الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي و البحث العلمي

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de RELIZANE
Faculté des Sciences et de la Technologie
Département : Electrotechnique et d'automatique

جامعة أحمد زبانة-غليزان
Ahmed Zabana Relizane University

MEMOIRE

En vue de l'obtention du diplôme de MASTER en :

Automatique et Systèmes

Intitulé

Code de Hsiao (22,16) pour la Protection des Mémoires SRAM Embarquée dans les Microsatellites

Présenté par :
Mr : HAMDAOUI Faouzi
Mr : HIRECHE Seif Eddine Chahine

Devant les membres de jury :

Président : Mr AIT SAID Hakim	Maître de conférences (A)	(U.Relizane)
Encadrant : Mr DAHMANE Ghenam	Maître assistant (A)	(U.Relizane)
Examinateur : Mme RERIBALLAH Hafida	Maître de conférences (B)	(U.Relizane)

Année universitaire : 2024/2025

Remerciements

Cette mémoire a été réalisée grâce à l'aide d'une seule personne, à qui nous exprimons notre gratitude.
Nous tenons à exprimer notre gratitude à notre directeur de thèse, Mr.Dahman Ghanem, pour sa patience et sa coopération, et surtout pour ses conseils avisés qui ont éclairé notre réflexion.
Nous adressons donc nos sincères remerciements au professeur pour ses efforts. Que Dieu lui accorde le succès et guide ses pas.

Dédicaces

Je dédie ce travail à ma famille, pour son amour, son soutien indéfectible et sa patience tout au long de mon parcours. Sans vous, rien de tout cela n'aurait été possible.

À mes camarades diplômés, avec qui j'ai partagé des moments inoubliables, des défis, des réussites et, surtout, une merveilleuse aventure humaine. Merci pour votre soutien mutuel, vos éclats de rire et cette amitié unique.

À mes amis, seif eddine madjber, Boumediene m'hammed, Boumediene karim, abed djafer, mechouar ahmed, bekkadour abdelkader, marouf tayeb, hachemi andelkarim, harir mohamed, miloudi mohamed, merci à tous.

Hamdaoui faouzi

Je dédie ce travail à ma famille, pour son amour, son soutien indéfectible et sa patience tout au long de mon parcours. Sans vous, rien de tout cela n'aurait été possible.

À mes camarades diplômés, avec qui j'ai partagé des moments inoubliables, des défis, des réussites et, surtout, une merveilleuse aventure humaine. Merci pour votre soutien mutuel, vos éclats de rire et cette amitié unique.

À mes amis, harir mohamed, miloudi mohamed, rahal benauda, hachemi abdelkarim, kaddouri abdelouahab, kaddouri oussama, mehdi abdenour, bouchama mohamed, merci à tous.

Hireche seif eddine chahine

Résumé

Ce mémoire traite de la conception de systèmes embarqués fiables pour microsatellites, exposés aux radiations spatiales. Après l'étude des orbites et des sous-systèmes critiques, l'accent est mis sur les erreurs induites par les rayonnements ionisants. Trois codes correcteurs d'erreurs ont été analysés : Hamming (12,8), Hsiao (22,16) et Quasi-cyclique (16,8), implémentés en VHDL et simulés. Le code Hamming se distingue par sa simplicité, le code Hsiao par sa faible consommation énergétique, et le code Quasi-cyclique par sa robustesse, mais au prix d'une consommation accrue de ressources FPGA.

Abstract

This memo deals with the design of reliable embedded systems for microsatellites, exposed to space radiation. After the study of orbits and critical subsystems, the focus is on errors induced by ionizing radiation. Three error-correcting codes were analyzed : Hamming (12,8), Hsiao (22,16) and Quasi-cyclic (16,8), implemented in VHDL and simulated. The Hamming code is distinguished by its simplicity, the Hsiao code by its low energy consumption, and the Quasi-cyclic code by its robustness, but at the cost of increased consumption of FPGA resources.

ملخص

تتناول هذه المذكرة تصميم أنظمة مدمجة موثوقة للأقمار الصناعية الصغيرة المعرضة للإشعاع الفضائي .بعد دراسة المدارات والأنظمة الفرعية الحرجة، ينصب التركيز على الأخطاء الناتجة عن الإشعاع المؤين .تم تحليل ثلاثة رموز لتصحيح الأخطاء : هامينغ(12،8) ، وهسياو(22،16) ، وشبه الدوري(16،8) ، والتي طُبّقت باستخدام لغة VHDL وحاكيتها .يتميز رمز هامينغ ببساطته، ورمز هسياو بانخفاض استهلاكه للطاقة، ورمز شبه الدوري بمتانته، ولكن على حساب زيادة استهلاك موارد.FPGA

Table des matières

Chapitre I : Effet des radiations spatiales sur les circuits intégrés

Chapitre II : Microsatellites d'observation de la Terre

Chapitre III : Théorie des codes correcteurs d'erreurs

Chapitre IV : Conception des circuits EDAC (Error Detection And Correction)

Chapitre V : Simulation des circuits EDAC

Liste des figures

Liste des tableaux

Acronymes

SRAM	Static random-access memory
SEE	Single event effect
SEU	Single event upset
SEL	Single event latchup
SEB	Single event Burnout
SEGR	Single event Gate Rupture
SET	Single event Transient
SEFI	Single Event Functional Interrupt
ECC	Error correcting code
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
MeV	Mega electron-volt
GeV	Giga-electronvolts
keV	Kilo-electronvolts
MOSFET	Metal-oxide semiconductor field-effect transistor
CCD	Charge-coupled device
CMOS	Complementary metal-oxide semiconductor
FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Circuit
SIG	Geographic information system
LEO	Low earth orbit
MEO	Medium earth orbit
SSO	Sun-Synchronous Orbit
GEO	Geostationary earth Orbit
GTO	Geostationary Transfer Orbit
TTC&M	Sous-système de télémétrie et télécommande
SCAO	Système de contrôle d’attitude et d’orbite
OBC	On-Board Computer
CRC	Cyclic redundancy check
LUT	Look up table
SEC	Single error correction
DED	Double error detection

Rem	Remainder (reste)
EDAC	Error Detection And Correction
CPU	Central Proccessing Unit
MBU	Multiple bit Upset
IEEE	Institute of Electrical and Electronics Engineers
CLB	Bloc logique configurable
IOB	Input Output Bloc
RTL	Register Transfer Level
XDC	Xilinx Design Constraints

Introduction générale

Le secteur spatial a connu ces dernières années une véritable explosion technologique, avec un nombre croissant de satellites en orbite. Cette croissance s'accompagne d'une demande croissante de systèmes embarqués fiables et performants. Cependant, l'espace reste un environnement dangereux, notamment en raison des radiations qui peuvent perturber le fonctionnement des composants électroniques.

Ces particules énergétiques, qu'il s'agisse de protons, d'électrons ou d'ions lourds, interagissent directement avec les semi-conducteurs, provoquant parfois des défaillances critiques. La SRAM est particulièrement sensible aux erreurs appelées Single Event Upset (SEU), où un seul choc peut légèrement perturber une cellule mémoire. Ce type d'erreur, même temporaire, peut avoir de graves conséquences sur des systèmes critiques tels que la navigation, les communications ou la gestion à bord. D'autres effets plus graves, comme l'effet Single Event Latch-up (SEL), peuvent provoquer des surintensités susceptibles d'endommager définitivement les circuits. À long terme, une exposition continue aux rayonnements entraîne également une détérioration progressive des composants, tels que les transistors, ou des dommages à la structure du matériau.

Pour faire face à ces risques, l'une des solutions les plus utilisées consiste à intégrer des mécanismes de correction d'erreurs (Error Correcting Code – ECC) dans les systèmes de mémoire. Ces codes permettent de détecter les erreurs causées par les radiations et, dans certains cas, de les corriger automatiquement. Par exemple, les codes de Hamming peuvent corriger une seule erreur, tandis que d'autres, comme les codes Hsiao ou quasi-cycliques, sont plus robustes, mais nécessitent davantage de ressources de calcul. Le code Hsiao(22,16) représente une alternative intéressante, qui est une version améliorée des codes de Hamming. Ce code encode un mot de 16 bits en un mot de 22 bits en ajoutant 6 bits de parité, garantissant ainsi la détection et la correction des erreurs tout en réduisant la complexité matérielle et la surcharge mémoire.

Dans ce projet, le choix s'est porté sur la conception d'une architecture numérique basée sur le code Hsiao afin d'améliorer la fiabilité de la mémoire SRAM utilisée dans les petits satellites. L'objectif est de développer un circuit de détection et de correction des erreurs (Error Detection and Correction – EDAC) pour corriger les erreurs induites par l'effet SEU. L'ensemble du système sera implémenté en VHDL, simulé à l'aide de Vivado, puis testé sur une carte FPGA Xilinx pour évaluer ses performances dans un environnement proche de l'espace. Une comparaison sera ensuite établie entre le circuit EDAC basé sur le code Hsiao et d'autres circuits EDAC traditionnels tels que ceux utilisant le code de Hamming ou le code quasi-cyclique.

Cette mémoire comporte cinq chapitres, le premier chapitre traite de l'effet des radiations spatiales sur les circuits intégrés, tels que les effets SEU et SEL.

Le deuxième chapitre présente les différentes orbites des microsatellites ainsi que les composants matériels embarqués (sous-systèmes).

Le troisième chapitre se concentre sur les algorithmes des codes linéaires utilisés pour la détection et la correction des erreurs, notamment le code de Hamming (12,8), le code Hsiao (22,16) et le code quasi-cyclique (16,8).

Le quatrième chapitre est consacré au développement d'un circuit de détection et de correction des erreurs (EDAC) pour chaque code, avec une explication détaillée de l'architecture d'encodage et de décodage.

Enfin, le cinquième chapitre introduit le langage VHDL, détaille la carte FPGA, décrit le fonctionnement du logiciel Vivado, puis présente les simulations des trois codes et l'analyse des résultats obtenus.

Chapitre I
Effet des radiations spatiales sur les circuits intégrés

I.1 Introduction

Les circuits intégrés intégrés dans les satellites et autres engins spatiaux sont exposés à un environnement de rayonnement intense, qui peut provoquer des perturbations et des dommages irréparables. Dans ce chapitre, nous explorons les sources de rayonnement spatial, leurs effets sur les circuits électroniques et les stratégies d'atténuation.

I.2 Environnement radiatif spatial

Les sources principales de radiations énergétiques présentes dans l'espace sont :

- ✓ Les protons et électrons piégés dans les ceintures de Van Allen,
- ✓ Le rayonnement cosmique (protons et ions lourds),
- ✓ Les protons et ions lourds provenant des éruptions solaires.

1- Espace :

Le domaine spatial impose des exigences de fiabilité extrêmement strictes pour les circuits intégrés en raison de l'environnement radiatif complexe. La diversité des particules et l'impossibilité d'intervenir sur les systèmes en mission rendent la phase de spécification essentielle et délicate. De nombreux satellites ont subi des dysfonctionnements causés par les radiations, comme l'indiquent plusieurs études. Cet environnement radiatif spatial est particulièrement complexe, car il résulte de l'interaction de trois composantes avec la magnétosphère terrestre.**[1]**

a) Le vent solaire

Le vent solaire est un flux de particules chargées (principalement des protons, des électrons et des ions lourds) émis en continu par le soleil, dans l'espace ces particules peuvent avoir des effets néfastes sur les systèmes électroniques embarqués dans les satellites et les engins spatiaux. La figure I.1 illustre le vent solaire.

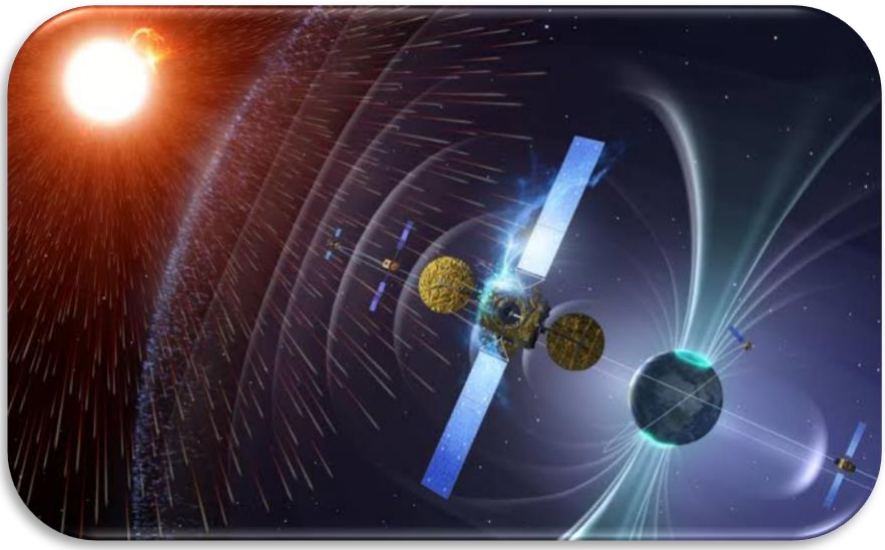


Figure I.1 : Le vent solaire

b) Les ceintures de Van Allen

Les ceintures de Van Allen (Figure I.2) sont des zones de radiation intense entourant la Terre, formées par des particules chargées piégées par le champ magnétique terrestre. Elles ont un impact significatif sur les circuits intégrés embarqués dans les satellites et engins spatiaux qui traversent ou opèrent dans ces régions.
Les ceintures de Van Allen se divisent en deux principales couches :

- **Ceinture interne (1 000 à 12 000 km d'altitude)**

Contient principalement des protons énergétiques (10-100 MeV).
Zone de radiation la plus intense, affectant fortement les composants électroniques.

- **Ceinture externe (13 000 à 60 000 km d'altitude)**

Contient principalement des électrons énergétiques (0,1-10 MeV).
Ces électrons peuvent induire des charges électriques dangereuses dans les matériaux isolants des satellites.**[1]**

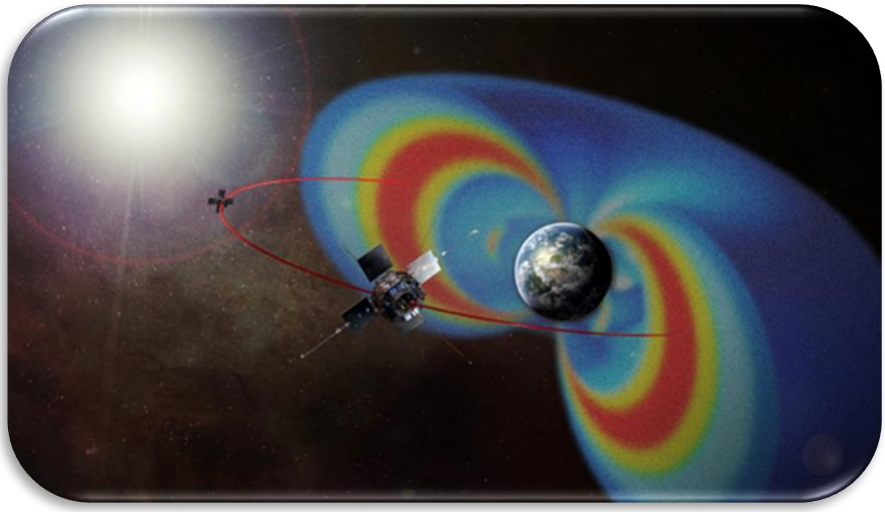


Figure I.2 : Les ceintures de Van Allen

c) Les rayons cosmiques galactiques

Les rayons cosmiques (Figure I.3) sont des particules chargées extrêmement énergétiques provenant de l'extérieur du système solaire. Ils sont principalement composés de protons, de particules alpha et d'éléments dont le numéro atomique varie de 1 à 92. Bien que leur flux soit faible, leur énergie, qui peut atteindre plusieurs centaines de GeV, leur permet de traverser des matériaux, y compris les blindages et même les satellites. Malgré leur faible nombre, ils représentent une menace pour les composants électroniques en raison des Single Event Effects (SEEs) qu'ils peuvent engendrer.**[2]**

Tout comme les protons piégés, l'intensité des rayons cosmiques fluctue en fonction du cycle solaire. Elle est maximale lors du minimum solaire et diminue lors du maximum solaire. Le champ magnétique terrestre offre une certaine protection aux engins spatiaux selon leur altitude et leur inclinaison orbitale. Cependant, dans les régions polaires, où les lignes du champ magnétique sont ouvertes, ces particules ont un accès libre et peuvent interagir directement avec les systèmes électroniques embarqués.

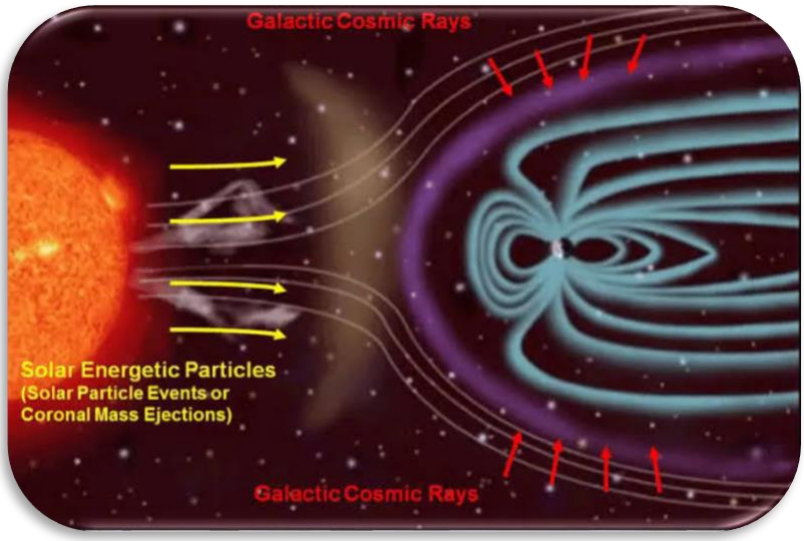


Figure I.3 : Les rayons cosmiques galactiques

I.3 La sensibilité des circuits intégrés face aux différents effets de radiations

Les radiations spatiales peuvent provoquer plusieurs types de perturbations dans les circuits intégrés. Ces perturbations peuvent être temporaires ou permanentes, selon l'intensité et la nature de l'interaction entre les particules énergétiques et le semi-conducteur.

Les effets d'événement unique (SEE) sont causés par une seule particule énergétique et peuvent prendre de nombreuses formes. Les perturbations d'événement unique (SEU) sont des erreurs douces et non destructives. Elles apparaissent généralement sous forme d'impulsions transitoires dans les circuits logiques ou de support, ou sous forme de basculements de bits dans les cellules de mémoire ou les registres. Plusieurs types d'erreurs matérielles, potentiellement destructrices, peuvent apparaître : le verrouillage d'événement unique (SEL) entraîne un courant de fonctionnement élevé, supérieur aux spécifications de l'appareil, et doit être effacé par une réinitialisation de l'alimentation. D'autres erreurs matérielles incluent la surchauffe des MOSFET de puissance, la rupture de grille, les bits gelés et le bruit dans les CCD.

I.3.1 Single Event Effect (SEE)

Les effets d'événements uniques (Single Event Effects, SEE) se produisent lorsqu'une seule particule énergétique interagit avec un circuit intégré et provoque un dysfonctionnement.(Figure I.4)

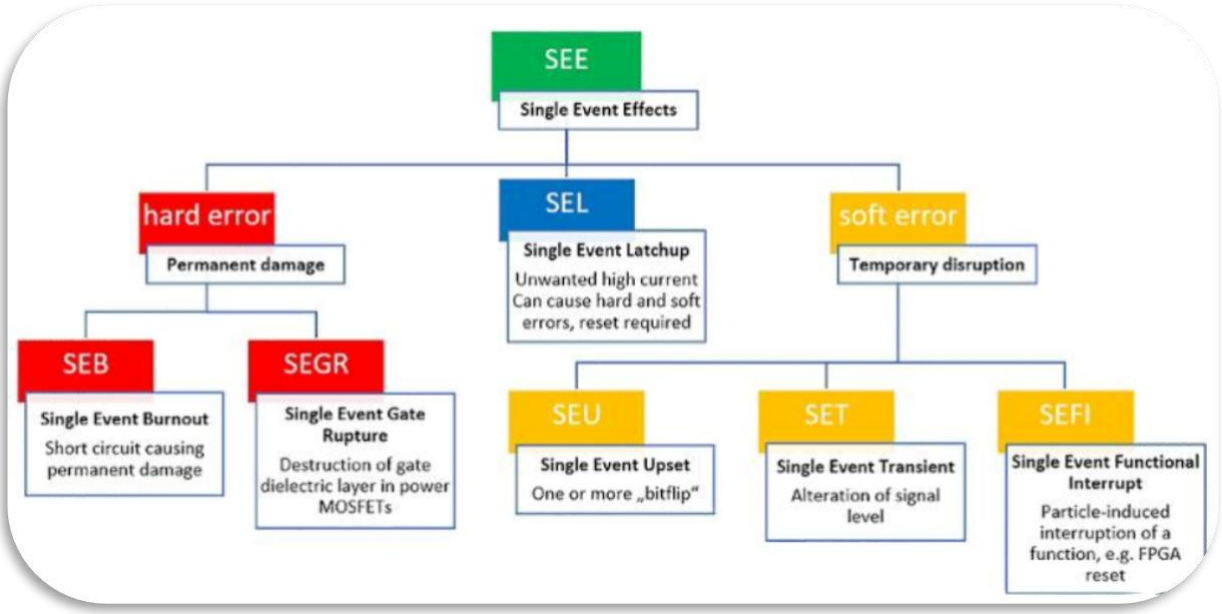


Figure I.4 : schéma des effets d'événement uniques**.[3]**

Ce schéma présente une classification des effets d'événements uniques (SEE – Single Event Effects), qui sont des perturbations causées par des particules énergétiques frappant un circuit électronique. Ces effets peuvent être temporaires (erreurs molles) ou permanents (erreurs dures) et affectent principalement les systèmes électroniques embarqués, comme ceux utilisés dans les satellites et les missions spatiales.

