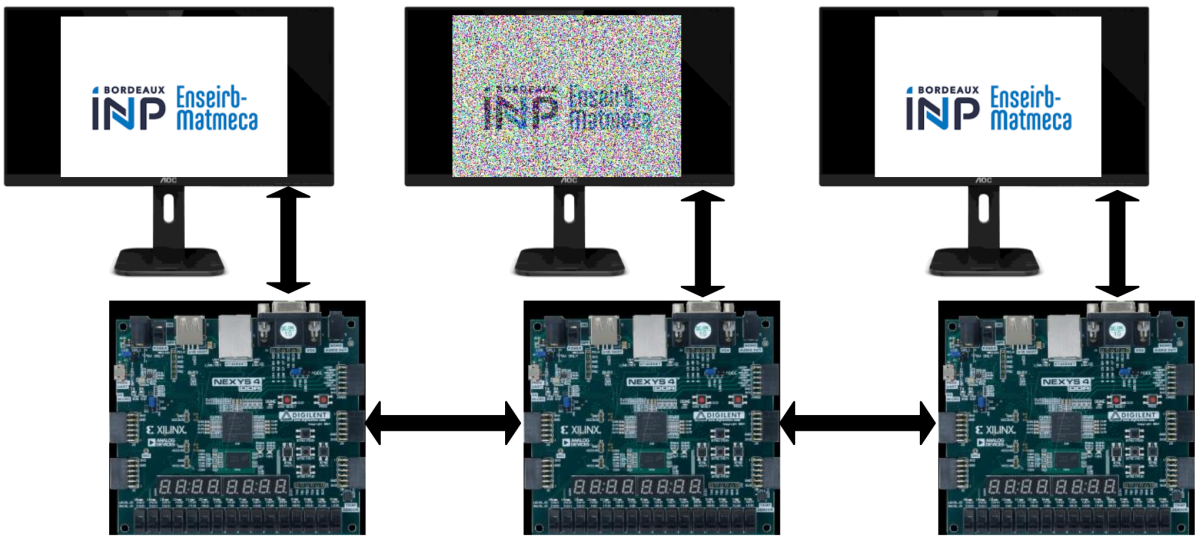
Proposition de projet thématique : Codes correcteurs d'erreurs sur les corps finis et applications à l'architecture numérique

Nasr-Allah HITAR

30 décembre 2025



1 Introduction

Contexte et motivation Les systèmes numériques modernes, en particulier ceux embarqués ou fonctionnant dans des environnements bruités (communications, stockage, calcul haute performance), sont exposés à des risques d'erreurs de données. Les codes correcteurs d'erreurs (ECC) permettent de détecter et de corriger ces erreurs de manière algébrique, garantissant ainsi l'intégrité des données. L'utilisation de corps finis fournit un cadre mathématique rigoureux pour la construction de codes efficaces (codes de Reed-Solomon, BCH, LDPC, etc.). Parallèlement, les architectures RISC-V ou STM32, de par leur ouverture et leur modularité, offrent un terrain idéal pour l'implémentation matérielle et les tests de mécanismes de correction d'erreurs, notamment sur FPGA (pour un RISC).

Objectifs du projet Ce projet vise à :

- Étudier les fondements mathématiques (on se limite aux bases) des codes correcteurs d'erreurs basés sur les corps finis (polynômes, arithmétique dans $\mathbb{F}_{2^m}$) et des bases en probabilités afin d'étudier la modélisation des canaux sans mémoire.
- Introduire la notion d'entropie de Shannon pour quantifier l'information transmise et reçue.
- Concevoir et implémenter un code correcteur (ex. Reed-Solomon ou BCH) en logiciel (C/C++) puis en matériel (VHDL).
- Intégrer ce code correcteur dans une architecture RISC-V implémentée sur FPGA (ou sur STM32), afin de protéger la mémoire ou les canaux de communication interne.
- Réaliser un démonstrateur fonctionnel sur carte FPGA avec injection contrôlée d'erreurs et mesure du taux d'erreur binaire (BER).

