du mot reçu — il comprend le nombre de bits utiles pour la « fiabilité » et le bit de signe. Pour passer de 4 à 5 bits de quantification, il faut mettre en œuvre des ressources mémoires supplémentaires essentiellement au niveau de la mémorisation des symboles. Les blocs qui vont en être affectés le plus en surface seront donc les RAM, le multiplieur (qui permet de construire l'information extrinsèque pour l'itération suivante) et dans une moindre mesure « composantes MFI » et « select_pos_MFI » (blocs qui doivent gérer les composantes les moins fiables). Il y a 6 RAM de 128 mots de 4 ou 5 bits.
Dans le tableau 2, on peut noter que le passage de 4 bits de quantification à 5 bits de quantification se fait avec une augmentation d'environ 15% de la surface.
Sur la figure 4, on peut voir qu'avec 5 bits de quantification plutôt que 4, les performances sont améliorées d'une décade à Eb/N0=4dB, ou de 0,1dB pour un TEB de 10^{-6}. Suivant la limite souhaitée, il peut s'avérer intéressant de gagner ces 0,1dB avec un coût minime sur la complexité. Il est à noter que les performances sont très fortement dégradées lorsqu'on quantifie sur 3 bits seulement, ce qui ne permet pas d'envisager l'intégration de l'algorithme dans cette configuration.

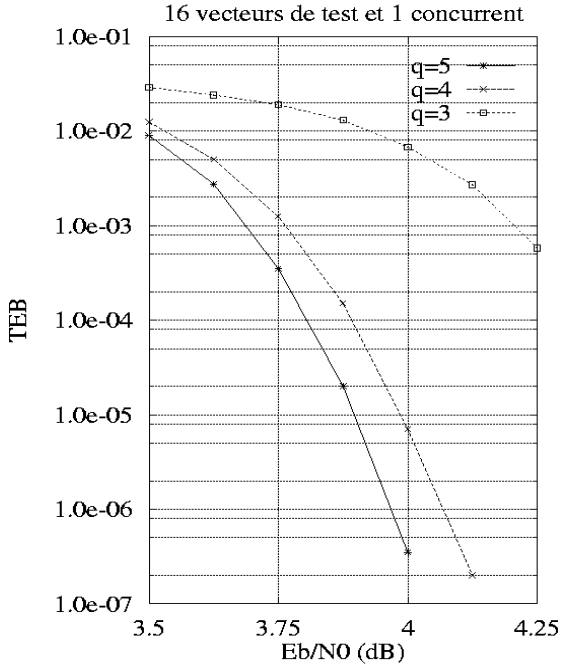

FIG. 4 : Quantification et performances

4.3. Le nombre de vecteurs de test

TAB. 3 : Comparaison de surface entre un algorithme à 8, 16 ou 32 vecteurs de test, 1 concurrent et une quantification sur 5 bits

	RAM	Reste du circuit	Total	Nb de bascules
8 vecteurs	240 CLB	513 CLB	753 CLB	253
16 vecteurs	240 CLB	545 CLB	785 CLB	275
32 vecteurs	240 CLB	640 CLB	880 CLB	329

L'augmentation du nombre de vecteurs de test augmente les ressources matérielles nécessaires : il faut en particulier retenir un plus grand nombre de composantes « les moins fiables » (CMF) et de vecteurs. Sur le schéma global (figure 2), les blocs qui vont jouer sur la surface seront

essentiellement les blocs entrant dans la construction des vecteurs de test « select_pos_MFI » et « syndrome_MFI » ainsi que le bloc de traitement des métriques « calcul_MD_MC ».
Dans le tableau 3, on note que le passage de 8 vecteurs de test à 16 vecteurs se fait avec une augmentation de surface de 4% tandis que l'amélioration des performances est de presque une décade à Eb/N0=4dB (figure 5) ou de 0,07dB pour un TEB de 10^{-6}. Pour ce qui est du passage de 16 vecteurs à 32 vecteurs, l'augmentation de surface est plus importante (12%) mais le gain en performance est plus faible (0,04dB pour un TEB de 10^{-6}).

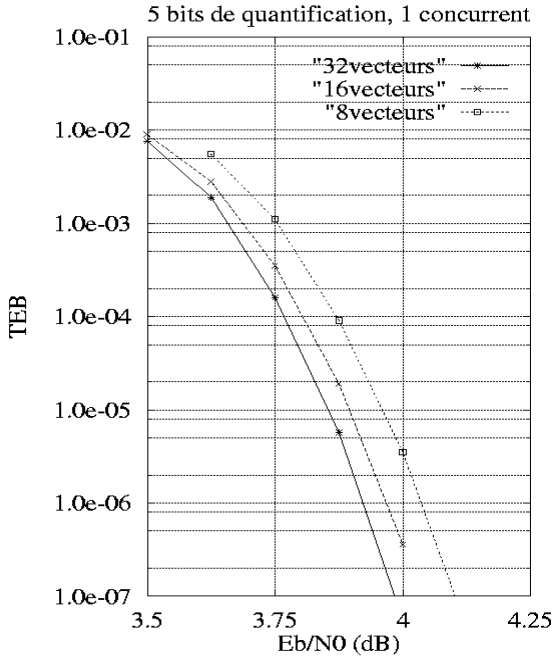

FIG. 5 : Vecteurs de test et performances

5. Conclusion

Cette étude a montré que certaines dégradations des performances réduisaient de manière significative la complexité d'un circuit : tout n'est qu'histoire de compromis. Toutes les simulations qui illustrent ces propos, ont été obtenues pour 6 itérations. Ceci correspond à 12 circuits FPGA 20000 portes contenant chacun une demi-itération (ce qui correspond à l'intégration du synoptique de la figure 2). Il y a d'autres moyens de gagner en surface sur un décodage complet :

- en réduisant le nombre d'itérations, par exemple avec 4 itérations. Dans ce cas, si l'on fait un décodage série 8 circuits sont suffisants au lieu de 12, d'où un coût allégé tant en surface qu'en prix, mais il y aura certainement des dégradations de performances ;
- en effectuant toutes les itérations sur le même « hardware », cette solution ne peut se faire qu'au prix d'une réduction de la fréquence maximale de fonctionnement du circuit.

La version qui nous semble être le meilleur compromis entre performances et complexité moyenne est le décodeur élémentaire qui utilise 5 bits de quantification, 16 vecteurs de test et 1 concurrent. Le décodage complet met en œuvre 4 itérations.

Références

[1] C. Berrou, A. Glavieux, P. Thitimajshima. « Near Shannon limit error-correcting coding and decoding : turbo-codes ». *IEEE proc. ICC'93, Geneva, May 1993.*

[2] R. Pyndiah, A. Glavieux , A. Picart et S. Jacq. « Near optimum decoding of product codes ». *Proc. Of IEEE GLOBECOM'94 Conference, vol.1/3, Nov.-Dec.1994, San-Francisco, pp 339-343.*

[3] P. Adde, R. Pyndiah, J.R. Inisan et Y. Sichez. « Conception d'un turbo décodeur de code produit ». *GRETSI'97, Grenoble, pp 1003-1010*.

[4] O. Raoul. « Conception et performances d'un circuit intégré turbo décodeur de codes produits ». *Thèse, Université de Bretagne Occidentale, 1997.*