1- Erreurs dures :

Ces erreurs provoquent des dégradations irréversibles du circuit, pouvant entraîner une panne définitive, comme :

- **Single Event Burnout (SEB)**

Se produit dans les composants de puissance (MOSFETs, transistors bipolaires).
Une particule énergétique induit un court-circuit, causant une surchauffe et la destruction du composant.

- **Single Event Gate Rupture (SEGR)**

Affecte principalement les MOSFETs de puissance.

Une particule ionisante peut provoquer la rupture de la couche diélectrique de la grille, rendant le transistor inutilisable.

2- Erreurs molles :

Ces erreurs n'endommagent pas physiquement le circuit, mais perturbent temporairement son fonctionnement, comme :

- **Single Event Upset (SEU)**

Une particule modifie l'état d'un bit dans une mémoire ou un registre ($0 \rightarrow 1$ ou $1 \rightarrow 0$).

- **Single Event Transient (SET)**

Se produit dans les circuits analogiques et numériques.
Une particule altère brièvement un signal, provoquant des glitches dans les circuits logiques, Peut affecter le timing et les signaux de contrôle.

- **Single Event Functional Interrupt (SEFI)**

Affecte des circuits complexes comme les FPGA, microprocesseurs et ASIC, et Peut induire un redémarrage forcé (reset) ou une panne temporaire.

Souvent causé par une perturbation dans les circuits de contrôle internes.

3- Single Event Latchup (SEL)

Se produit dans les circuits CMOS lorsqu'une particule déclenche un chemin conducteur indésirable.

Peut provoquer une consommation excessive de courant, risquant d'endommager le circuit.

Peut être récupéré par un reset, mais peut aussi causer une panne permanente s'il n'est pas contrôlé.**[4]**

I.3.2 Single-Event Upset (SEU)

Single event upset (SEU) se produit lorsqu'une particule énergétique modifie l'état logique d'un bit dans une mémoire ou un registre,il génère des charges électriques qui peuvent changer l'état logique d'un bit **($0 \rightarrow 1$ ou $1 \rightarrow 0$)**. sans causer de dommage physique au circuit.

- **Mécanisme du SEU :**

Le phénomène de Single Event Upset (SEU) représente une erreur transitoire causée par l'interaction d'une particule ionisante (comme un proton ou un ion lourd, souvent présent dans l'environnement spatial) avec un circuit intégré, typiquement dans une cellule mémoire ou un bascule. Dans le schéma présenté (figure 1.4), la cellule est composée de deux inverseurs croisés (P1-N1 et P2-N2), formant un bistable capable de maintenir un état logique stable. Lorsqu'une particule frappe la région du drain du transistor N2, qui est à l'état bloqué (off), elle dépose de l'énergie sous forme d'ionisation dans le substrat de silicium. Cette ionisation génère une grande quantité de paires électron-trou, dont certaines sont collectées par la jonction PN du drain. Si la charge collectée dépasse un certain seuil critique, elle provoque une chute brutale de tension sur le nœud QB, initialement à l'état haut. Ce changement active N1 (dont la grille est reliée à QB), ce qui entraine un basculement de l'état logique de la cellule : la sortie Q passe de 0 à 1, et QB de 1 à 0. Ainsi, l'information stockée est inversée, sans action du circuit logique environnant, uniquement à cause de l'interaction avec une particule. Ce type d'erreur est temporaire mais critique dans les systèmes embarqués ou spatiaux, où l'environnement radiatif est intense, car il peut entraîner des corruptions de données ou des comportements imprévus du système.**[5]**

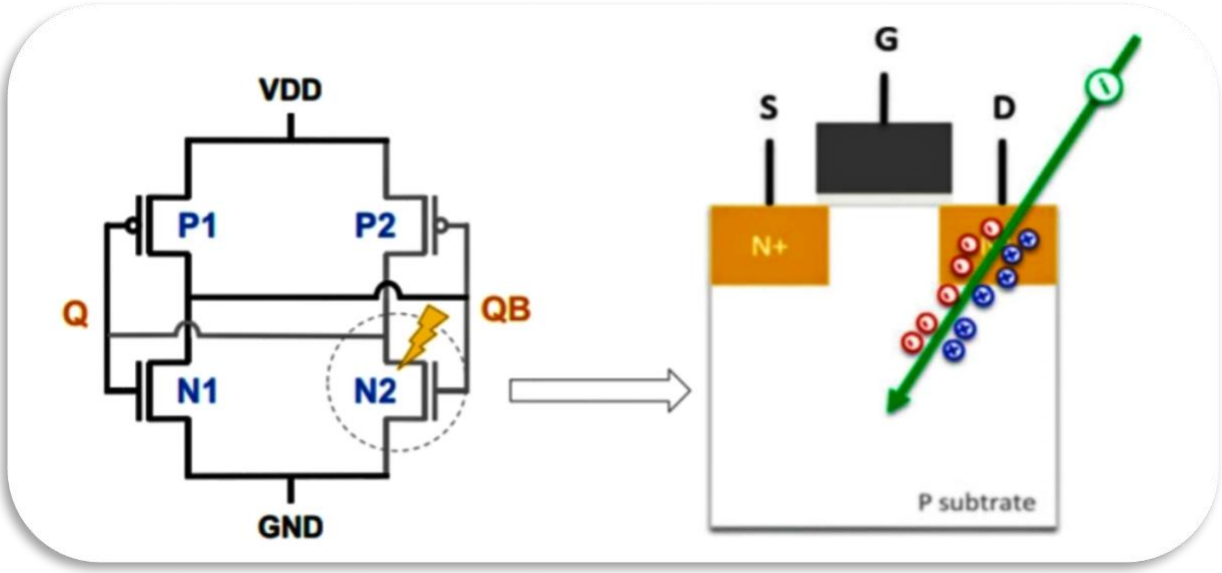


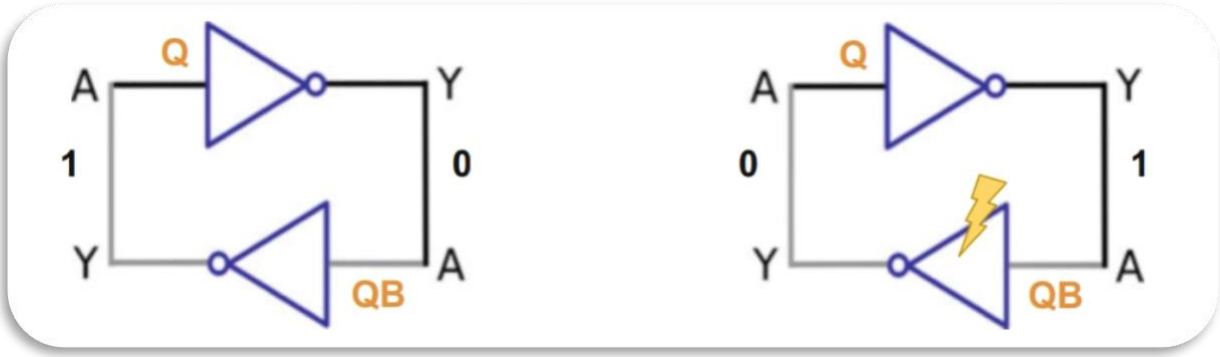
Figure I.5 : Représentation au niveau transistoriel d'un Single Event Upset (SEU).**[5]**

- **Conséquences du SEU :**

Altération des données stockées dans la mémoire.

Dysfonctionnements logiciels si les instructions exécutées sont modifiées.

Risques critiques dans les systèmes embarqués nécessitant une fiabilité élevée (ex. contrôle de vol spatial).



État 1 (stable) **État 2** (perturbation)

Figure I.6 : représentation du SEU (Single-Event Upset) dans un onduleur à couplage croisé Paire.**[5]**

I.3.3 Single-Event Latchup (SEL)

Single event latchup (SEL) est un phénomène dangereux qui peut entraîner une panne définitive d'un circuit intégré. Il se produit principalement dans les circuits CMOS lorsqu'une particule ionisée déclenche un court-circuit entre l'alimentation et la masse.

- **Mécanisme du SEL :**

Dans les technologies CMOS, la proximité des transistors NMOS et PMOS crée involontairement une structure parasite de type NPNP (Figure I.7), semblable à un thyristor. Si cette structure entre en conduction, elle provoque une amplification incontrôlée du courant, avec un gain théoriquement infini, ce qui engendre un court-circuit entre les rails d'alimentation. Ce phénomène, appelé verrouillage (ou latch-up), peut entraîner la destruction du circuit s'il n'est pas immédiatement interrompu en coupant l'alimentation.**[4]**

- **Méthodes de prévention du SEL :**

La méthode de durcissement la plus couramment employée contre les effets du SEL (Single Event Latch-up) consiste à surveiller le courant d'alimentation du circuit et à couper automatiquement l'alimentation dès que ce courant dépasse un seuil prédéfini. Cette approche entraîne une réinitialisation du circuit. Toutefois, certaines technologies, telles que le SOI (Silicon On Insulator) ou le DNW (Deep N-Well).

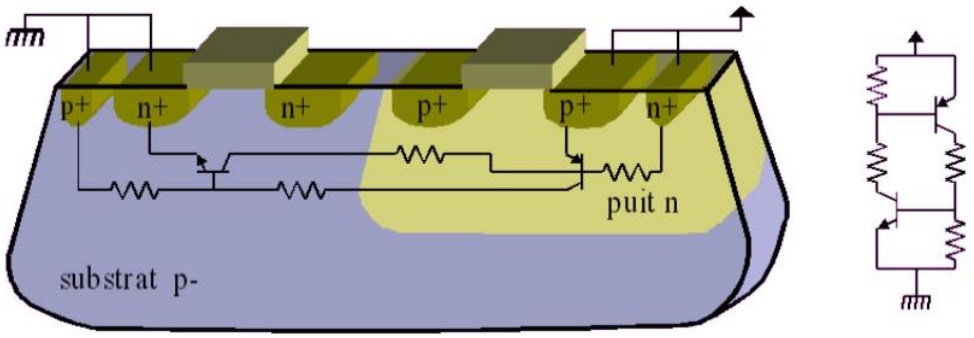


Figure I.7 : Mécanisme de single event latchup (SEL)[**4]**

I.4 Conclusion

Dans ce premier chapitre, nous avons étudié l'environnement radiatif spatial, ses effets sur les circuits électroniques embarqués et la sensibilité des composants aux radiations. L'espace est un milieu hostile où les particules énergétiques, issues principalement du vent solaire, des rayons cosmiques galactiques et des ceintures de Van Allen, peuvent provoquer des dégradations sur les circuits intégrés des satellites. Ces particules interagissent avec les composants électroniques, entraînant des effets transitoires et permanents tels que les Single Event Effects (SEE), les Single Event Upsets (SEU), les Single Event Latchup (SEL).

En conclusion, la gestion des effets radiatifs est un défi majeur dans la conception des microsatellites, car la performance et la durée de vie des systèmes électroniques en dépendent fortement. Comprendre ces phénomènes et intégrer des solutions adaptées dès la phase de conception est essentiel pour assurer le bon fonctionnement et la réussite des missions spatiales, notamment en orbite basse où l'exposition aux radiations reste significative. L'évolution des technologies et des matériaux permettra de mieux protéger les systèmes embarqués et d'améliorer la robustesse des satellites face aux environnements radiatifs de l'espace.

Chapitre II
Microsatellites d'observation de la Terre

II.1 Introduction

Les microsatellites, pesant entre 10 et 100 kg, représentent une alternative compacte et économique aux satellites traditionnels. Leur développement a été rendu possible grâce à la miniaturisation des composants, à l'optimisation énergétique et à la baisse des coûts. Flexibles et modulaires, ils s'adaptent à diverses missions comme l'observation de la Terre, la météorologie ou les télécommunications. Déployés en constellation, ils offrent une couverture étendue et des données en quasi-temps réel. Leur durée de vie varie de 5 à 10 ans, mais leur conception doit faire face aux défis de l'environnement spatial. L'essor des microsatellites a permis à de nouveaux acteurs, comme des start-ups et universités, de participer à l'aventure spatiale, marquant une révolution dans le secteur grâce à leur accessibilité et leur performance.**[6]**

II.2 Applications des satellites d'observation de la Terre

Observer la Terre, c'est l'analyser, la surveiller et chercher à la comprendre dans toutes ses dimensions. Cette observation se structure autour de quatre missions principales.

- ✓ **Voir :** Observer un phénomène ou un objet localisé géographiquement (latitude/longitude) sur une image récente de la Terre, comme la construction d'un bâtiment ou l'étendue d'une inondation.

- ✓ **Comparer** : Suivre l'évolution d'un lieu à travers des images prises à différentes dates, pour analyser des changements comme l'urbanisation ou la déforestation.

- ✓ **Détecter** : Identifier rapidement des modifications naturelles ou humaines sur de vastes territoires, telles que des catastrophes, des pollutions ou des activités illégales.

- ✓ **Mesurer / Extraire** : Obtenir et analyser des données géographiques (formes, surfaces, volumes…) sur des éléments naturels ou artificiels, grâce à des outils de traitement d'images et des logiciels SIG.**[7]**

II.3 Différentes orbites des microsatellites

Le figure (II.1) représente les différents orbites des microsatellites :

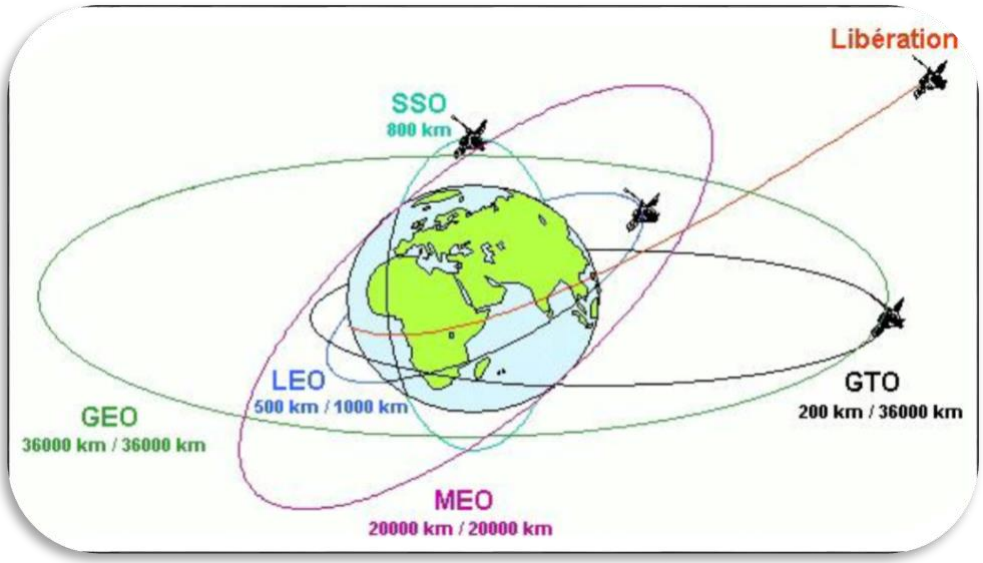


Figure II.1 : Les différents orbites des microsatellites

1. Satellites en orbite terrestre basse

Les orbites terrestres basses (LEO) sont généralement circulaires ou légèrement elliptiques, situées à moins de 2 000 km d'altitude. Les satellites en LEO complètent une orbite en 90 à

120 minutes, avec une couverture instantanée de 3 000 à 4 000 km. Ils restent visibles depuis un point au sol pendant un maximum de 20 minutes.

2. L'orbite terrestre moyenne

Les orbites terrestres moyennes (MEO), également appelées orbites circulaires intermédiaires (ICO), sont des orbites circulaires situées à une altitude d'environ 10 000 km, correspondant à une période orbitale d'environ 6 heures.

3. L'orbite SSO (Sun-Synchronous Orbit)

L'orbite SSO est une orbite polaire à environ 800 km d'altitude. Un satellite en SSO passe toujours au-dessus d'un même point de la Terre à la même heure locale, ce qui est très utile pour les satellites d'observation et les missions météorologiques nécessitant des conditions d'éclairage constantes.

4. L'orbite géostationnaire

Abrégée GEO (geostationary orbit) est un cas particulier de l'orbite géosynchrone avec une inclinaison nulle par rapport au plan équatorial. L'altitude de l'orbite géostationnaire est 35 786 km au-dessus du géoïde terrestre ; on parle couramment de satellites à 36 000 km.**[8]**

5. L'orbite GTO (Geostationary Transfer Orbit)

C'est une orbite de transfert qui permet aux satellites de passer d'une orbite basse à une orbite géostationnaire. Elle s'étend de 200 km à 36 000 km et sert de trajectoire intermédiaire pour les satellites avant d'atteindre leur position finale en GEO.

6. Trajectoire de libération

La trajectoire de libération représente une trajectoire qui permet à un satellite ou une sonde de quitter définitivement l'attraction gravitationnelle terrestre et de se diriger vers l'espace interplanétaire, généralement pour des missions d'exploration vers la Lune, Mars ou d'autres destinations du système solaire.

II.4 Composants matériels de microsatellites

Un microsatellite est composé de plusieurs sous-systèmes essentiels qui assurent son bon fonctionnement en orbite. Chaque sous-système joue un rôle spécifique pour garantir la réussite de la mission.

II.4.1 Sous-système de propulsion

Le sous-système de propulsion est l'ensemble des dispositifs d'un véhicule spatial destinés à générer une poussée, permettant de modifier sa vitesse, sa trajectoire ou son orientation. Il fonctionne en libérant de l'énergie, il existe différents types de propulsion, eux-mêmes divisés en sous-groupes :

- **La propulsion chimique** dans laquelle un gaz à haute est libéré avec une importante quantité de mouvement,

- **La propulsion électrique** qui utilise une énergie électrique ou électromagnétique afin d'augmenter la quantité de mouvement du carburant expulsé.
- **La propulsion « sans-carburant »** qui, comme son nom l'indique, ne nécessite pas de carburant.**[9]**

II.4.2 Sous-système de contrôle thermique

Le contrôle thermique d'un satellite (Figure II.2) maintient ses composants dans des plages de température adaptées au fonctionnement et à la survie, limite les variations thermiques et protège l'intégrité structurelle. Il gère les apports et pertes de chaleur dus à la dissipation interne, au rayonnement solaire, à l'albédo, à l'infrarouge terrestre et au fond spatial.**[9]**

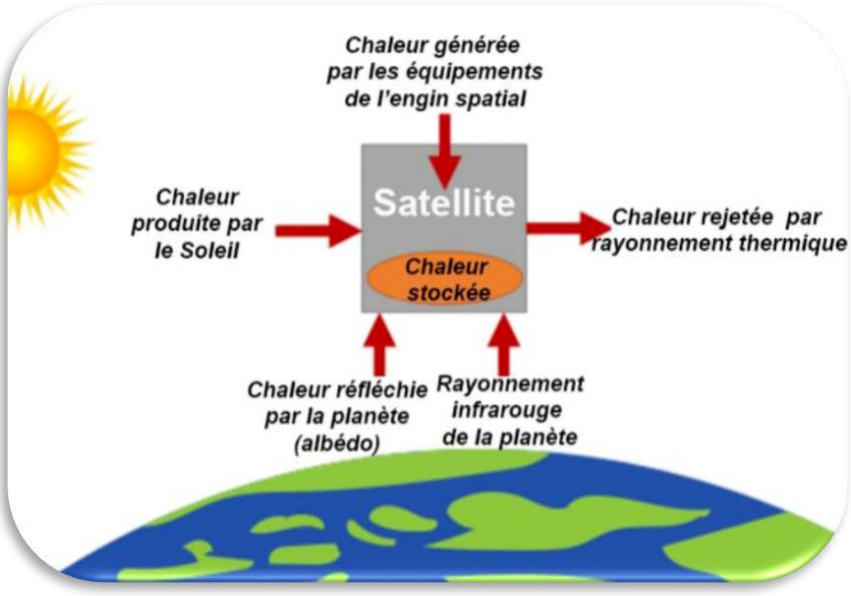


Figure II.2 : Schéma simplifié des flux de chaleur externes d'un satellite en orbite autour d'une planète

II.4.3 Sous-système d'alimentation électrique

La génération d'énergie est essentielle au fonctionnement d'un engin spatial, qui doit produire lui-même l'électricité nécessaire à ses systèmes. Le système d'alimentation se compose de quatre éléments : la production, le stockage, la régulation et la distribution.

La production repose principalement sur l'énergie solaire, captée par des panneaux (souvent déployables) fixés sur la structure du satellite.**[9]**

II.4.4 Sous-système de télémétrie et télécommande

Le sous-système TTC&M (Figure II.3) est essentiel pour assurer le bon fonctionnement et la sécurité d'un satellite en orbite. Il regroupe les fonctions suivantes :

- **Suivi :** Permet de déterminer la position, l'orbite et le mouvement du satellite via des signaux de balise reçus par les stations au sol.
- **Télémétrie :** Transmet au sol les données collectées par les capteurs embarqués (tension, courant, température, pression, état des relais, etc.).

- **Commande :** Reçoit les instructions depuis le sol pour contrôler divers sous-systèmes du satellite (orientation des antennes, gestion de l'alimentation, modes de fonctionnement, etc.).

Le système embarqué comprend des antennes, un récepteur de commande, un émetteur de télémétrie et parfois des capteurs de suivi. Au sol, il inclut une antenne, un récepteur de télémétrie, un émetteur de commande, un système de suivi, et les unités de traitement des données.