2 Contexte théorique et état de l'art

2.1 Corps finis et probabilités : les bases

Les corps finis, également appelés corps de Galois, sont des structures algébriques finies qui jouent un rôle crucial dans la théorie des codes correcteurs. Un corps fini est noté $\mathbb{F}_{p^n}$, où p est un nombre premier et n un entier positif. Pour les codes binaires, on utilise souvent $\mathbb{F}_{2^m}$. La construction d'un tel corps se fait à l'aide d'un polynôme irréductible de degré m sur $\mathbb{F}_2$. Les éléments du corps sont alors représentés comme des polynômes de degré inférieur à m à coefficients dans $\mathbb{F}_2$, ou encore comme des vecteurs binaires de longueur m. Les opérations arithmétiques (addition, multiplication, inversion) sont définies modulo ce polynôme irréductible.

Parallèlement, la modélisation des canaux de transmission nécessite des bases en probabilités. Un canal sans mémoire est un modèle où les erreurs sur les symboles transmis sont indépendantes. La probabilité d'erreur sur un bit (BER) est un paramètre clé pour évaluer les performances d'un code correcteur.

2.2 Entropie de Shannon et information

La théorie de l'information de Shannon fournit un cadre pour quantifier l'information. L'entropie $H(X)$ d'une variable aléatoire X mesure l'incertitude moyenne associée à ses valeurs. Pour une source discrète avec n symboles de probabilités p_i, l'entropie est définie par :

$$H(X) = \sum_{i=1}^{n} p_i \log_2(\frac{1}{p_i})$$

Dans le contexte de la transmission, on s'intéresse également à l'information mutuelle $I(X;Y)$ entre l'entrée X et la sortie Y du canal, qui mesure la quantité d'information que Y apporte sur X. L'entropie conditionnelle $H(X|Y)$ représente l'incertitude restante sur X après observation de Y. Ces concepts permettent de calculer la capacité d'un canal, c'est-à-dire le débit maximal d'information qui peut être transmis avec une fiabilité arbitrairement élevée.

Dans ce projet, on calculera l'entropie de la source et l'information mutuelle pour évaluer l'efficacité du code correcteur. On comparera la quantité d'information avant et après transmission pour mesurer l'impact du bruit et de la correction.

2.3 Codes correcteurs d'erreurs algébriques

Les codes correcteurs d'erreurs sont des techniques qui ajoutent de la redondance aux données pour permettre la détection et la correction d'erreurs. Les codes linéaires forment une classe importante, où les mots de code sont des combinaisons linéaires de vecteurs de base. Parmi eux, les codes cycliques sont particulièrement intéressants car ils peuvent être implémentés efficacement à l'aide de registres à décalage. Les codes BCH (Bose-Chaudhuri-Hocquenghem) et Reed-Solomon sont des codes cycliques construits sur les corps finis. Ils sont largement utilisés dans les systèmes de communication et de stockage.

Le processus d'encodage consiste à associer à un bloc de données un mot de code en ajoutant des symboles de redondance. Le décodage, plus complexe, comprend la détection d'erreurs et leur correction. Des algorithmes tels que l'algorithme de Berlekamp-Massey

ou l'algorithme d'Euclide étendu sont employés pour décoder les codes BCH et Reed-Solomon. La capacité de correction d'un code dépend de sa distance minimale, qui est le nombre minimal de symboles différents entre deux mots de code distincts.

2.4 Architecture RISC-V et implémentation FPGA

L'architecture RISC-V est une architecture de jeu d'instructions ouverte, qui permet des implémentations variées. Sa simplicité et son extensibilité en font un choix populaire pour la recherche et des applications. Les FPGA sont idéaux pour prototyper des processeurs RISC-V et y intégrer des modules spécifiques, comme un code correcteur d'erreurs.

L'intégration d'un module ECC dans un processeur RISC-V peut se faire via un bus périphérique ou en l'insérant directement dans le chemin de mémoire. Cette intégration permet de protéger les données en mémoire ou lors de transferts entre le processeur et ses périphériques.