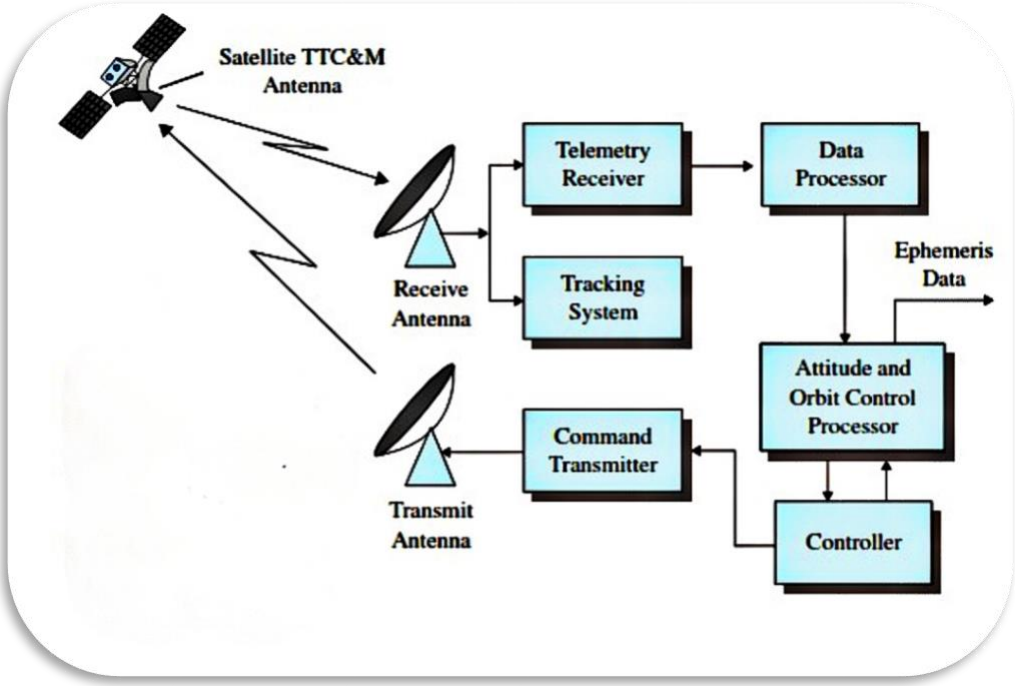


Figure II.3 : Schéma du sous-système TTC&M d'un satellite**[10].**

II.4.5 Sous-système de contrôle d'attitude

Le système de contrôle d'attitude et d'orbite, ou SCAO, joue un rôle central dans le bon fonctionnement d'un satellite. Il s'occupe à la fois de son orientation – c'est-à-dire la façon dont il se tourne dans l'espace – et de sa position sur l'orbite. En fonction des objectifs de la mission, le SCAO ajuste l'attitude du satellite pour qu'il pointe dans la bonne direction. Quant à sa trajectoire, elle doit rester très précise, et c'est le sous-système de propulsion qui permet de faire les petits ajustements nécessaires. Ces manœuvres peuvent être planifiées depuis la Terre, mais de plus en plus souvent, elles sont calculées directement à bord grâce à des navigateurs embarqués. Cela rend les satellites plus autonomes et réactifs.**[10]**

II.4.6 Sous-système de la charge utile

La charge utile d'un satellite est l'ensemble des équipements qui remplissent sa mission principale, qu'il s'agisse d'observation de la Terre, de télécommunications, de navigation ou de recherche scientifique. Elle est composée de capteurs, d'instruments scientifiques, d'antennes de communication, de transpondeurs et de systèmes informatiques spécialisés.

II.4.7 Sous-système OBC (On-Board Computer)

Le figure II.4 présente l'ordinateur de bord (OBC) que joue un rôle central dans la gestion du satellite. Il traite les commandes envoyées depuis le sol, supervise le fonctionnement de tous les sous-systèmes, et assure la collecte ainsi que la transmission des données. Ses responsabilités incluent notamment le déploiement initial des équipements, la gestion de

l'alimentation des charges utiles, la synchronisation temporelle du système, la commande conditionnelle des modules, la récupération après panne, ainsi que le traitement des télécommandes et télémétries.**[11]**

Figure II.4 : les Cartes d'extensions du l'obc des microsatellites

II.5 Conclusion

Dans ce chapitre, nous explorons les applications, les types d'orbites et les sous-systèmes des petits satellites, soulignant leur importance pour le succès des missions spatiales. Aujourd'hui, les petits satellites jouent un rôle clé dans de nombreux domaines, notamment l'observation de la Terre, les communications, la navigation, la recherche scientifique et les applications militaires. Leur flexibilité et leur faible coût les rendent particulièrement attractifs pour des tâches spécifiques qui nécessitent des technologies avancées et une mise en œuvre rapide.

Le choix de l'orbite est un facteur critique dans la conception et les performances des satellites. Selon l'objectif de la mission, un petit satellite peut être placé en orbite terrestre basse (LEO) pour une observation détaillée de la Terre, en orbite terrestre moyenne (MEO) pour la navigation ou en orbite géostationnaire (GEO) pour des services de communication continus. Chaque type d'orbite présente des avantages et des défis, qui ont un impact direct sur la conception des sous-systèmes et la durée de vie du satellite.

Chapitre III
Théorie des codes correcteurs d’erreurs

III.1 Introduction

Dans les systèmes de communication numériques et le stockage de données, la fiabilité de l'information est souvent mise à l'épreuve par des perturbations telles que le bruit, les interférences électromagnétiques, ou encore les effets des radiations dans les environnements spatiaux. Ces perturbations peuvent entraîner des erreurs dans les données transmises ou enregistrées. Les théories des codes correcteurs d'erreurs (ECC – Error Correction Codes) constituent un ensemble de méthodes mathématiques destinées à détecter et corriger ces erreurs, assurant ainsi l'intégrité des informations.

Le principe fondamental repose sur l'ajout de redondance à l'information initiale, sous forme de bits supplémentaires appelés bits de contrôle ou bits de parité. Cette redondance permet non seulement de détecter qu'une erreur s'est produite, mais aussi, dans de nombreux cas, d'identifier sa position et de la corriger automatiquement, sans nécessiter de retransmission.

Les premiers travaux dans ce domaine remontent à la théorie de l'information de Claude Shannon (1948), qui a posé les bases théoriques du codage de l'information. Depuis, de nombreux types de codes ont été développés, chacun présentant un compromis entre capacité de correction, complexité de mise en œuvre, et redondance introduite. Parmi les plus connus, on retrouve les codes de Hamming et hsiao, les codes cycliques (comme le CRC), les codes BCH, et les codes Reed-Solomon.

III.2 Codes linéaires

Un code linéaire ***C(n,k)*** est un code correcteur d'erreurs dans lequel chaque mot de code est un vecteur appartenant à un sous-espace vectoriel défini sur un corps fini, généralement le corps binaire F^n_2. Cela signifie que :

- La somme de deux mots valides est toujours un mot valide (propriété de linéarité).
- Le code peut être défini à partir d'une matrice génératrice , qui transforme les données d'entrée en mots codés.

Un code linéaire est noté par les paramètres ***[n,k]*** , où :

- ✓ **k** Est le nombre de bits d'information,
- ✓ **n** Est le nombre total de bits du mot codé,
- ✓ Il ajoute donc $p = n - k$ bits de redondance pour détecter et corriger les erreurs.

Si le code a une distance minimale d_{min} , on peut le noter $C(n,k,d_{min})$.
Un code $C(n,k,d_{min})$ permet :

- De détecter jusqu'à $d_{min} - 1$ erreurs,
- Et de corriger jusqu'à $t = \left[\frac{dmin-1}{2}\right]$erreurs.

La distance minimale d_{min} est le plus petit poids trouvé parmi ces mots de codes.**[12]**

- **Example :**

C = 1001 (poids 2)
C = 1011 (poids 3)
C = 1111 (poids 4)

Donc tout les mot non nuls ont un poids de 2 => $\mathbf{d_{min} = 2}$

III.2.1 Les codes de hamming

Les codes de Hamming sont une classe de codes correcteurs d'erreurs linéaires permettant la détection et la correction d'erreurs sur un seul bit. Introduits par Richard Hamming en 1950, ils restent largement utilisés dans les communications numériques et les systèmes de mémoire où l'intégrité des données est essentielle. Leur principe repose sur l'insertion de bits de parité soigneusement calculés dans le flux de données afin de détecter et de localiser les erreurs de transmission.**[13]**

III.2.1.1 Hamming (12.8)

1. Principe de codage

on fixe un entier k et on code chaque bloc de bits de données par un Bloc de ***n*** bits en ajoutant donc ***p*** bits, bits de correction, a certaines positions au bloc de ***k*** bits.

Le tableau suivant indique les nombres de bits de correction ***p***, pour différentes valeurs des données ***k***.

p=3	*k=4*	*n=7*
p=4	*k=8*	*n=12*
p=4	*k=11*	*n=15*

Tableau III.1 : Les bits de données *k* et leurs bits de parité *p.*

Position des k bits de correction

Les k bits de correction sont places dans le bloc envoyé aux positions d'indice une puissance de 2 (2^0,2^1,2^2,…,2^n) en comptant à partir de la gauche. Ainsi, en notant *p0 p1 p2 p3* les bits de Correction et *d0 d1 d2 d3 d4 d5 d6 d7* les bits de données, le bloc envoyé est :

$$C = \{ p_3\, p_2\, p_1\, p_0\, d_7\, d_6\, d_5\, d_4\, d_3\, d_2\, d_1\, d_0 \} \qquad \text{(Eq.1)}$$

Calcul des k bits des parités

les bits de correction sont calcules en utilisant XOR entre les bits de données ***k*,** en utilisant l'algorithme suivent :

***P0* :** Vérifié un bit et saute un bit.
***P1* :** Vérifié deux bits et saute deux bits.
***P2* :** Vérifié trois bits et saute trois bits.
***P3* :** Vérifié quatre bits et saute quatre bits.**[13]**

Donc les équations de parité donne par :

$p_0 = d_0 \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_6$
$p_1 = d_0 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6$
$p_2 = d_1 \oplus d_2 \oplus d_3 \oplus d_7$
$p_3 = d_4 \oplus d_5 \oplus d_6 \oplus d_7$

Ça donne :

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

P est une matrice de parité dont les lignes sont des bits de données et dont les colonnes sont des bits de contrôle.

Remarque : les opérations d'addition sont faites à XOR. Par exemple :

$$1+0=1 \ ; \ 1+1=0 \ ; \ 1+1+1=1 \text{ etc.} \qquad \text{(Eq.2)}$$

Matrice de génératrice G

Dans un code linéaire, la matrice génératrice permet de coder un message (vecteur de données) en un mot codé (codeword) que l'on peut transmettre, l'opération de codage est :

$$C = k[P \quad I_k] = [kP \quad k] \qquad \text{(Eq.3)}$$

Mais pour construire la matrice génératrice, on utilise la forme systématique :

$$G = (P^T \quad I_k) \qquad \text{(Eq.4)}$$

- I_k est la matrice identité $k \times k$.
- P est une matrice $k \times r$ et $r = n - k$ qui encode les relations de parité.

Alors la matrice **G** devient **:**

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

2. Principe de décodage

Il est à noter que l'opération de décodage consiste à la correction des éventuelles erreurs survenues lors de la transmission d'un mot code.

➢ Décodage par syndrome

Soit ***r*** un vecteur de longueur ***n*** sur F_2 , et soit ***H*** la matrice de contrôle pour le code ***C*** . Le vecteur :

$$s = rH^T \quad \text{(Eq.5)}$$

r est le mot code recû , dans ce cas :

$$r = \{ p_3 ` p_2 ` p_1 ` p_0 ` d_7 ` d_6 ` d_5 ` d_4 ` d_3 ` d_2 ` d_1 ` d_0 ` \} \quad \text{(Eq.6)}$$

Le syndrome permet de détecter si une erreur s'est produite pendant la transmission.**[14]**

- Si $s = 0$, alors ***r*** est un mot code valide (aucune erreur détectée).
- Si $s \neq 0$, cela signifie qu'une erreur s'est produite.

On suppose qu'un mot code ***c*** dans un code en bloc linéaire ***C*** est transmis et que le mot reçu ***r*** est le vecteur reçu. On peut écrire :

$$r = c + e \quad \text{(Eq.7)}$$

Où ***e*** est le vecteur d'erreur , contient la position d'erreur.

En fait, le syndrome est uniquement lié au vecteur d'erreur ***e*** , car :

$$s = rH^T = (c + e)H^T = eH^T \quad \text{(Eq.8)}$$

On mettons $cH^T = 0$ car le ***c*** est un mot code correct.

H La matrice de contrôle, notée le plus souvent H, comporte (n−k) lignes et n colonnes, permet de calculant le syndrome de mot reçu ***r*** Elle S'écrit à l'aide de la matrice de parité P :

$$H = (I_{n-k} \quad P) \quad \text{(Eq.9)}$$

Ca donne :

$$\boldsymbol{H} = \begin{bmatrix} 1\,0\,0\,0 & 1\,1\,1\,1\,0\,0\,0\,0 \\ 0\,1\,0\,0 & 1\,0\,0\,0\,1\,1\,1\,0 \\ 0\,0\,1\,0 & 0\,1\,1\,0\,1\,1\,0\,1 \\ 0\,0\,0\,1 & 0\,1\,0\,1\,1\,0\,1\,1 \end{bmatrix}$$

Une fois trouvée l'erreur $\boldsymbol{e}$, on retrouve le mot de code $\boldsymbol{c} = \boldsymbol{e} + \boldsymbol{r}$ corrigé.

$S_0 = P_0` \oplus d_0` \oplus d_1` \oplus d_3` \oplus d_4` \oplus d_6`$
$S_1 = P_1` \oplus d_0` \oplus d_2` \oplus d_3` \oplus d_5` \oplus d_6`$
$S_2 = P_2` \oplus d_1` \oplus d_2` \oplus d_3` \oplus d_7`$
$S_3 = P_3` \oplus d_4` \oplus d_5` \oplus d_6` \oplus d_7`$

Nous trouvons le même résultat que (Eq.5).

- **Table de décodage par syndrome**

Le syndrome dépend uniquement du vecteur d'erreur , que l'on choisit avec un poids minimal.
Ainsi, il suffit de stocker dans une table les syndromes et leurs vecteurs d'erreur associés.
Cette table est appelée table de décodage par syndrome (Look-Up Table).

Alors, le décodage se fait à une 3 étape :

1. Calculer le syndrome pour $\boldsymbol{s} = \boldsymbol{r}\boldsymbol{H}^{\boldsymbol{T}}$ avec $\boldsymbol{r}$ est le vecteur reçu.

2. Recherche le vecteur erreur $\boldsymbol{e}$ correspondant à $\boldsymbol{s}$ dans le table de décodage.

3. Calculer le mot code $\boldsymbol{c} = \boldsymbol{r} + \boldsymbol{e}$ corrigé.

BITS DE SYNDROME	LOOK UP TABLE
0000	*000000000000*
1000	*100000000000*
0100	*010000000000*
0010	*001000000000*
0001	*000100000000*
1100	*000010000000*
1011	*000001000000*
1010	*000000100000*
1001	*000000010000*
0111	*000000001000*
0110	*000000000100*
0101	*000000000010*
0011	*000000000001*

Tableau III.2 : Look up table pour le code hamming (12.8)

- ***Remarque :*** le code hamming (12.8) DED-SEC , Il détecte deux erreurs mais elles ne peuvent pas être corrigées car :

Example : Nous mettons deux erreurs dans le 1ère et 2ème positions et calculons le syndrome d'erreur, donc :

$\boldsymbol{e} = 000000000011 \rightarrow \boldsymbol{s} = \boldsymbol{e}.\boldsymbol{H}^{T}$

Donc $\boldsymbol{s} = 0110$, mais cette syndrome correspondant à le 3ème bit dans le tableau , Cela empêche de corriger deux erreurs car l'emplacement exact de l'erreur ne peut pas être connu.

III.2.2 Les codes cycliques

Soit $\boldsymbol{C}$ l'ensemble des mots de code d'un code ***[n, k, dmin]***, Le Code est dit cyclique si l'ensemble des mots du code est stable Par d´ecalage circulaire.

Pour ***C*** un mot code donne par :

$$C = (C_0, C_1, C_2, \ldots\ldots, C_{n-1}) \quad \text{(Eq.10)}$$

Après le décalage circulaire :

$$C1 = (C_0, C_1, C_2, \ldots\ldots, C_{n-1}) \quad \text{(Eq.11.1)}$$

$$C2 = (C_{n-1}, C_0, C_1, \ldots\ldots, C_{n-2}) \quad \text{(Eq.11.2)}$$

$$C3 = (C_{n-1}, C_{n-2}, C_0, \ldots\ldots, C_{n-3}) \quad \text{(Eq.11.3)}$$

Selon la propriété de décalage cyclique, un décalage vers la droite ou vers la gauche dans les bits d'un mot de code doit générer un autre mot de code.**[15]**
Le polynôme du mot de code sera représenté comme :

$$C(x) = C_0 + C_1x^1 + C_2x^2 + \ldots\ldots + C_{n-1}x^{n-1} \quad \text{(Eq.12)}$$

Par example le code C = [1010], le polynôme est :

$$C(x) = 1*x^0 + 0*x^1 + 1*x^2 + 0*x^3 \quad \text{(Eq.13.1)}$$

$$C(x) = 1 + x^2 \quad \text{(Eq.13.2)}$$

Pour le polynôme générateur ***G*** est :

$$G(x) = g_0 + g_1x + g_2x^2 + \ldots\ldots + g_{n-k}x^{n-k} \quad \text{(Eq.14)}$$

Codage

Pour le codage, les mots de code sont classés en mots de code systématiques et non systématiques :

1. Un forme non systématique :

Le code non systématique est celui dans lequel les bits de données et de parité existent dans un format mélangé.

Example : pour le code C(7.4) avec un polynôme générateur :

$$G(x) = x^3 + x + 1 \quad \text{(Eq.15)}$$

Considérant un message :

$$m = [0110] \quad \text{(Eq.16)}$$

Ça donne :

$$M(x) = x + x^2 \quad \text{(Eq.17)}$$

Le mot de code est donné par :

$$C(x) = M(x).G(x) \quad \text{(Eq.18)}$$

Alors :

$$C(x) = (x + x^2)\,(x^3 + x + 1) \quad \text{(Eq.19.1)}$$

$$C(x) = x^5 + x^4 + x^3 + x^2 + x^2 + x \quad \text{(Eq.19.2)}$$

L'addition sera effectué avec XOR et la somme de 2 bits similaires donne 0.

Donc :

$$C(x) = x^5 + x^4 + x^3 + x \quad \text{(Eq.20)}$$

Le mot de code sera :

$$C = [0101110] \quad \text{(Eq.21)}$$

2. Un forme systématique :

Un mot de code systématique est un mot dans lequel les bits de parité et les bits de message sont présents sous des formes séparées.

$$C = [\text{Bits de parité} \mid \text{Bits de données}] \quad \text{(Eq.22)}$$

Considérant un message :

$$m = [1110] \quad \text{(Eq.23)}$$

Alors, le polynôme du message doit être :

$$M(x) = 1 + x + x^2 \quad \text{(Eq.24)}$$

Et le polynôme générateur :

$$G(x) = x^3 + x + 1 \quad \text{(Eq.25)}$$

L'équation pour déterminer le mot de code dans la forme systématique donne par :

$$C(x) = X^{n-k}\, M(x) + P(x) \quad \text{(Eq.26)}$$

X^{n-k} : Le degré de polynôme générateur , alors $X^{n-k} = x^3$
P(x) = le polynôme de parité et donne par :

$$P(x) = \text{Rem}\ \{X^{n-k}\, M(x) / G(x)\} \quad \text{(Eq.27)}$$

Rem : c'est le reste de la division.

Pour déterminer P(x) :

$$P(x) = \text{Rem}\ \{X^{n-k}\, M(x) / G(x)\} = \{X^{7-4}(x^2 + x + 1 / x^3 + x + 1\} \quad \text{(Eq.28.1)}$$

$$P(x) = \text{Rem}\ \{x^3\,(x^2 + x + 1) / (x^3 + x + 1)\} \quad \text{(Eq.28.2)}$$

$$P(x) = \text{Rem}\ \{(x^5 + x^4 + x^3) / (x^3 + x + 1)\} \quad \text{(Eq.28.3)}$$

Après la division le reste est : P(x) = x

Donc :

$$C(x) = X^{n-k}\, M(x) + P(x) \quad \text{(Eq.29.1)}$$

$$C(x) = x^3(x^2 + x + 1) + x \quad \text{(Eq.29.2)}$$

$$C(x) = x + x^3 + x^4 + x^5 \quad \text{(Eq.29.3)}$$

Alors, le code est :

$$C = [0101110] \quad \text{(Eq.30)}$$

Les bits de parité est (010) et les bits de message est (1110) ,nous remarquons que les bits de message n'est pas effectué.

Décodage

Soit R(x) le polynôme du code reçu et E(x) le polynôme erreur.

Le syndrome de l'erreur donne par cette équation :

$$S = \text{Rem}\ \{\frac{R(x)}{G(x)}\} \quad \text{(Eq.31)}$$

Si R(x) est un combinaison d'un polynôme code C et polynôme erreur E :

$$R(x) = C(x) + E(x) \quad \text{(Eq.32)}$$

Alors, nous trouvons :

$$S = \text{Rem}\ \{\frac{C(x)+ E(x)}{G(x)}\} \quad \text{(Eq.33)}$$

$$S = \text{Rem}\ \{\frac{C(x)}{G(x)}\} + \text{Rem}\ \{\frac{E(x)}{G(x)}\} \quad \text{(Eq.34)}$$

Puisque c'est :

$$S = \text{Rem}\ \{\frac{C(x)}{G(x)}\} = 0 \quad \text{(Eq.35)}$$

Par ce que le mot de code C est un correct.

Alors, l'équation de S doit être comme :

$$S = \text{Rem}\ \{E(x)/G(x)\} \quad \text{(Eq.36)}$$

Example : supposons un erreur dans le bit 3ème ,donc le code erreur est :

$$e = [0000100] \quad \text{(Eq.37)}$$

Et le polynôme erreur :

$$E(x) = x^4 \quad \text{(Eq.38)}$$

Alors, pour calculer le syndrome de cette erreur on utilisant l'eq :

$$S = \text{Rem } \{\frac{x^4}{x^3 + x + 1}\} \quad \text{(Eq.39)}$$

Après le calcule , le reste est :

$$S = x^2 + x \quad \text{(Eq.40)}$$

Ça c'est le syndrome correspondant à l'erreur polynôme pour le 3ème bit.

Example : nous avons le mot de code que trouvé avant dans l'étape de codage :

$$C = [0101110] \rightarrow C(x) = x^5 + x^4 + x^3 + x \quad \text{(Eq.41)}$$

Supposons que l'erreur s'est produite dans la position 3ème, donc le mot reçu est :

$$R = [0101010] \rightarrow R(x) = x^5 + x^3 + x \quad \text{(Eq.42)}$$

Pour calculer le syndrome en utilisant l'eq qui donné :

$$S = \text{Rem } \{\frac{x^5 + x^3 + x}{x^3 + x + 1}\} \quad \text{(Eq.43)}$$

Donc , le résultat nous trouvons est :

$$S = x^2 + x \quad \text{(Eq.44)}$$

Nous remarquons que le résultat est le même lorsque nous utilisons le polynôme reçu ou le polynôme erreur.

III.2.2.1 Quasi-cyclique (16.8)

Le code quasi-cyclique (16,8) est un code correcteur d'erreurs qui transforme 8 bits d'information en des mots codés de 16 bits.

Il appartient à la famille des codes linéaires, mais se distingue par sa structure régulière : si l'on effectue une rotation cyclique par blocs sur un mot de code, le résultat reste un mot valide.

Matrice de codage et décodage

Pour la matrcie ***G*** et ***H*** nous avons deux forme, la forme systématique et non systématique avec le polynôme générateur utilisé :

$$g(x) = x^8 + x^7 + x^6 + x^4 + x^2 + 1 \qquad \text{(Eq.45)}$$

1. La forme non systématique :

La matrice génératrice ***G*** peut être construite en décalant le polynôme vers la droite et peut être écrite comme :

$$G = \begin{bmatrix} g(x) \\ xg(x) \\ x^2g(x) \\ . \\ . \\ xk-1g(x) \end{bmatrix}$$

Donc la matrice ***G*** :

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

La matrice ***H*** peut être construite en utilisant le polynôme quotient ***h(x)*** de la division du $x^n + 1$ par le polynôme ***g(x)*** .

Dons ce cas :

$$x^n + 1 = x^{16} + 1 \qquad \text{(Eq.46)}$$

Le quotient de la division est :

$$h(x) = x^8 + x^7 + x^5 + x^2 + x + 1 \quad \text{(Eq.47)}$$

La matrice ***H*** peut être :

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

Example : supposons un message :

$$m = (11110000) \quad \text{(Eq.48)}$$

Nous codons le message par la matrice ***G*** pour obtenir le mot de code ***c*** :

$$c = (1100000111010000) \quad \text{(Eq.49)}$$

Ensuite, on vérifions la validité de mot de code par la matrice de décodage ***H,*** si le syndrome égal à 0 c'est-à-dire le mot de code est valide ,si non il y à un erreur.

$$S = (00000000) \quad \text{(Eq.50)}$$

Donc le mot de code ***c*** valide.

2. La forme systématique :

Dans la forme systématique, nous séparons les bits de parité et les bits de données et pour obtenir ce forme on utilisons l'équastion suivant :

$$G = [P \quad I_{n-k}] \quad \text{(Eq.51)}$$

La matrice ***P*** peut être construite avec cette division :

$$x^{(n-k+i)} / g(x) \quad \text{(Eq.52)}$$

i c'est le numéro de chaque ligne .

Par example le 1ère ***i = 0*** donc :

$$\frac{x8}{g(x)} = x^7 + x^6 + x^4 + x^2 + 1 \qquad \text{(Eq.53)}$$

Et jusqu'à la dernière ligne ***i = k-1***

Après le calcule on trouvons la matrice ***P*** :

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

Donc la matrice ***G*** doit être :

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

Pour la matrice ***H*** peut être construite avec cette équation :

$$H = [I_{k\times k} \quad P^T] \qquad \text{(Eq.54)}$$

Donc :

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

Example : supposons le même message de dernière fois m = (11110000) .

Dons le codage on trouvons le mot ***c :***

$$c = (1111111011110000) \quad \text{(Eq.55)}$$

Et dans le décodage on trouvons :

$$S = (00000000) \rightarrow \text{aucune erreur} \quad \text{(Eq.56)}$$

Mais supposons un erreur dans le 4ème bit :

$$r = (1111111011111000) \quad \text{(Eq.57)}$$

Après le calcule de syndrome on trouvons :

$$S = (01001010) \quad \text{(Eq.58)}$$

C'est le 13ème colonne de matrice ***H*** associé du 4ème bit .

Dans le cas de deux bits par exemple le 1ère et 2ème bit :

$$r = (1111111011110011) \quad \text{(Eq.59)}$$

Après le calcule de syndrome :

$$S = (01001110) \quad \text{(Eq.60)}$$

Le syndrome de deux bits peut être calculant par l'addition XOR de deux syndromes associés des bits erronées ou les colonnes de la matrice ***H.***

Par example notre cas le 1ère et 2ème bits :

$$11110111 \oplus 10111001 = 01001110 \quad \text{(Eq.61)}$$

Nous constatons que c'est le même résultat que précédemment.

III.2.3 Les codes hsiao

Le code Hsiao est une variante optimisée du code de Hamming, conçue pour offrir à la fois :

- La correction d'une seule erreur (SEC : Single Error Correction),
- La détection de deux erreurs (DED : Double Error Detection),
- Avec une complexité matérielle réduite.

Il a été proposé par Ming-Der Hsiao en 1970 pour améliorer l'efficacité des circuits de contrôle d'erreurs dans les systèmes numériques.

III.2.3.1 Hsiao (22.16)

La Structure du code Hsiao (22,16) est :

- **k = 16 :** bits de données.
- **n = 22 :** bits totaux (données+ parité).
- **r = n – k = 6 :** bits de parité.

Le mot code C est à la forme **systématique :**

$$C = \{p_5\ p_4\ p_3\ p_2\ p_1\ p_0\ d_{15}\ d_{14}\ d_{13}\ d_{12}\ d_{11}\ d_{10}\ d_9\ d_8\ d_7\ d_6\ d_5\ d_4\ d_3\ d_2\ d_1\ d_0\} \quad \text{(Eq.62)}$$

1. **Construction de la matrice de parité P :**

La matrice P est de taille r × k = 6 × 16.
Elle doit satisfaire plusieurs conditions :

- Pas de colonne nulle (pour détecter les erreurs).
- Chaque colonne a un poids impair (souvent poids 3).
- Pas de doublons entre colonnes.
- Pas de combinaison de deux colonnes qui donne une troisième colonne, pour assurer détection double-erreur.**[16]**

Ça donne :

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

2. Principe de codage

Principe de codage d'un code Hsiao repose sur la génération d'un mot codé de ***n*** bits à partir d'un mot de k bits de données, en ajoutant $r = n - k$ bits de parité calculés selon une matrice génératrice ***G.***

La matrice génératrice ***G*** est de dimension $\mathbf{k \times n = 16 \times 22}$ et donne par l'equation :

$$G = [P^T \ I_k] \qquad \text{(Eq.63)}$$

Nous avons la matrice P donc :

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

Ou calculé les bits de parité par l'addition (XOR) :

$P0 = d0 \oplus d1 \oplus d3 \oplus d4 \oplus d6 \oplus d10 \oplus d11 \oplus d14$
$P1 = d0 \oplus d2 \oplus d3 \oplus d5 \oplus d9 \oplus d10 \oplus d13 \oplus d15$
$P2 = d1 \oplus d2 \oplus d4 \oplus d5 \oplus d8 \oplus d9 \oplus d12 \oplus d14$
$P3 = d0 \oplus d1 \oplus d2 \oplus d3 \oplus d4 \oplus d7 \oplus d8 \oplus d11 \oplus d13$
$P4 = d0 \oplus d1 \oplus d2 \oplus d5 \oplus d6 \oplus d7 \oplus d10 \oplus d12 \oplus d15$
$P5 = d0 \oplus d1 \oplus d4 \oplus d5 \oplus d6 \oplus d9 \oplus d11 \oplus d13 \oplus d14$

On obtient le même résultat que la matrice génératrice .

3. Principe de décodage

La méthode de décodage du code hsiao est le même de code hamming .

Construction de la matrice de contrôle H

La matrice H est de dimension $r \times n$ et donne par l'equation :

$$H = [I_r \quad P] \qquad \text{(Eq.64)}$$

Donc :

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

- **Décodage par syndrome**

Soit r est un mot code recû dans bloc linéaire C donne par $r = c + e$:

$$r = \{p5'p4'p3'p2'p1'p0'd15'd14'd13'd12'd11'd10'd9'd8'd7'd6'd5'd4' \; d3'd2'd1'd0'\} \qquad \text{(Eq.65)}$$

L'equation de syndrome donne par :

$$s = rH^T = (c + e)H^T = eH^T \qquad \text{(Eq.66)}$$

- ✓ Si $\mathbf{s = 0}$, aucune erreur détectée.
- ✓ Si $\mathbf{s \neq 0}$, erreur détectée.

Le syndrome peut être calculé avec :

$S0 = p0' \oplus d0' \oplus d1' \oplus d3' \oplus d4' \oplus d6' \oplus d10' \oplus d11' \oplus d14'$
$S1 = p1' \oplus d0' \oplus d2' \oplus d3' \oplus d5' \oplus d9' \oplus d10' \oplus d13' \oplus d15'$
$S2 = p2' \oplus d1' \oplus d2' \oplus d4' \oplus d5' \oplus d8' \oplus d9' \oplus d12' \oplus d14'$
$S3 = p3' \oplus d0' \oplus d1' \oplus d2' \oplus d3' \oplus d'4 \oplus d7' \oplus d8' \oplus d11' \oplus d13'$
$S4 = p4' \oplus d0' \oplus d1' \oplus d2' \oplus d5' \oplus d6' \oplus d7 \; '\oplus d10' \oplus d12' \oplus d15'$
$S5 = p5' \oplus d0' \oplus d1' \oplus d4' \oplus d5' \oplus d6' \oplus d9' \oplus d11' \oplus d13' \oplus d14'$

- **Table de décodage par syndrome**

Ce table est 23 ligne et 2 colonne , contient des syndromes et leurs vecteurs d'erreur associés

BITS DE SYNDROME	LOOK UP TABLE
000000	*00000000000000000000*
100000	*10000000000000000000*
010000	*01000000000000000000*
001000	*00100000000000000000*
000100	*00010000000000000000*
000010	*00001000000000000000*
000001	*00000100000000000000*
010010	*00000010000000000000*
100101	*00000001000000000000*
101010	*00000000100000000000*
010100	*00000000010000000000*
101001	*00000000001000000000*
010011	*00000000000100000000*
100110	*00000000000010000000*
001100	*00000000000001000000*
011000	*00000000000000100000*
110001	*00000000000000010000*
110110	*00000000000000001000*
101101	*00000000000000000100*
001011	*00000000000000000010*
011110	*00000000000000000001*
111101	
111011	

Tableau III.3 : Look up table pour le code hsiao (22.16)

III.3 Conclusion

Les codes correcteurs d'erreurs jouent un rôle fondamental dans la fiabilité des systèmes numériques, notamment dans les communications, les mémoires et les environnements sensibles aux perturbations comme l'espace. Parmi eux, les codes **Hamming**, **Hsiao** et **quasi-cycliques** offrent différentes solutions selon les besoins en performance, en détection et en complexité.

Chapitre IV
Conception des circuits EDAC (Error Detection And Correction)

IV.1 Introduction

Les systèmes embarqués modernes, en particulier ceux utilisés dans les applications spatiales, doivent faire face à des environnements hostiles où les radiations peuvent provoquer des erreurs dans les circuits électroniques, y compris la mémoire SRAM, largement utilisée en raison de sa vitesse, de sa simplicité et de sa sensibilité particulière aux perturbations telles que les perturbations à événement unique (SEU), qui peuvent inverser de manière aléatoire l'état des bits. Pour garantir l'intégrité des données et maintenir la fiabilité du système, l'utilisation de circuits de détection et de correction d'erreurs (EDAC) est essentielle.

De nombreuses familles de codes de correction ont été développées à cet effet. Les codes de Hamming sont parmi les plus classiques, permettant à la fois la correction d'erreur simple et la détection d'erreur double avec un coût matériel modéré. Les codes Hsiao qui en découlent sont optimisés pour une implémentation matérielle plus efficace grâce à leur matrice de parité à faible poids, ce qui les rend particulièrement adaptés aux architectures à faibles ressources. Enfin, des codes quasi-cycliques plus robustes permettent de corriger plusieurs erreurs simultanément tout en conservant une structure régulière, idéale pour les applications FPGA et les systèmes à haute fiabilité.

Cette étude porte sur l'analyse et l'implémentation de ces différents codes dans le cadre de circuits EDAC conçus pour protéger les blocs SRAM des erreurs induites par rayonnement.

IV.2 Architecture d'encodage et décodage

❖ Encodage

Le processus d'encodage des données avant leur écriture dans une architecture mémoire protégée par un mécanisme de correction d'erreurs (Figure IV.1). Les données générées par le processeur (CPU) sont d'abord envoyées vers un encodeur, constitué ici de portes logiques XOR. Cet encodeur applique un algorithme de codage (comme Hamming, Hsiao ou un code quasi-cyclique) afin de générer les bits de parité correspondants à chaque mot de données. Ensuite, les données originales sont écrites dans la mémoire principale (Data SRAM), tandis que les bits de parité générés sont stockés dans une mémoire dédiée (Parity SRAM). Cette séparation permet de protéger efficacement les données contre les erreurs simples ou multiples, tout en facilitant les opérations de décodage et de correction lors des lectures futures.

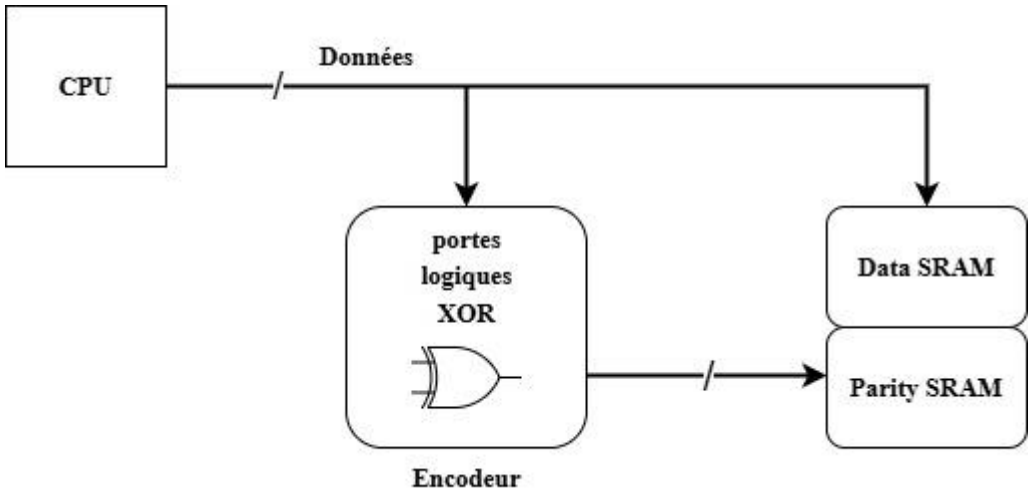


Figure IV.1 : Architecture d'encodage

❖ Décodage

Le processus de décodage et de correction d'erreurs dans une architecture mémoire protégée par un code correcteur (figure IV.2), comme Hamming ou Hsiao. Lors d'une opération de lecture, les données sont récupérées depuis la Data SRAM tandis que les bits de parité correspondants sont lus depuis la Parity SRAM. Ces deux ensembles sont envoyés à un module de génération de syndrome basé sur des portes logiques XOR. Le résultat, noté s, représente le syndrome indiquant la présence et la nature potentielle d'une erreur. Ce syndrome est ensuite transmis à une LUT (Look-Up Table) qui interprète sa signification et fournit un vecteur d'erreur e. Ce vecteur est dirigé vers le bloc de correction, qui applique les modifications nécessaires aux données initialement lues. Les données corrigées d' sont alors transmises au CPU. Ce mécanisme assure la fiabilité des lectures mémoire, même en présence d'erreurs simples, sans intervention logicielle, ce qui est essentiel dans les systèmes embarqués critiques.

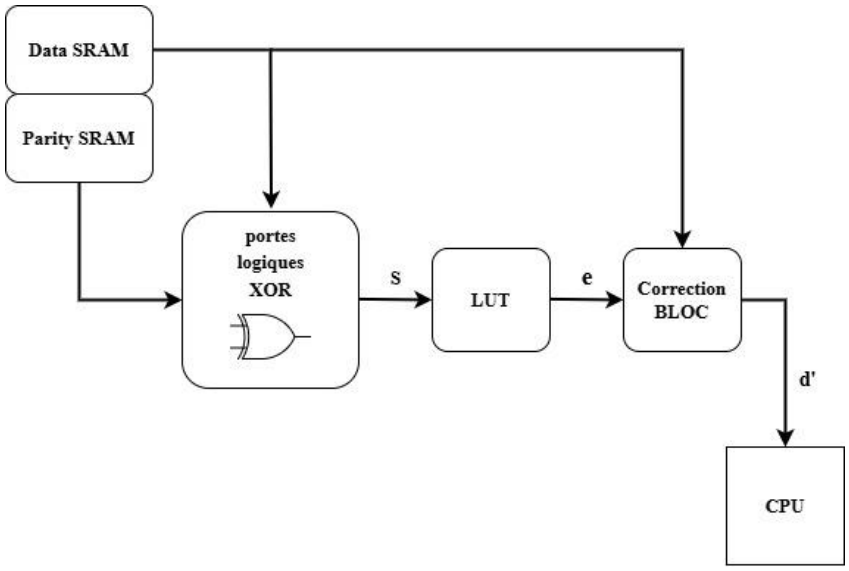


Figure IV.2 : Architecture de décodage

IV.3 Circuit EDAC basé sur le code hamming

Le code hamming utilise une architecture matérielle basée sur une FPGA (XC7A200T) intégrant une logique EDAC pour la correction d'erreurs en mémoire (figure IV.3). Le CPU 16 bits envoie des données qui sont scindées en deux mots de 8 bits, chacun traité par un bloc EDAC-Hamming pour générer 4 bits de parité. Les 8 bits de données sont stockés dans une mémoire SRAM principale, tandis que les 4 bits de parité sont enregistrés dans une SRAM dédiée aux parités. Lors de la lecture, les données et les parités sont récupérées et corrigées si nécessaire avant d'être renvoyées au CPU, assurant une fiabilité renforcée face aux erreurs simples.

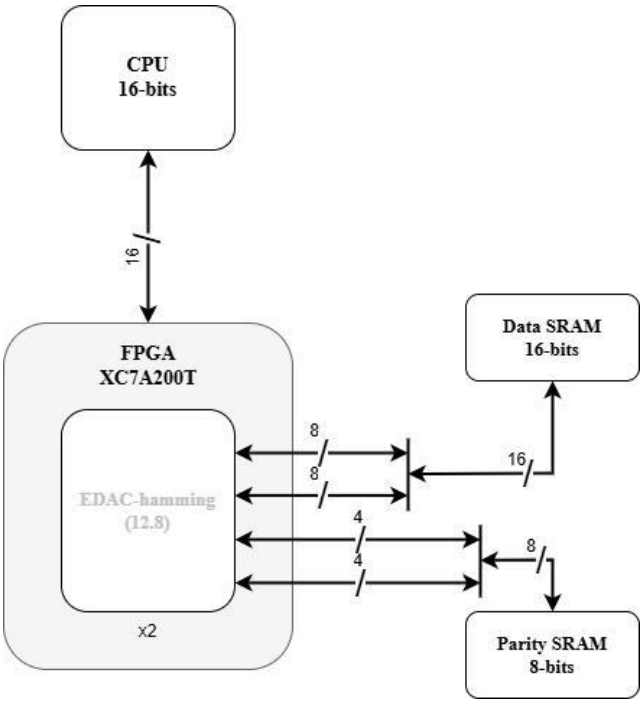


Figure IV.3 : Architecture FPGA XC7A200T intégrant un système EDAC avec code correcteur Hamming (12.8)

- **Mécanisme de codage et d'écriture**

Le mécanisme d'encodage et d'écriture des données dans un système mémoire équipé d'un code de correction d'erreurs de type Hamming (12,8) représente dans le figure(IV.4). Lorsqu'une donnée de 8 bits provenant du processeur (CPU) est adressée à la mémoire, elle est d'abord transmise au générateur de parité, qui calcule les 4 bits de parité associés selon les règles du code Hamming. Ces bits de parité (P0 à P3) sont ensuite envoyés vers une mémoire dédiée appelée Parity Memory, tandis que les 8 bits de données d'origine sont stockés dans la Data Memory. Ce découplage entre données et parités permet une organisation mémoire optimisée et facilite les opérations de correction ultérieures. Le système conserve ainsi, pour chaque adresse mémoire, un mot codé de 12 bits réparti entre les deux blocs mémoire (8 bits pour les données, 4 pour la parité), assurant la correction d'une erreur simple et la détection d'une double erreur lors des lectures.