3 Méthodologie et plan de travail

Le projet sera divisé en cinq phases principales :

3.1 Phase 1 : Étude théorique et modélisation

Dans cette phase, une revue bibliographique approfondie sera réalisée sur les codes correcteurs et leur implémentation matérielle. On choisira un code spécifique, par exemple un code Reed-Solomon sur $\mathbb{F}_{2^4}$ ou $\mathbb{F}_{2^8}$, et on en modélisera mathématiquement l'encodage et le décodage. On étudiera également les concepts d'entropie et d'information mutuelle pour quantifier l'information transmise. Une simulation logicielle (avec MATLAB ou Python) sera mise en place pour valider le modèle et évaluer les performances théoriques du code.

3.2 Phase 2 : Implémentation logicielle

On développera une bibliothèque de fonctions en C++ pour réaliser les opérations arithmétiques dans le corps fini choisi. Cette bibliothèque inclura l'addition, la multiplication et l'inversion. Ensuite, on implémentera l'encodeur et le décodeur du code correcteur en C++, en intégrant l'algorithme de correction d'erreurs. Des tests unitaires seront réalisés pour valider chaque fonction, et on comparera les résultats avec des références existantes (exemple : AFF3CT).

3.3 Phase 3 : Conception matérielle en VHDL

La conception matérielle commencera par la description en VHDL des opérations arithmétiques dans le corps fini. On concevra ensuite les blocs encodeur et décodeur, en les structurant autour d'une machine à états finis (FSM) pour contrôler le flux de données. Ces blocs seront simulés avec Vivado pour vérifier leur fonctionnalité. On s'assurera que les résultats correspondent à ceux obtenus en logiciel.

3.4 Phase 4 : Intégration dans une architecture RISC-V sur FPGA

On utilisera un cœur RISC-V open-source et on l'implémentera sur une carte FPGA. Le module ECC sera intégré comme périphérique sur le bus (par exemple, par protocole UART) ou directement dans le contrôleur de mémoire. On procédera à la synthèse et à l'implémentation sur le FPGA, puis on validera le système par des tests matériels. On injectera des erreurs de manière contrôlée (via des interruptions logicielles ou une modification matérielle) pour vérifier la capacité de correction.

3.5 Phase 5 : Démonstrateur et analyse de performances

Un firmware simple sera développé pour le RISC-V, utilisant le code correcteur pour protéger des données en mémoire ou lors de communications. On mesurera les performances du système en termes de débit, de latence et de ressources FPGA consommées. On évaluera le taux de correction en fonction du nombre d'erreurs injectées, et on calculera l'entropie et l'information mutuelle pour quantifier l'information effectivement transmise. Enfin, on rédigera un rapport complet et on préparera une démonstration pour la soutenance.

4 Calendrier prévisionnel

— **Semaines 1** : Phase 1 (Étude théorique)
— **Semaines 2-5** : Phase 2 (Implémentation logicielle)
— **Semaines 6-9** : Phase 3 (Conception VHDL)
— **Semaines 9-10** : Phase 4 (Intégration RISC-V/FPGA)
— **Semaines 11** : Phase 5 (Tests, démonstrateur, rédaction)

5 Conclusion

Ce projet thématique représente une opportunité unique de relier des concepts mathématiques avancés à une application concrète en architecture numérique. Il permettra de développer des compétences en théorie des codes, en conception VHDL, en intégration système sur FPGA et en programmation embarquée. L'ajout de l'étude de l'entropie de Shannon offre une perspective supplémentaire pour quantifier l'information et évaluer l'efficacité du code. Le démonstrateur final montrera la faisabilité et l'efficacité de l'approche, ouvrant la voie à des applications dans les systèmes fiables (aérospatial, automobile, communications sécurisées).

Références bibliographiques

— C.Jego, R.Tajan, C.Leroux, A.Cassagne, B.Le Gal *AFF3CT Fast Forward Error Correction Toolbox*.
— R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, 1983.
— S. Lin, D. J. Costello, *Error Control Coding*, Prentice Hall, 2004.
— J.-P. Deschamps, *Hardware Implementation of Finite-Field Arithmetic*, McGraw-Hill, 2009.
— C. E. Shannon, *A Mathematical Theory of Communication*, Bell System Technical Journal, 1948.
— D. Patterson, J. Hennessy, *Computer Organization and Design : The Hardware/-Software Interface*, Morgan Kaufmann, 2017.
— Documentation RISC-V : `https://riscv.org/`