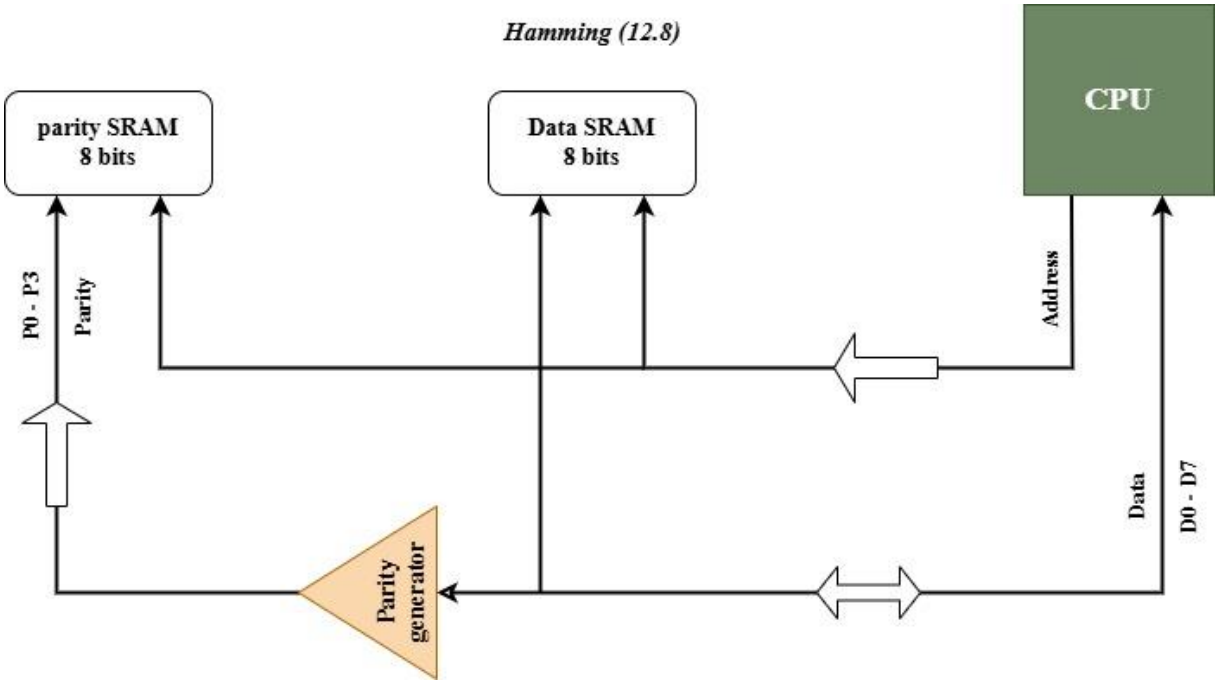


Figure IV.4 : Schéma d'encodage et d'écriture mémoire avec code hamming (12.8)**.[17]**

- **Mécanisme de décodage et lecture**

Le processus de décodage et de lecture dans la structure de mémoire protégée par le code de Hamming (12,8) est illustré dans la figure (IV.5). Lorsqu'une lecture de mémoire est initiée par le processeur (CPU), 8 bits de données sont récupérés de la mémoire de données, tandis que les 4 bits de parité correspondants sont lus simultanément à partir de la mémoire de parité. Ces deux ensembles (données et parités) sont ensuite transmis au générateur de syndrome, qui calcule le syndrome d'erreur en comparant la parité reçue (l'ancienne parité) avec la nouvelle parité recalculée à partir des données lues. Si le syndrome est nul, les données sont considérées comme normales. En revanche, si le syndrome indique une erreur, il est envoyé à l'unité de correction, qui est capable de localiser le bit erroné et de le corriger dynamiquement avant de renvoyer le mot corrigé au processeur. Ce mécanisme permet de corriger les erreurs en cours de fonctionnement sans intervention logicielle, tout en conservant la traçabilité via un compteur EDAC qui enregistre les corrections apportées. Cette stratégie garantit une fiabilité élevée du système de mémoire, ce qui est essentiel dans les applications critiques où l'intégrité des données est de la plus haute importance.

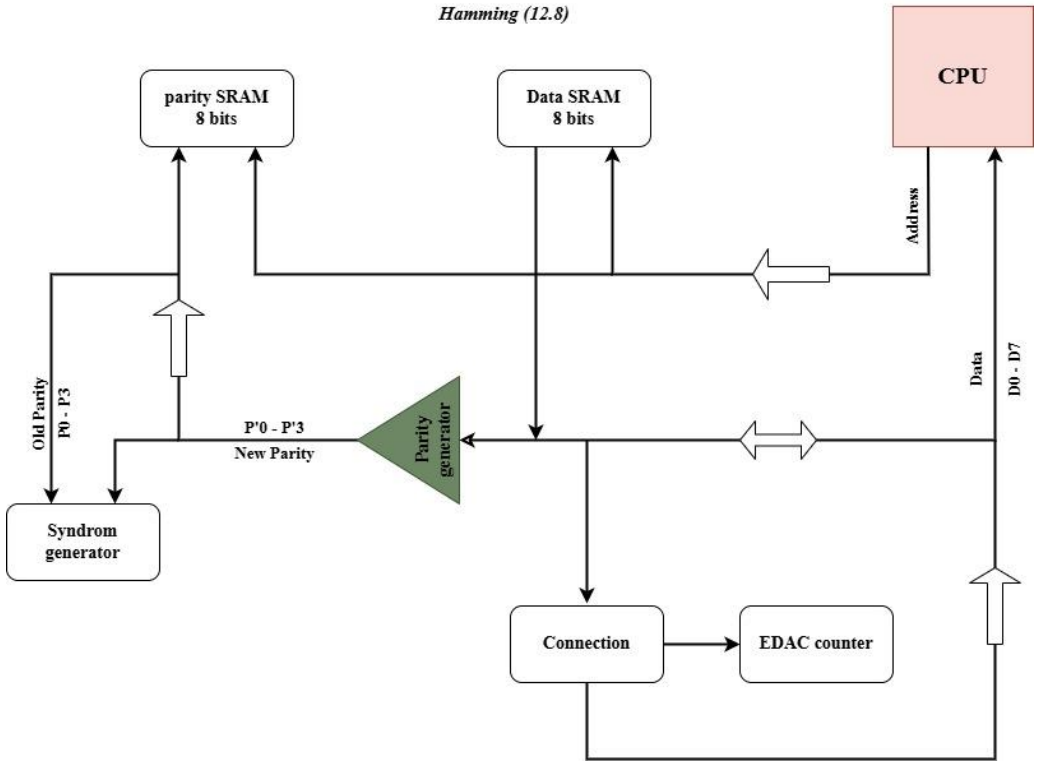


Figure IV.5 : Schéma de décodage et de lecture mémoire avec code hamming (12.8)

IV.4 Circuit EDAC basé sur le code Quasi-cyclique

Pour le code quasi-cyclique (16,8) ,Le CPU 16 bits envoie des données qui sont divisées en deux mots de 8 bits, chacun étant encodé par un bloc EDAC quasi-cyclique afin de produire un mot codé de 16 bits, comprenant 8 bits d'information et 8 bits de redondance. Les mots complets sont ensuite stockés dans une mémoire SRAM, soit en totalité, soit partagés entre une mémoire de données et une mémoire de parité. Lors de la lecture, les 16 bits sont récupérés et traités par le décodeur, qui calcule le syndrome et corrige les erreurs détectées selon les règles du code quasi-cyclique.(Figure IV.6)

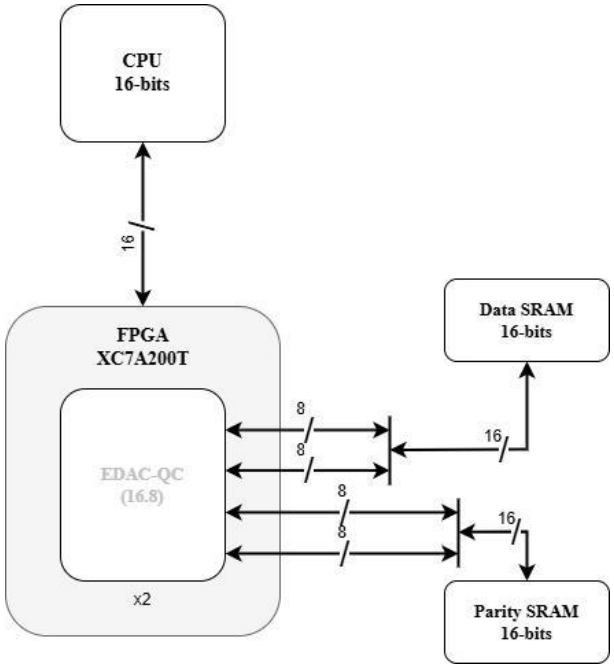


Figure IV.6 : Architecture FPGA XC7A200T intégrant un système EDAC avec code correcteur Quasi-cyclique (16.8)

- **Mécanisme de codage et d'écriture**

Le mécanisme d'encodage et d'écriture des données dans un système mémoire protégé par un code correcteur d'erreurs quasi-cyclique (16,8). Lorsqu'un mot de 8 bits provenant du processeur (CPU) est destiné à la mémoire, il est tout d'abord transmis à l'encodeur quasi-cyclique, qui génère un mot codé de 16 bits en appliquant une transformation linéaire basée sur une matrice génératrice spécifique. Ce mot codé contient les 8 bits de données originales ainsi que 8 bits de redondance. Les 16 bits sont ensuite stockés dans la mémoire, soit dans un bloc unifié, soit en deux mémoires distinctes l'une pour les données et l'autre pour les bits de parité. Cette architecture permet une meilleure tolérance aux erreurs multiples, tout en maintenant une organisation structurée adaptée à la détection et à la correction lors de la lecture. Le recours à un codage plus robuste est essentiel dans les systèmes embarqués soumis à de fortes contraintes de fiabilité, comme ceux utilisés dans les applications spatiales, militaires ou industrielles critiques.(figure IV.7)

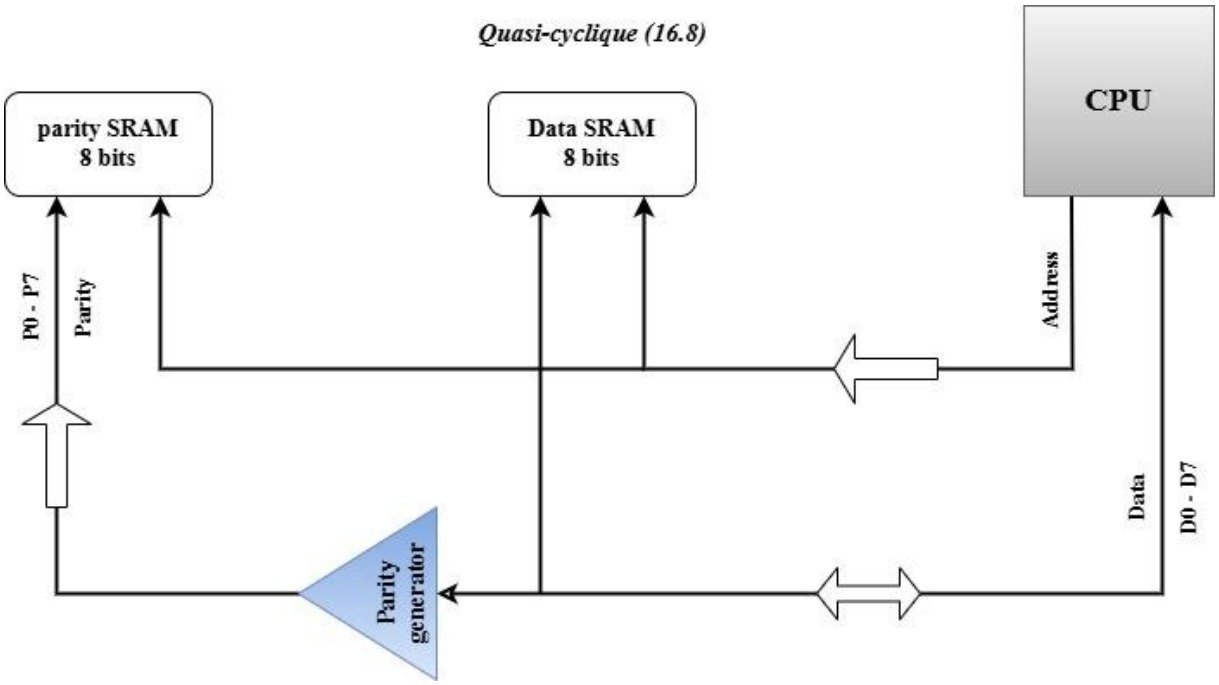


Figure IV.7 : Schéma d'encodage et d'écriture mémoire avec code Quasi-cyclique (16.8)

- **Mécanisme de décodage et lecture**

Le processus de décodage et de lecture dans une architecture mémoire sécurisée à l'aide d'un code quasi-cyclique (16,8). Lorsqu'une opération de lecture est lancée par le processeur (CPU), un mot complet de 16 bits est extrait de la mémoire, contenant à la fois les 8 bits de données et les 8 bits de redondance générés lors de l'écriture. Ce mot codé est ensuite transmis au décodeur quasi-cyclique, qui procède au calcul du syndrome à l'aide d'une matrice de contrôle (H). Le syndrome permet d'évaluer si une ou plusieurs erreurs sont présentes dans le mot lu. Si le syndrome est nul, le mot est considéré comme valide. Dans le cas contraire, le syndrome est analysé pour localiser les erreurs à l'aide d'une table de syndromes ou d'un algorithme de décodage spécifique, permettant de corriger jusqu'à deux erreurs, voire plus selon le code utilisé. Une fois les corrections appliquées, les 8 bits d'information sont extraits et renvoyés vers le processeur. Ce mécanisme matériel de correction automatique assure une forte résilience aux erreurs multiples, tout en réduisant la charge logicielle. Il garantit une fiabilité élevée des données, critère essentiel dans les systèmes embarqués soumis à des environnements agressifs, tels que les satellites, les équipements militaires ou les dispositifs industriels de sécurité.(Figure IV.8)

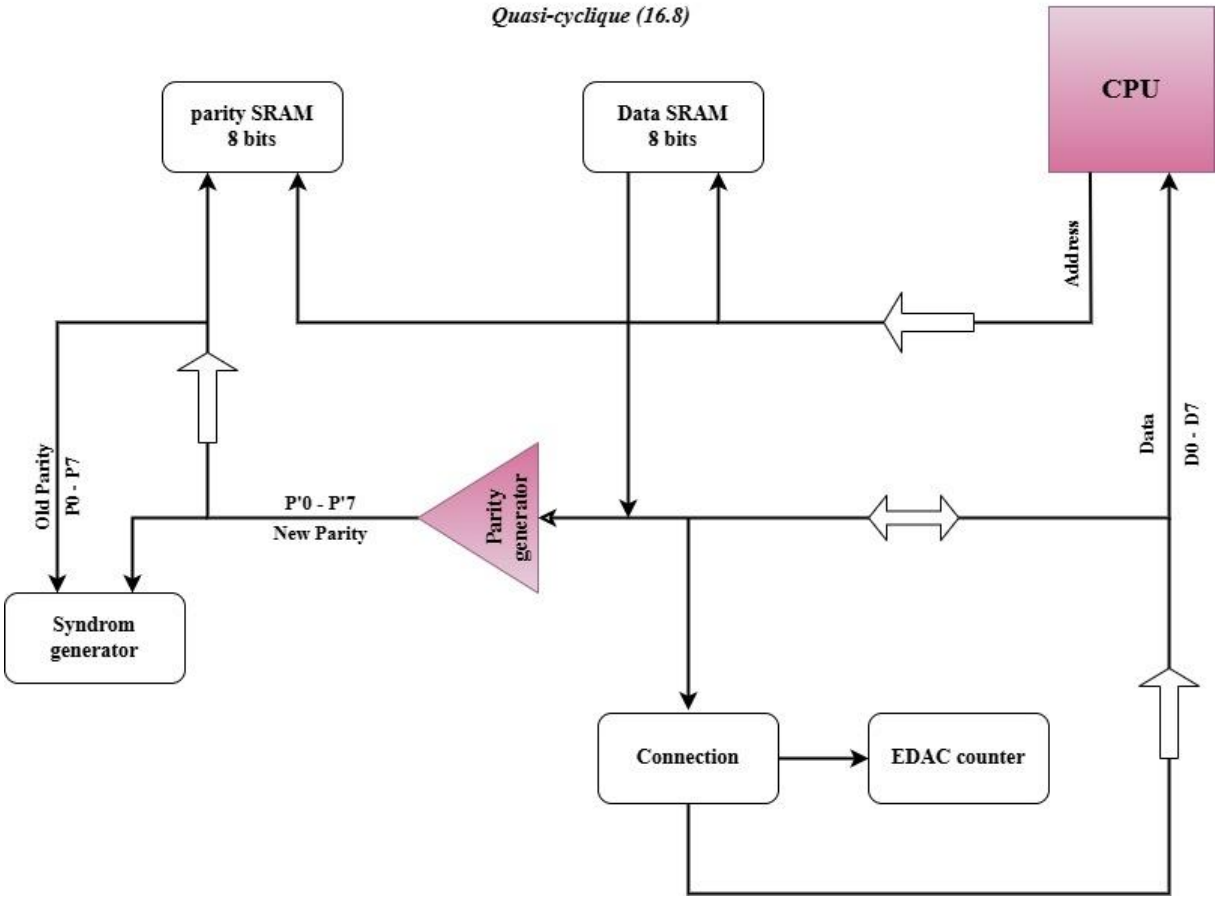


Figure IV.8 : Schéma de décodage et de lecture mémoire avec code Quasi-cyclique (16.8)

IV.5 Circuit EDAC basé sur le code hsiao

L'EDAC de code Hsiao (22,16), conçu pour assurer la correction d'erreurs simples et la détection d'erreurs doubles (figure IV.9) avec un minimum de bits redondants actifs. Le processeur 16 bits envoie des données qui sont traitées en blocs de 16 bits par un encodeur Hsiao, lequel génère 6 bits de redondance. Les 16 bits de données sont stockés dans une mémoire SRAM principale, tandis que les 6 bits de parité sont enregistrés dans une mémoire SRAM dédiée. Lors de la lecture, l'ensemble des 22 bits est récupéré et transmis à une unité de décodage capable de détecter et corriger les erreurs simples à la volée, avant de restituer les données corrigées au CPU. Ce schéma permet une robustesse accrue du système mémoire, essentielle dans les environnements à fortes contraintes comme le spatial ou l'aéronautique.

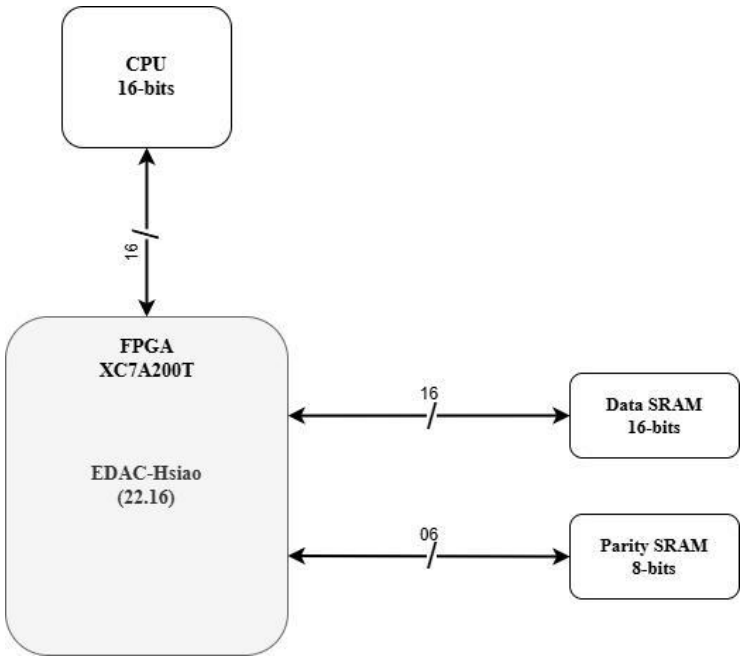


Figure IV.9 : Architecture FPGA XC7A200T intégrant un système EDAC avec code correcteur Hsiao (22,16)

- **Mécanisme de codage et l'écriture**

Le figure (IV.10) présente en détail le mécanisme d'encodage et d'écriture des données dans une architecture mémoire protégée par un code de correction d'erreurs de type Hsiao (22,16). Lorsqu'un mot de 16 bits issu du processeur (CPU) est transmis, il est d'abord acheminé vers un générateur de parité qui calcule 6 bits de redondance selon la matrice de Hsiao, reconnue pour sa capacité à détecter les erreurs doubles et corriger les erreurs simples tout en minimisant le nombre de bits à 1. Les 6 bits de parité sont ensuite stockés dans une mémoire dédiée aux parités (Parity Memory), tandis que les 16 bits de données originales sont enregistrés dans la mémoire principale (Data Memory). Ce découplage entre données et parités permet une organisation modulaire et efficace de l'espace mémoire, tout en préparant le système à une correction rapide lors de la lecture. Chaque adresse mémoire conserve ainsi un mot codé de 22 bits réparti sur deux mémoires physiques.

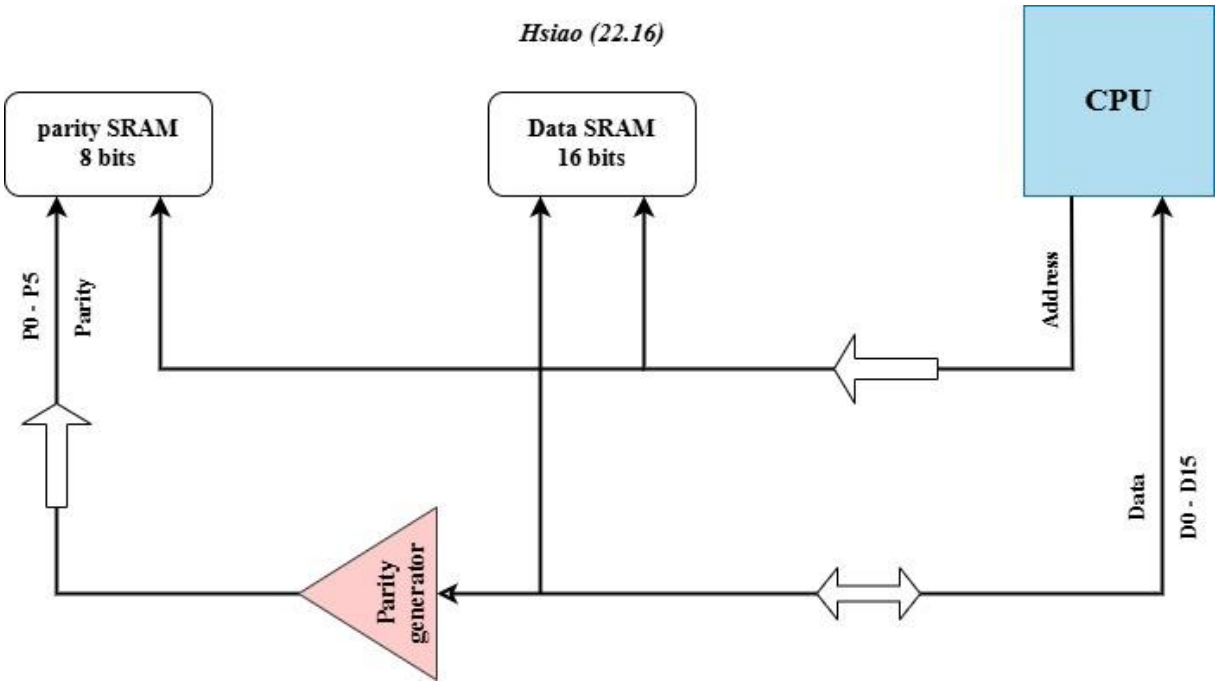


figure IV.10 : Schéma d'encodage et d'écriture mémoire avec code Hsiao (22,16)

- **Mécanisme de décodage et lecture**

Le processus de décodage et de lecture dans une architecture mémoire sécurisée à l'aide du code de Hsiao (22,16) représente dans le figure (IV.11), reconnu pour son efficacité dans la détection d'erreurs doubles et la correction d'erreurs simples. Lorsqu'une requête de lecture est initiée par le processeur (CPU), les 16 bits de données sont extraits depuis la Data Memory, tandis que les 6 bits de parité correspondants sont simultanément récupérés depuis la Parity Memory. Ces 22 bits sont ensuite traités par une unité de décodage, qui commence par recalculer les bits de parité à partir des données lues, puis les compare aux parités stockées afin de générer un syndrome d'erreur. Si le syndrome est nul, le mot est considéré comme valide. Dans le cas contraire, le syndrome est analysé par le correcteur EDAC pour localiser, puis corriger l'éventuel bit erroné. Grâce à la structure particulière du code de Hsiao, cette correction peut être effectuée rapidement et avec une faible consommation logique. Le système conserve également un compteur d'événements EDAC afin de suivre les erreurs détectées ou corrigées au fil du temps.

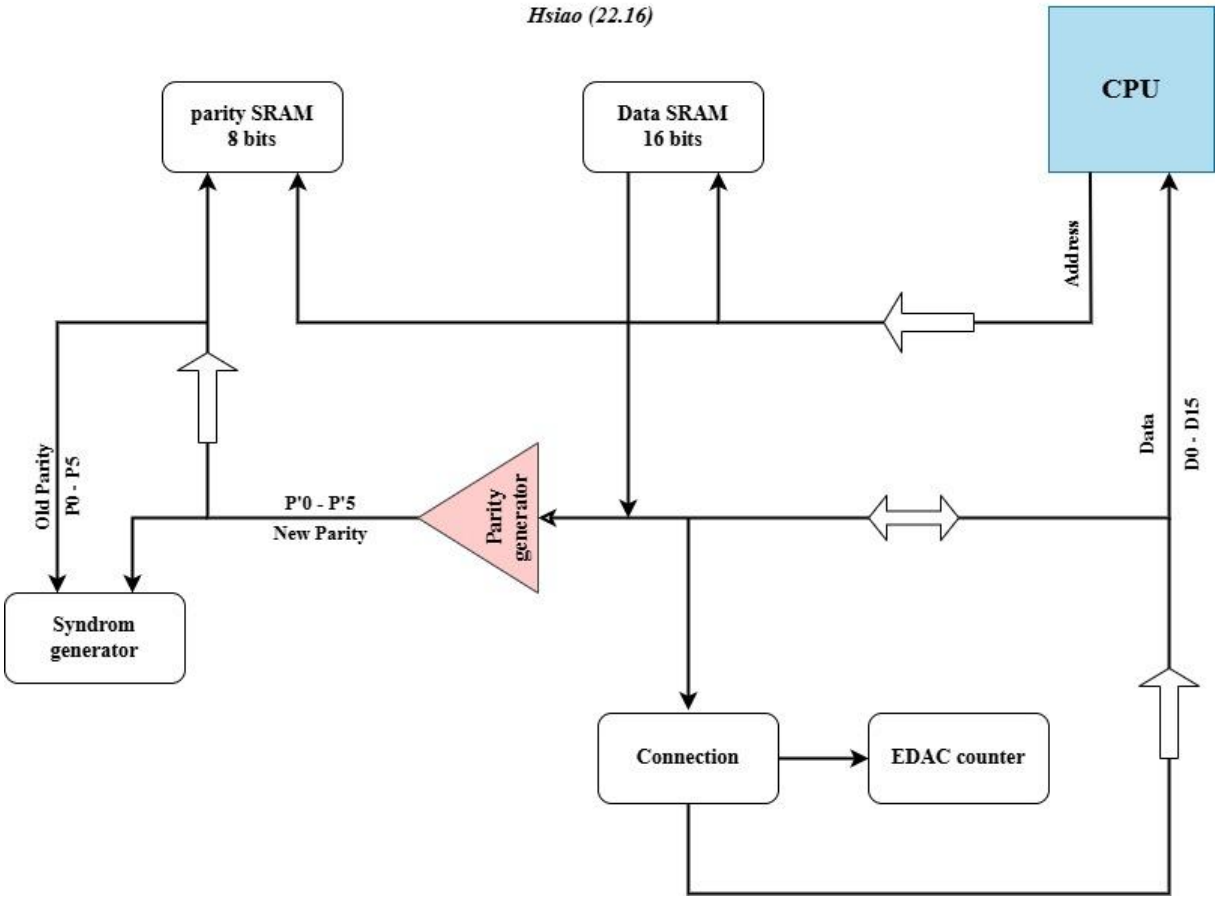


Figure IV.11 : Schéma de décodage et de lecture mémoire avec code Hsiao (22,16)**.[17]**

IV.6 Memory overhead

Dans le contexte de la détection et de la correction d'erreurs (EDAC), la surcharge mémoire désigne le nombre de bits (ou d'espace mémoire) supplémentaire requis pour stocker la redondance (bits de parité/de contrôle) avec les données d'origine. Il s'agit d'un compromis entre fiabilité et efficacité des ressources, calculant avec :

$$\boldsymbol{Overhead} = \frac{\boldsymbol{Parity}}{\boldsymbol{Data}} \times \mathbf{100\%} \qquad [\mathbf{18}] \tag{Eq.67}$$

1- Hamming (12,8) :

- Overhead = 0.5 = 50%.
- Protège 8 bits de données grâce à 4 bits supplémentaires.
- Le mot de code de 12 bits entraîne une surcharge mémoire de 50 %.

- Cette méthode est légère, mais ne corrige que les erreurs sur un seul bit et ne convient pas aux erreurs sur plusieurs bits (MBU).

2- Quasi-cyclique (16,8) :

- Overhead = 1.0 = 100%.
- Protège 8 bits de données grâce à 8 bits supplémentaires.
- La taille du mot de code est doublée, ce qui entraîne une surcharge mémoire de 100 %.
- Cependant, cette méthode peut corriger jusqu'à 2 erreurs, ce qui la rend robuste face aux MBU, au prix d'une utilisation mémoire plus importante.

3- Hsiao (22,16) :

- Overhead = 0.375 = 37.5%.
- Protège un bloc beaucoup plus grand de 64 bits grâce à seulement 8 bits supplémentaires.
- La surcharge est bien plus faible : seulement 12,5 %.
- Il corrige les erreurs sur un seul bit et détecte les erreurs sur deux bits (SEC-DED), ce qui le rend efficace et fiable pour les systèmes à haute fiabilité.

Tableau résumé :

EDAC	Data bits	Parity bits	Codeword	Memory overhead
Hamming (12.8)	**8**	**4**	**12**	**50 %**
Quasi-cyclique (16.8)	**8**	**8**	**16**	**100 %**
Hsiao (22.16)	**16**	**6**	**22**	**37.5 %**

Tableau IV.1 : Mémoire supplémentaire requise par différents codes EDAC

Finalement , Le code Hsiao offre le meilleur équilibre entre fiabilité et efficacité, Le code quasi-cyclique offre une protection renforcée, mais la surcharge est la plus élevée, tandis que le code Hamming est léger, mais moins résistant aux erreurs multiples.

IV.7 Conclusion

La mise en œuvre de mécanismes de correction d'erreurs (EDAC) dans les architectures mémoire est essentielle pour garantir l'intégrité des données, en particulier dans les systèmes embarqués critiques comme les satellites. À travers l'étude des codes Hamming (12,8), Hsiao (22,16) et quasi-cycliques (16,8), nous avons exploré différentes approches de protection mémoire. Chacune de ces méthodes présente un compromis entre complexité matérielle, capacité de correction et efficacité. Les architectures basées sur FPGA, telles que celles intégrant le XC7A200T, permettent de déployer ces logiques de codage et de décodage directement au niveau matériel, assurant une correction à la volée sans intervention logicielle. En séparant la mémoire des données de celle des parités, et en intégrant des modules de génération de syndrome et de correction automatique, ces systèmes garantissent une fiabilité élevée face aux perturbations comme les rayonnements spatiaux. L'ensemble des mécanismes étudiés dans ce chapitre met en évidence l'importance cruciale de l'EDAC dans le maintien de la performance et de la robustesse des systèmes numériques dans des environnements hostiles.

Chapitre V
Simulation des circuits EDAC

V.1 Introduction

Dans le cadre de la simulation et de l'implémentation des circuits de détection et correction d'erreurs (EDAC), le langage VHDL (VHSIC Hardware Description Language) a été utilisé. Ce langage est spécifiquement conçu pour la modélisation, la simulation et la synthèse de circuits numériques. Il permet de décrire le comportement logique ainsi que la structure des circuits à différents niveaux d'abstraction, ce qui en fait un outil particulièrement adapté à la conception de circuits de correction d'erreurs complexes comme les codes de Hamming, Hsiao ou quasi-cycliques.

Les simulations et synthèses ont été réalisées à l'aide de la suite logicielle Xilinx Vivado, un environnement de développement intégré largement utilisé dans l'industrie pour la conception de systèmes numériques sur FPGA. Vivado offre des outils puissants pour l'écriture du code VHDL, la simulation fonctionnelle et temporelle, ainsi que pour la synthèse logique et le placement-routage des circuits sur des FPGA de la famille Xilinx. Grâce à ses outils de visualisation et d'analyse, Vivado permet également d'évaluer les performances des circuits en termes de consommation de ressources, de fréquence maximale et de latence.

V.2 Langage VHDL

VHDL est un langage de description de matériel destiné à représenter le comportement ainsi que l'architecture d'un système électronique numérique. Son nom complet est VHSIC Hardware.

Description Language (VHSIC = Very High Speed Integrated Circuit).**[19]**

De plus, le VHDL :

- Peut être simulé,
- Peut être traduit en schéma de portes logiques.

V.2.1 Vhdl bases

Tout programme VHDL doit être composé au minimum d'une paire indissociable **entité/architecture**.

- Entité : elle décrit les entrées/sorties du composant,

- Architecture : elle décrit le fonctionnement interne du composant.

Il a également besoin de bibliothèques définissant les types et les opérations des différents signaux Utilisés. Typiquement :

Library ieee ;

use ieee . std_logic_1164. **all** ; — définition du type bit , bit vector , …

use ieee . numeric_std. **all** ; — opérations signées et non signées

use ieee . std_logic_arith . **all** ; — opérations signées et non signées sur les vecteur

1. **Entité déclaration**

L'entité (ou entity en anglais) définit l'interface d'un composant matériel. Elle décrit ce que le composant reçoit comme entrées, ce qu'il produit comme sorties.

Dans cette example de encoder pour le code hamming (12.8) l'entré est D avec 8 bits et la sortie C avec 12 bits :

Entity HMCODE **is**

Port (D : **in** std_logic_vector(7 **downto** 0) ;

C :**out** std_logic_vector(11 **downto** 0)) ;

End HMCODE ;

- Déclaration d'une entité HMCODE.
- Déclaration des ports d'entrées/sorties de cette entité.
- Déclaration du type des entrées/sorties (in, out, …).

2. Architecture déclaration

Une architecture décrit le fonctionnement du système, l'implémentation de la fonctionnalité voulue. On y définit des signaux, l'équivalent le plus proche des variables en programmation informatique**.**

- **Flot de données** Une architecture décrit le fonctionnement de l'entité via des équations booléennes, Ou des formes conditionnelles.**[19]**

Example : L'architecture de codage de hamming (12.8) :

Architecture HM_CONCURENT **of** HMCODE **is**

Begin

C(0) <= D(0) ;
C(1) <= D(1) ;
C(2) <= D(2) ;
C(3) <= D(3) ;
C(4) <= D(4) ;
C(5) <= D(5) ;
C(6) <= D(6) ;
C(7) <= D(7) ;
C(8) <= D(6)**xor** D(4)**xor** D(3)**xor** D(1)**xor** D(0) ;
C(9) <= D(6)**xor** D(5)**xor** D(3)**xor** D(2)**xor** D(0) ;
C(10) <=D(7)**xor** D(3)**xor** D(2)**xor** D(1) ;
C(11) <=D (7)**xor** D(6)**xor** D(5)**xor** D(4) ;

End HM_CONCURENT ;

Le code est calculé les bits de parité en utilisant les portes logiques (xor).

La description par flot de données peut devenir lourde pour des fonctions plus complexes. Il existe Deux autres moyens de décrire une architecture :

- **Structurelle** : tel un schéma, des composants sont instanciés et associés ensembles par des Signaux. On assemble ici une série de boîtes noires déjà écrites pour former un circuit plus Complexe.

Architecture HECC_ARCH **of** HECC **is**

Component HM_DECODE

Port (C :**in** std logic vector (11 **downto** 0) ; S :**out** std logic vector (3 **downto** 0)) ;

end component ;

Component LUT

Port (S :**in** std_logic_vector (3 **downto** 0) ; E : **in** std_logic ; El :**out** std logic_vector(7 **downto** 0))

end component :
Et les signaux :

Signal C1,C2 : std_logic_vector (11 **downto** 0) ;

Begin
INSTAN1 : QCCODE **port map**(D,C1) ;
INSTAN2 : ERR_GEN **port map**(C1,EG,C2) ;
INSTAN3 : QECC **port map**(C2,E,BC,ERR) ;
end struct ;

Ce type d'architecture utilise des éléments existants (dans un autre fichier ou une librairie) et les Connecte ensemble. Une instanciation est formée comme suit :

Un nom : pour identifier l'instanciation,
Un composant : qui est instancié,
Des connections : effectuées grâce à l'opération **port map**, qui indique comment connecter L'élément instancié avec les différents signaux.

- **Comportementale** : le comportement est décrit sous forme d'instructions séquentielles Rassemblées en processus.

Ce type d'architecture nécessite au moins un processus, qui comprend :

Nom : permettant d'identifier de manière unique le processus,

Liste de sensibilité : permettant d'identifier les signaux qui ont un impact sur le processus. Tout Changement dans au moins un de ces signaux provoque la réévaluation du Processus.
Liste d'instructions : qui, à la manière d'un programme C sont exécutées de manière séquentielle Pour la simulation, ou pour générer le circuit final.

3. Banc de test(Testbench)

Bien que ce ne soit pas une unité de conception au sens strict, un banc de tests est essentiel pour vérifier le comportement d'une conception VHDL. Il consiste en un ensemble De stimuli et de vérifications qui permettent de tester le composant ou le système.
En utilisant ces unités de conception, vous pouvez modéliser des systèmes électroniques De manière structurée et hiérarchique en VHDL, ce qui facilite la conception, la simulation et la vérification de vos projets.**[19]**

Template général :

```
library ieee;
use ieee.std_logic_1164.all;

entity test_UUT is -- empty entity
end test_UUT

architecture testbenk_arch of test_UUT is
Component UUT:
port
(
);
end component;

signal
signal :='0'; -- start value for inputs

begin
U1: UUT
port map (.
 .);
process
begin
wait;

end process;
end architecture;
```

Voici un processus pou code hamming (12.8) :

Process
begin

D <= « **11110000** » ;
EG <= « **000000000001** » ;
E <= ‘0’ ;
Wait for 100 ns ;

D<= « **11110000** » ;
EG <= « **000000000001** » ;
E <= ‘1’ ;
Wait for 100 ns ;

V.3 Les circuits FPGAs

Les FPGAs, en anglais « Field Programmable Gate Array », sont des circuits numériques qui peuvent être reconfigurés ou programmés. Ils facilitent le prototypage de circuits ASIC, ainsi que la création et la vérification de circuits et systèmes numériques simples ou sophistiqués. La configuration des circuits FPGA s’effectue en utilisant des langages de description matériel, tel que le langage VHDL (Very High Speed Integrated Circuits Hardware Description Language).**[20]**

V.3.1 Architecture

Un FPGA est une puce électronique formée d’un réseau de petits blocs logiques programmables, entourés de blocs pour gérer les entrées et les sorties, et reliés entre eux par un système de commutation permettant de créer n’importe quel circuit numérique.
La forme extérieure de la carte FPGA présenté sur le figure 5.1 suivent :

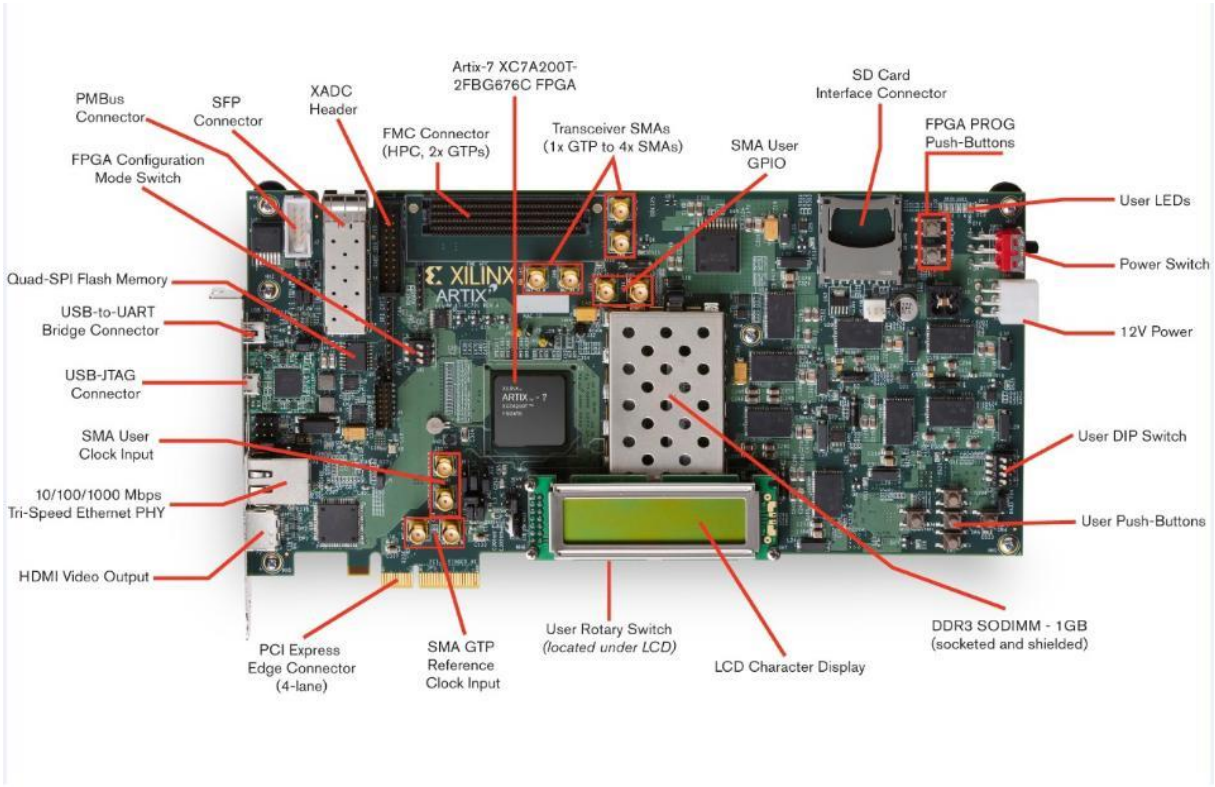


Figure V.1 : la carte FPGA face extérieure

Dans la forme interne, constitué à deux blocs principales connecté avec des interconnexion (figure V.2) :

- ✓ **Les blocs logiques programmables :** Les performances d'un FPGA dépendent principalement de ses blocs logiques configurables (CLB), qui intègrent à la fois des éléments de logique combinatoire et des registres de mémorisation basés sur des bascules D.

- ✓ **Les blocs d'entrées/sorties :** Les blocs d'entrées/sorties (IOBs) assurent l'interface entre les broches physiques du FPGA et la logique interne implémentée dans le circuit. Répartis tout autour de la périphérie du composant, chaque IOB est associé à une broche et peut être configuré pour fonctionner en entrée, en sortie.

- ✓ **Le réseau d'interconnexion :** Les connexions internes d'un FPGA sont constituées de segments métalliques. Le long de ces lignes de communication se trouvent des matrices de commutation, appelées switch blocks, disposées horizontalement et verticalement à travers le circuit, entre les différents CLB. Ces blocs de commutation permettent d'établir des liaisons entre les CLB, y compris sur de longues distances.

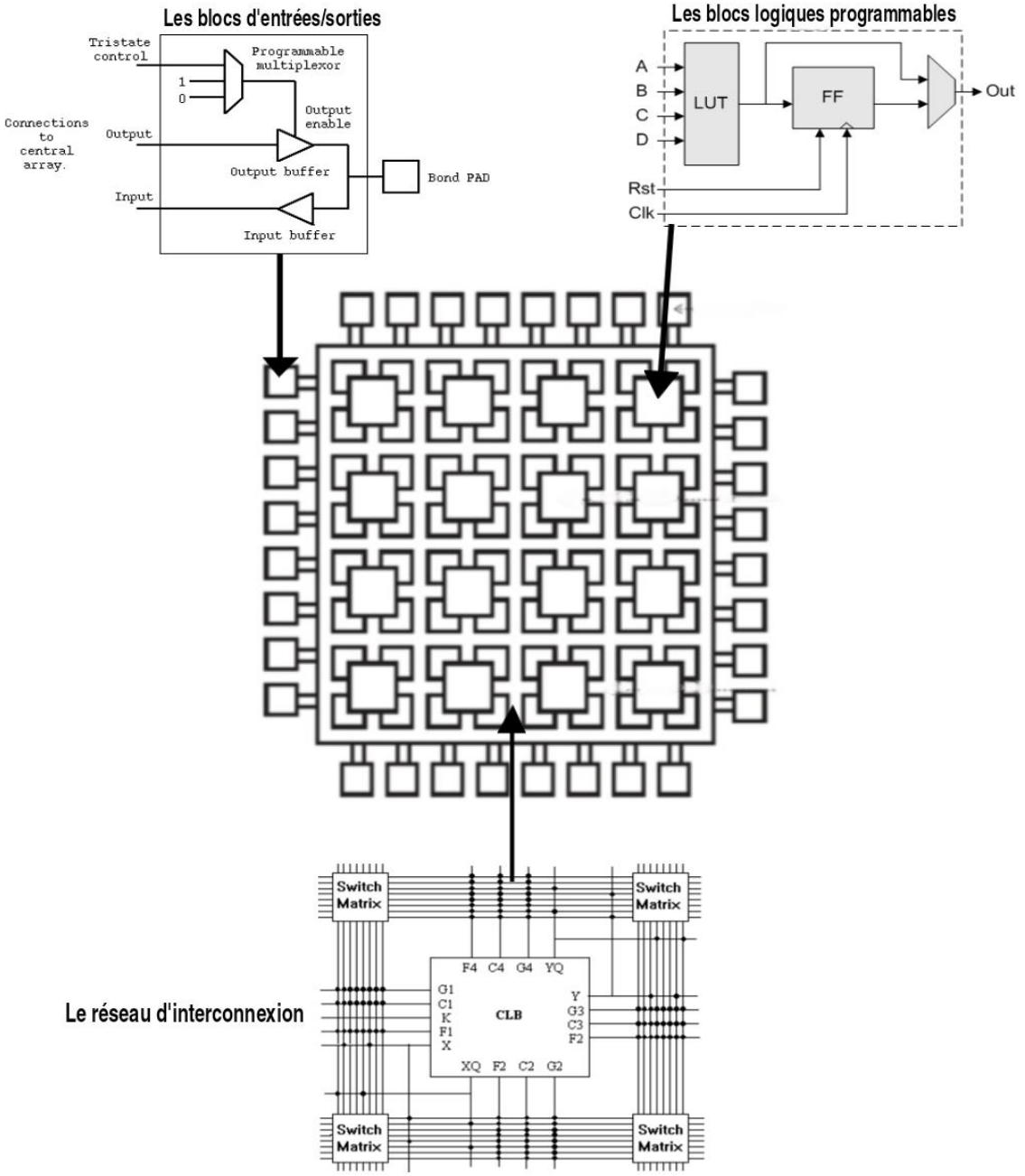


Figure V.2 : Architecture d'un carte FPGA.**[20]**

V.4 Logiciel vivado 2016.4

Vivado 2016.4 est une version de la suite de développement proposée par Xilinx pour la conception, la simulation, la synthèse et l'implémentation de circuits logiques sur FPGA. Ce logiciel permet de concevoir des systèmes numériques en langage HDL (comme VHDL ou Verilog), de les simuler, puis de les déployer sur les FPGA Xilinx.

Vivado 2016.4 intègre également des outils pour l'analyse temporelle, la gestion des contraintes, la génération de blocs IP, et l'optimisation des performances. Il remplace progressivement l'ancien outil ISE de Xilinx, en apportant une interface plus moderne et un moteur de synthèse plus performant.

V.4.1 L'outils de vivado

L'interface extérieur de vivado représente dans le figure V.3 :

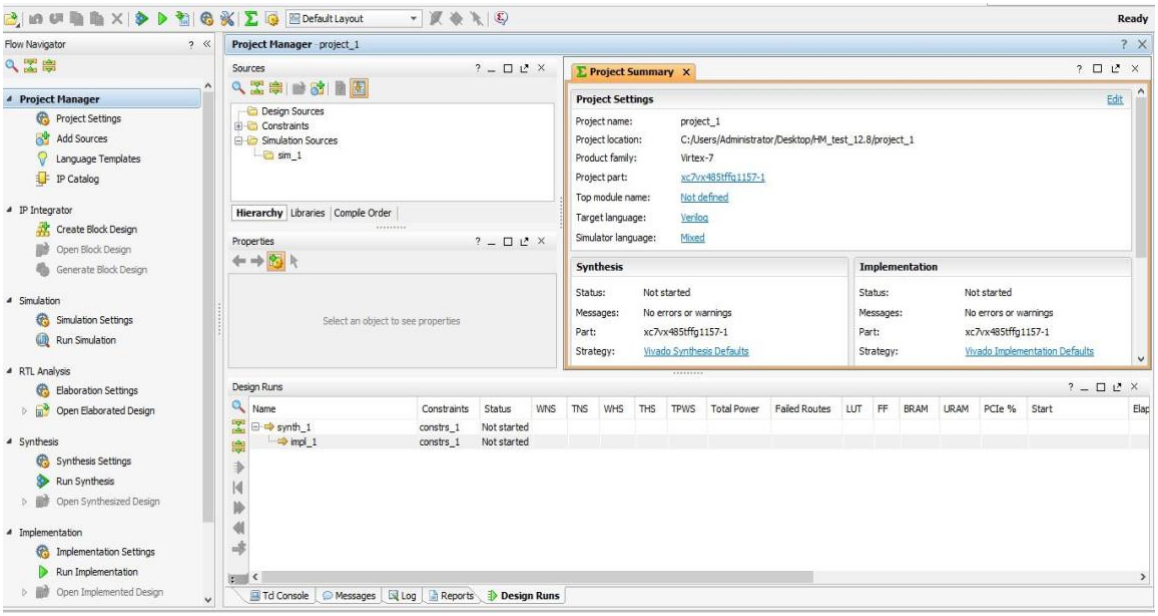


Figure V.3 : L'outils de vivado

1. **Project Manager**

Gère les paramètres et les sources de projet.

Project Settings : Modifier les paramètres du projet (nom, langage, FPGA cible, etc).
Add Sources : Ajouter des fichiers de code source, contraintes (XDC), ou de simulation.

2. **IP Integrator**

Pour créer des designs en mode graphique (block design)
Create Block Design : Créer un nouveau design basé sur des blocs IP.
Open Block Design : Ouvrir un design existant.

Generate Block Design : Génère les fichiers HDL à partir d'un schéma de blocs.

3. **Simulation**

Permet de configurer et lancer la simulation du design.

Simulation Settings : Configurer la simulation (fichier de testbench, options de simulation, etc).
Run Simulation : Lancer la simulation et voir les signaux avec un visualiseur (Waveform).

4. **RTL Analysis**

Analyse de votre design écrit en HDL avant la synthèse.

Elaboration Settings : Configurer l'élaboration (analyse statique du code).
Open Elaborated Design : Visualiser le design RTL avec ses blocs, connexions, et hiérarchie.

5. **Synthesis**

Pour convertir le code RTL en une structure logique (netlist).

Synthesis Settings : Configurer la synthèse (options d'optimisation, timing, etc.).
Run Synthesis : Lancer la synthèse du design.
Open Synthesized Design : Visualiser le netlist synthétisé et faire une première analyse de timing.

6. **Implementation**

Pour placer et router le design sur le FPGA cible.

Implementation Settings : Configurer les options de placement/routage.
Run Implementation : Lancer l'implémentation.
Open Implemented Design : Analyser les résultats de placement, routage, et timing final.

V.4.2 Création de projet

Pour créer un projet, il est nécessaire de passer par ces étapes :

Après l'overture de logiciel vivade on cliqué sur " **Create new project** " :

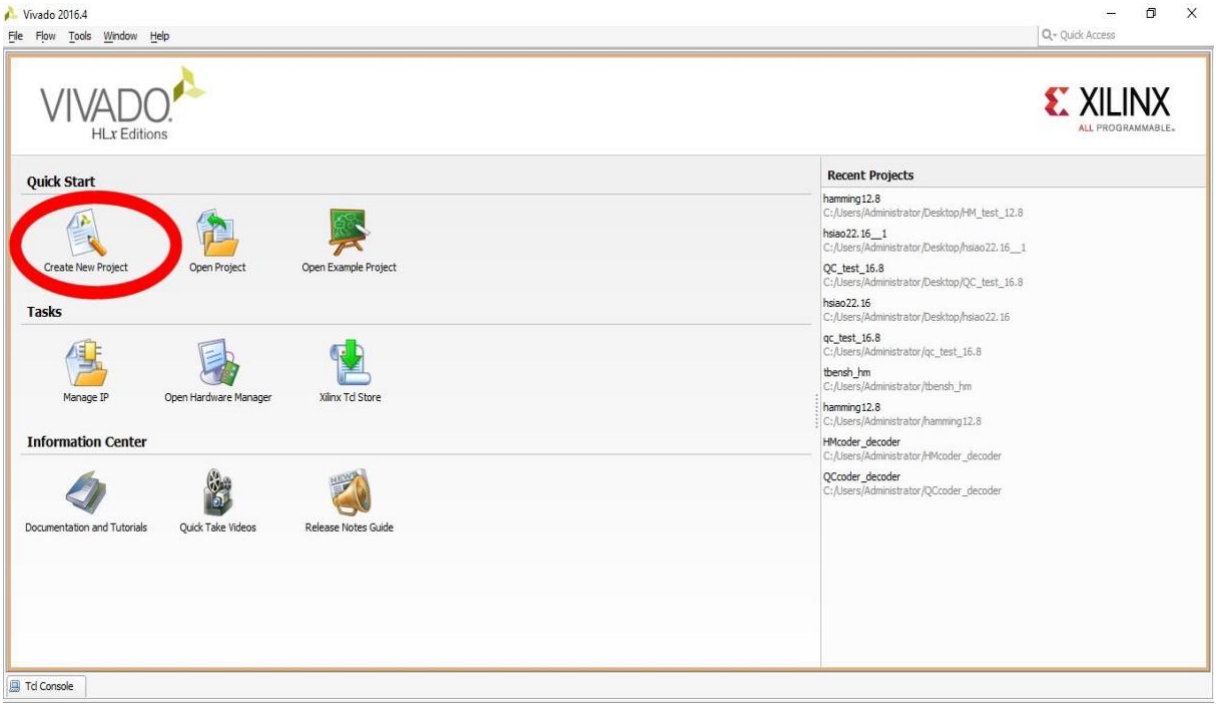


Figure V.4 : fenêtre pour créer un nouveau projet

Ensuite on donne un nom pour notre projet :

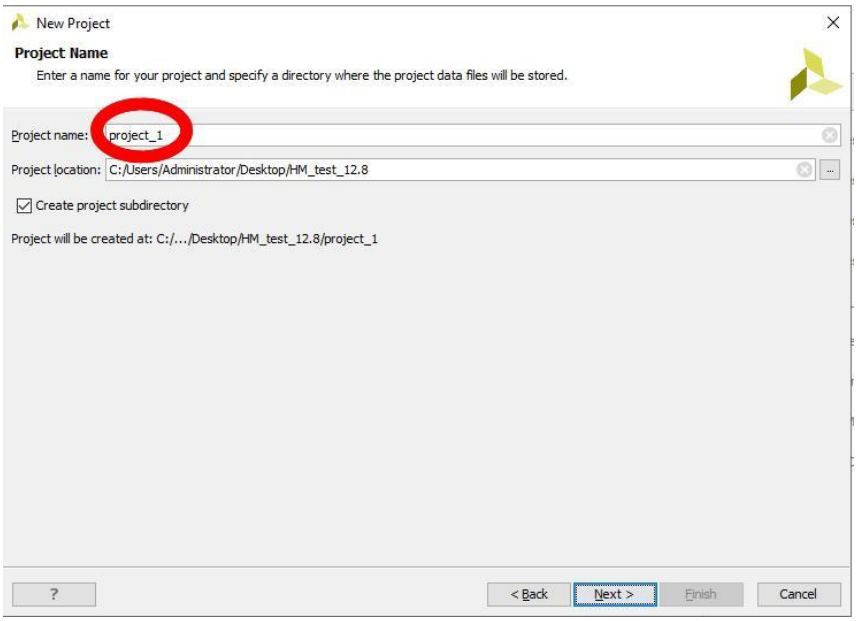


Figure V.5 : fenêtre pour le nom de projet

Pour l'étape suivent on sélectionné le type de projet :

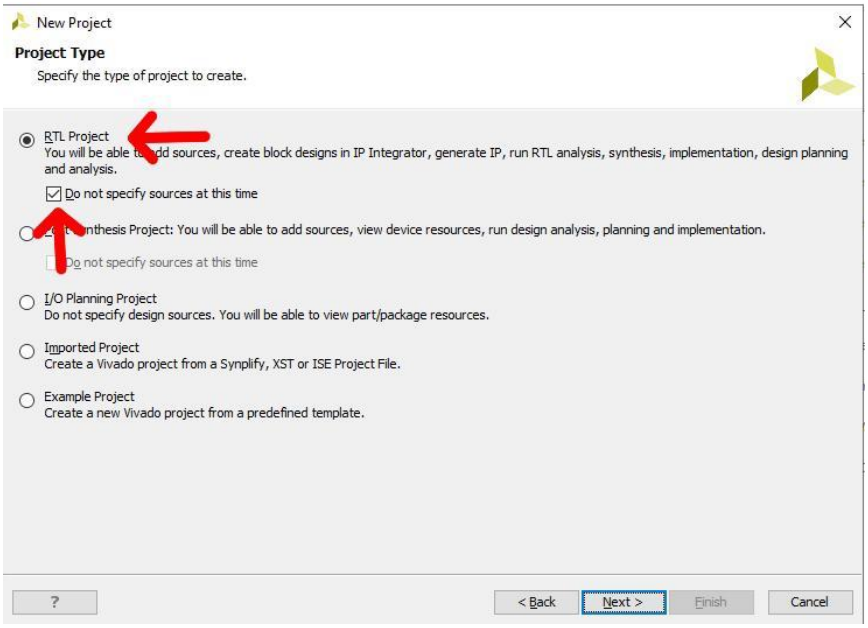


Figure V.6 : fenêtre des types des projets

Finalement, en sélectionné le “ **part or board** ”pour notre projet :

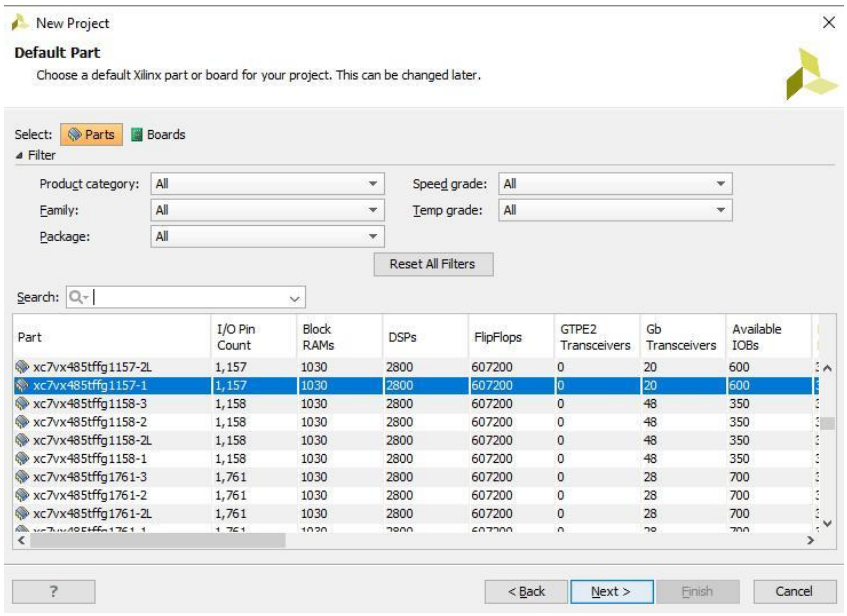


Figure V.7 : fenêtre des parts et boards

V.4.3 Ajouter des sources

Les étapes pour ajouter un source de code :

Premièrement, clique sur “ add sources ” dans la fenêtre de project manager :

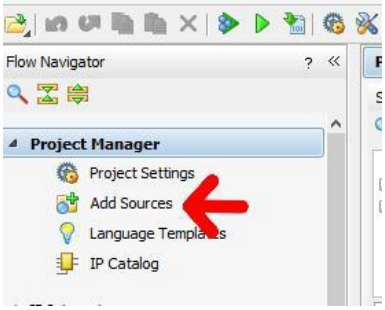


Figure V.8 : fenêtre de projet manager

Deuxième, dans la fenêtre des sources en sélectionné design pour le code principal ou la simulation pour le testbensh :

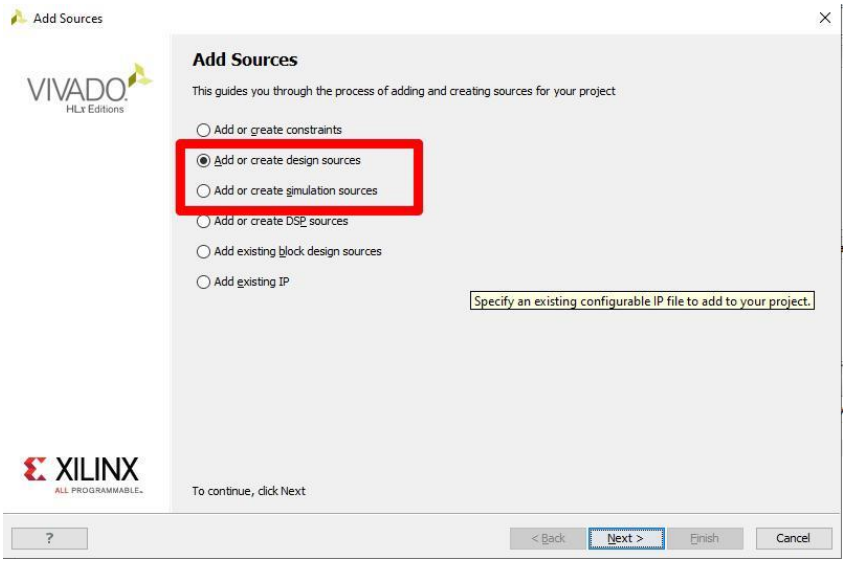


Figure V.9 : fenêtre pour choisir le type de source

Ensuite, Sélectionne un type de projet VHDL ou verilog et donne un nom pour le source et choisir un localisation four le fichier :

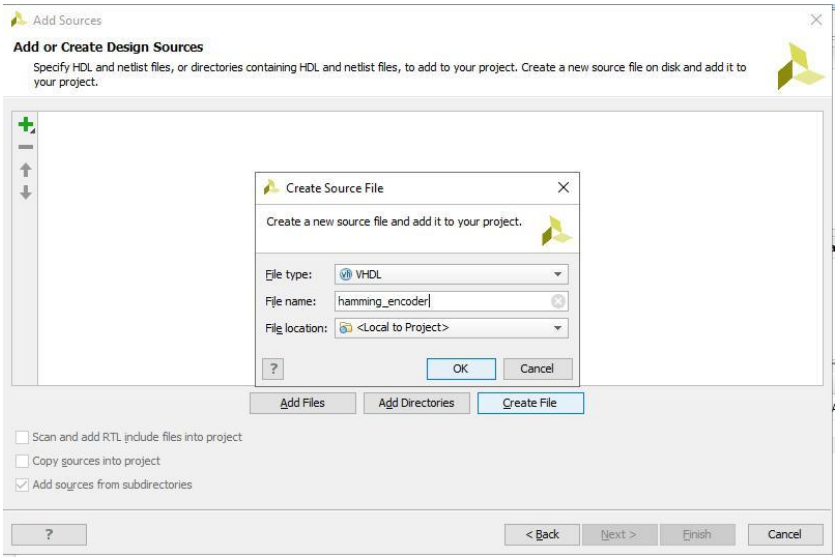


Figure V.10 : Fenêtre de créer un source code

Finalement, en trouver l'espace de code (VHDL) pour modifier ou copier le code de notre projet :

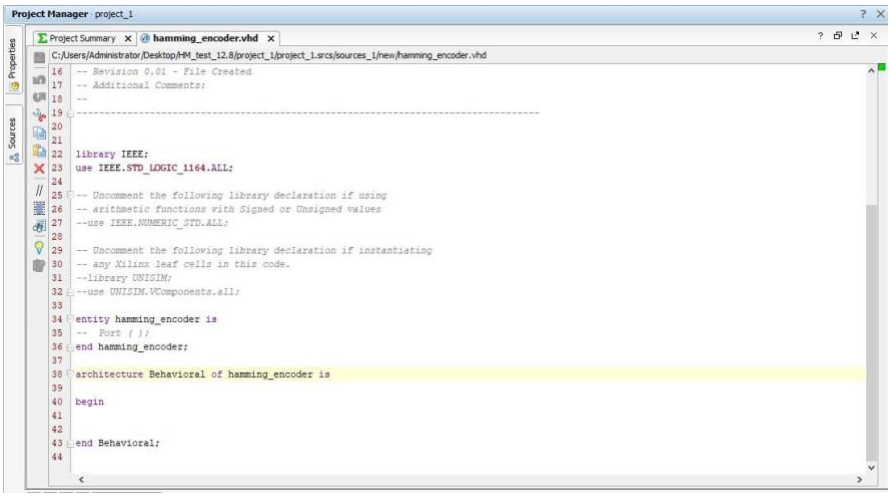


Figure V.11 : L'espace de code vhd

V.5 Simulation & RTL schématique

Nous avons simulé ces trois codes dans Vivado et les résultats sont les suivants :

V.5.1 Code de hamming (12.8)

1. **Simulation :**

Dans le hamming (12.8) en trouver 2 cas sur la simulation, pour un seul erreur et deux erreurs :

- **Un seul erreur :** le figure V.12 représente le résultat de simulation pour 1 erreur dans le mot de code :

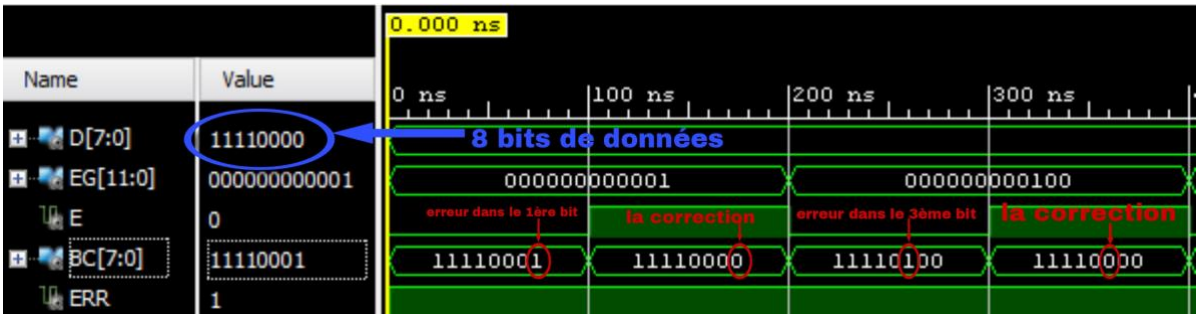


Figure V.12 : simulation de hamming (12.8) présente l'erreur et le corrigé.

Les bits de données nous entrons sont **D[7 :0]** Ce qui revient à **(11110000),** Nous avons mis l'erreur dans la 1ère position ce qui est **EG[11 :0]** l'erreur vecteur, maintenant le nouveau mot de code est **(11110001)** s'est pour **E** égal à 0 ,Cela signifie qu'il n'y a pas encore de correction mais après le activation la correction **E = 1**, le mot de code sortie corriger **BC[7 :0]** sans erreur.

- **Deux erreurs :** le figure V.13 représente le cas de 2 erreurs effectuées dans le mot de code entré.

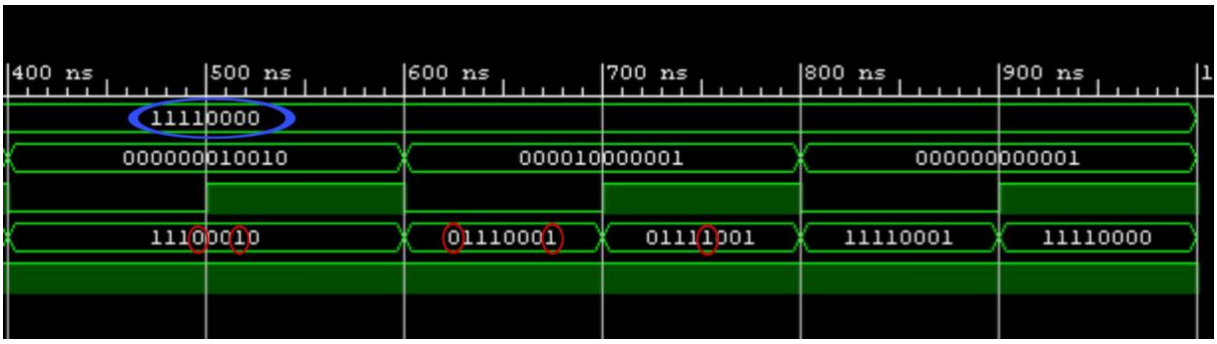


Figure V.13 : simulation de hamming (12.8) présente deux erreurs et le corrigé.

Nous avons mis les erreurs dans les positions 2ème et 5ème bits et le mot de code sortie est **(11100010)** sans correction Parce qu'il a découvert deux erreurs et en calculant le syndrome d'erreur, il a découvert qu'il n'existe pas.

Pour le deuxième cas, nous mettons les erreurs dans les positions du premier et du huitième bit et le mot de code résultant est **(01111001)**, attendez ! La position du bit corrigée est le 4ème bit, pourquoi ? Car en calculant le syndrome, il a découvert que l'erreur du syndrome remonte au quatrième bit.

Finalement, On peut dire que le code de Hamming ne peut pas corriger deux erreurs en même temps, mais seulement les détecter.

2. RTL schématique :

En général, rtl schématique c'est un des blocs contient des booléens de logique combinatoire dans ce cas nous avons trois blocs de base représente dans le figure V.14, chacun ayant sa propre fonction. Nous les afficherons tous ci-dessous :

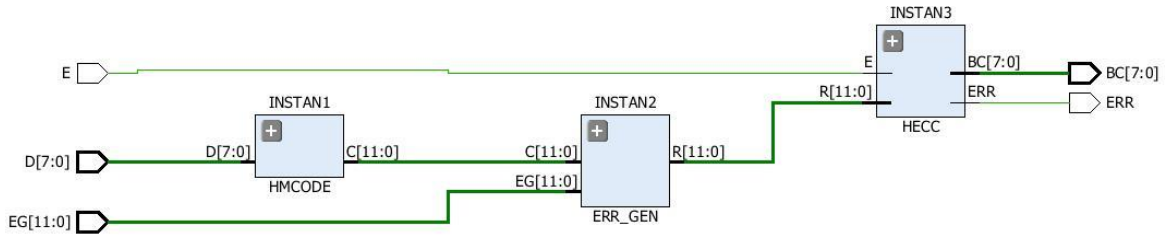


Figure V.14 : RTL schématique présente trois blocs de hammnig (12.8)

- **Bloc de codage :** ce bloc contient des booléens XOR pour le codage , les bits de données **D[7 :0]** entré le bloc et sortir un mot m de code **C[11 :0]** comme représenté dans le figure V.15.

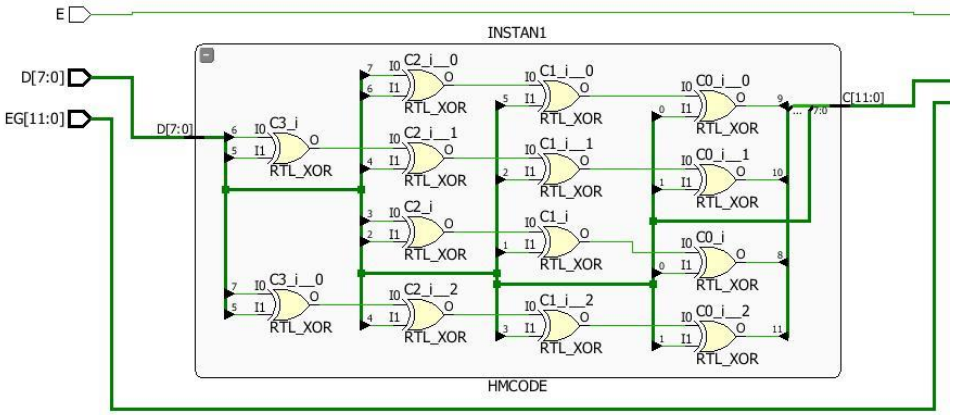


Figure V.15 : bloc de codage hamming (12.8)

- **Bloc de génération d'erreurs :** ce bloc génération (figure V.16) l'erreur par addition erreur vecteur **EG[11 :0]** avec le mot de code **C[11 :0]** et la sortir en trouve le mot code erroné **R[11 :0]** contient un erreur.

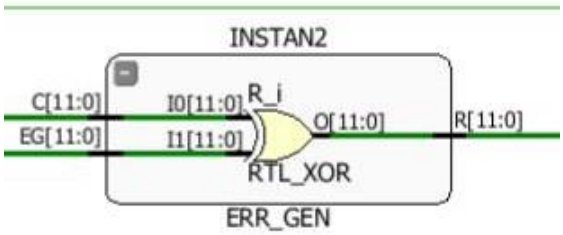


Figure V.16 : bloc de génération l'erreur pour hamming (12.8)

- **Bloc de décodage :** ce bloc (figure V.17) corriger l'erreur par calculer le syndrome à l'aide de mot code **R[11 :0]** avec des opérations XOR, ensuite le syndrome traverse dans **LUT** choisir l'erreur vecteur correspondant à le syndrome , finalement l'erreur vecteur sortie de **LUT** et un processus est effectué par l'opération XOR pour corriger l'erreur partout où elle existe, pour produire le mot correct, qui est symbolisé par **BC[7 :0].**

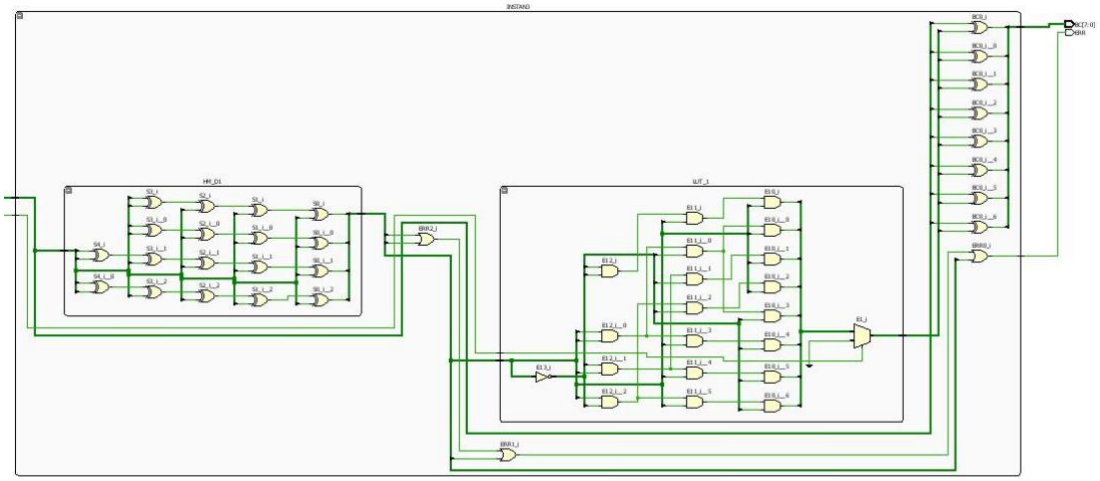


Figure V.17 : bloc de décodage pour hamming (12.8)

V.5.2 code Quasi-cyclique (16.8)

1. **Simulation :** le code de quasi cyclique, il a la capacité de détecter jusqu'à 4 bits et corriger 2 bits ,donc en trouver trois cas sur la simulation de 1 et 2 et 3 bits, présenté dans le figure V.18.

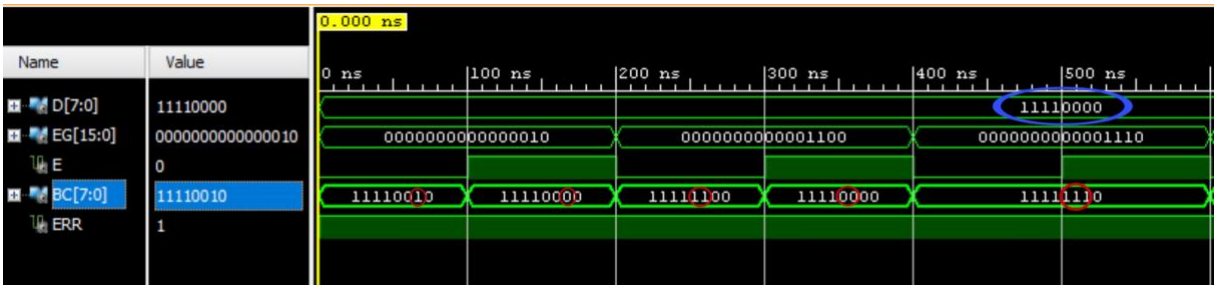


Figure V.18 : simulation de Quasi-cyclique (16.8) présenté trois cas des erreurs et le corrigé

Pour la premier cas d'un seul erreur, nous avons mis un erreur pour la donné **(11110000)** dans le deuxième bit s'est que devenir **(11110010)** , après l'activation de la correction **E** on trouvons l'original mot de code.

Pour la deuxième cas de 2 erreurs, nous avons mis deux erreurs dans les positions 3ème et 4ème bits qui donne **(11111100)** , après l'activation de la correction **E** on trouvons l'original mot de code.

Pour la troisième cas de 3 erreurs, nous avons mis trois erreurs dans les positions 1ère et 2ème et 3ème bits qui donne **(11111110)** , après l'activation de la Correction **E** Nous constatons que cela n'a pas été corrigé car il ne peut pas corriger trois ou quatre erreurs, mais seulement le détecter.

2. **RTL schématique :** le figure V.19 présente trois blocs de RTL.

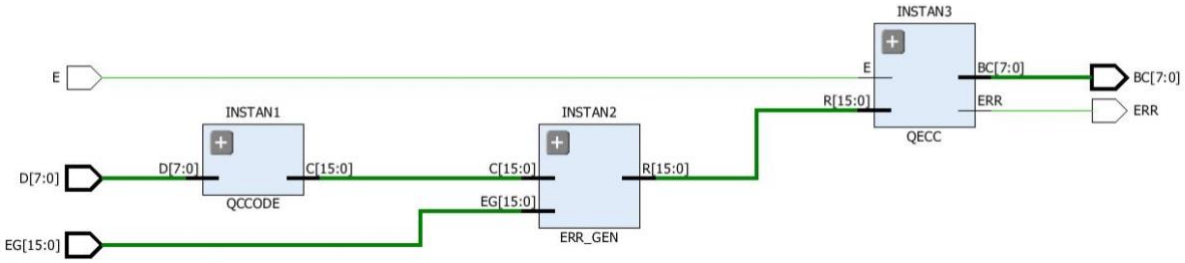


Figure V.19 : les blocs de Quasi-cyclique (16.8)

- **Bloc de codage :** pour le codage section (figure 5.20), la données **D[7 :0]** entré dans les booléens de XOR se qui donne **C[15 :0]** le mot de code dans la sortie.

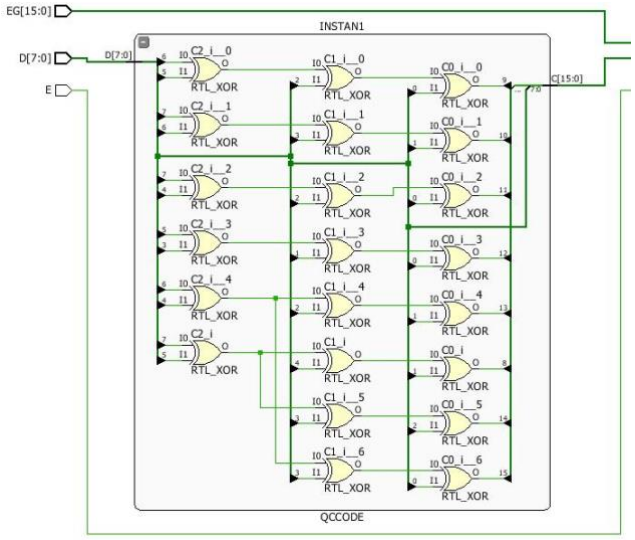


Figure V.20 : bloc de codage pour Quasi-cyclique (16.8)

- **Bloc de génération l'erreur :** le figure V.21 présenté l'opération XOR de **EG[15 :0]** l'erreur vecteur avec **C[15 :0]** le mot de code ,qui donne le mot code erroné **R[15 :0]**.

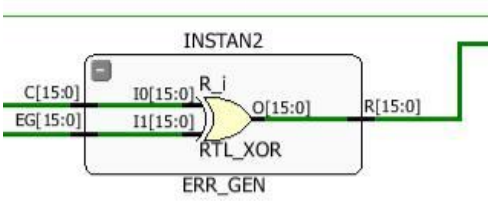


Figure V.21 : bloc de l'erreur génération pour Quasi-cyclique (16.8)

- **Bloc de décodage :** ce bloc (figure V.22) corriger l'erreur par calculer le syndrome à l'aide de mot code **R[15 :0]** avec des opérations XOR, ensuite le syndrome traverse dans **LUT** pour choisir l'erreur vecteur correspondant à le syndrome, finalement l'erreur vecteur sortie de **LUT** et un processus est effectué par l'opération XOR pour corriger l'erreur partout où elle existe, pour produire le mot correct, qui est symbolisé par **BC[7 :0]**.

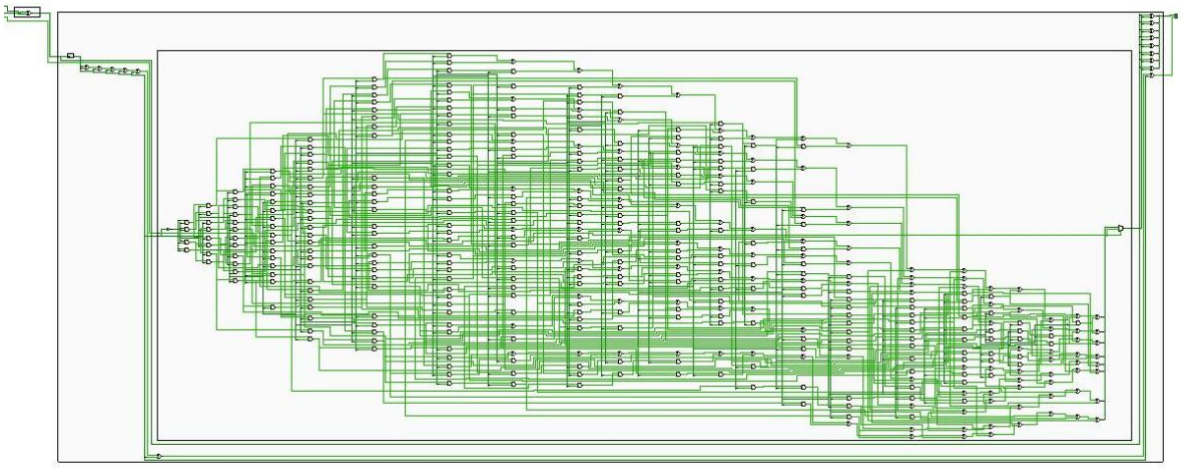


Figure V.22 : bloc de décodage pour le Quasi-cyclique (16.8)

V.5.3 Code hsiao (22.16)

1. **Simulation :** Ce code détecter deux erreurs et corriger un seul erreur, en trouvons deux cas de l'erreurs :

- **Un seul erreur :** Le figure V.23 présenté la génération d'un seul erreur et la correction.

Figure V.23 : simulation de hsiao (22.16) présente l'erreur et le corriger

Pour ce cas , en fait un erreur dans la position 6ème qui donne le mot erroné, après l'activation de la correction **E** en trouvons le mot de code corriger, ensuite en mis l'erreur dans la position 13ème et le corrigé normal.

- **Deux erreurs :** Le figure V.24 présenté la génération de deux erreurs dans le mot de code.

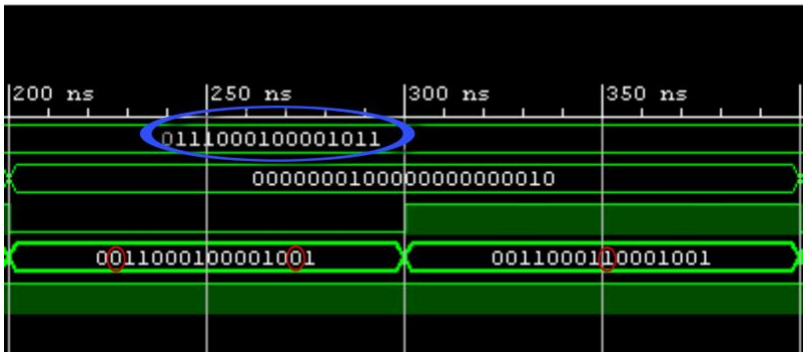


Figure V.24 : Simulation de hsiao (22.16) dans le cas de deux erreurs

Pour cette cas de 2 erreur, nous mettons les erreurs dans les positions du 2ème et du 15ème bit, après l'acrivation de la correction **E** La position du bit corrigée est le 8ème bit, Car en calculant le syndrome, il a découvert que l'erreur du syndrome remonte au 8ème bit.

On peut dire que le code de hsiao (22.16) ne peut pas corriger deux erreurs en même temps, mais seulement le détecter.

2. **RTL schématique :** pour le schématique de code hsiao (22.16) en trouvons ainsi trois blocs présenté danse figure V.25.

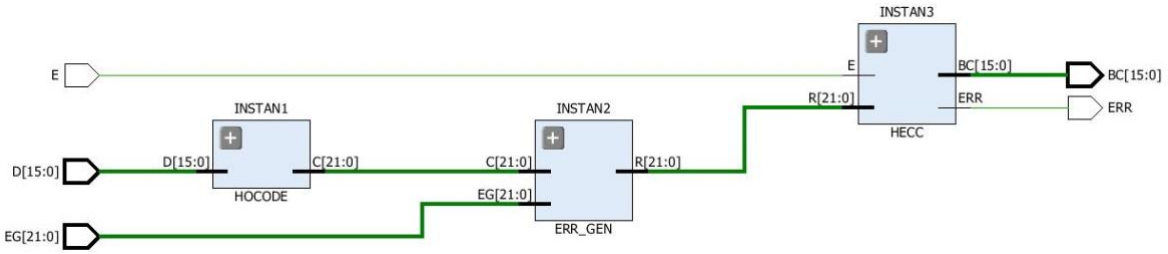


Figure V.25 : les blocs RTL de code hsiao (22.16)

- **Bloc de codage :** les bits de données **D[15 :0]** entré le bloc de codage qui contient des portes XOR, et dans la sortie nous trouvons le mot de code **C[21 :0]** (figure V.26).

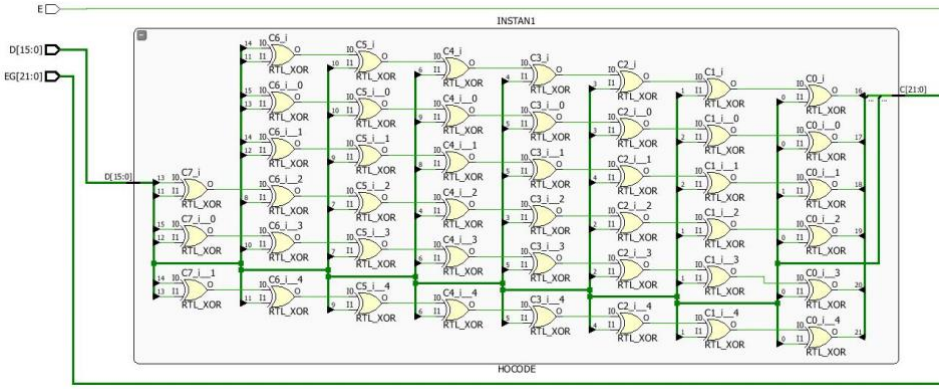


Figure V.26 : Bloc de codage pour le code hsiao (22.16)

- **Bloc de l'erreur génération :** ce bloc (figure V.27) fait l'opértaion XOR pour **EG[21 :0]** l'erreur génération et le mot de code **C[21 :0]** , qui donne le mot code erroné**R[21 :0].**

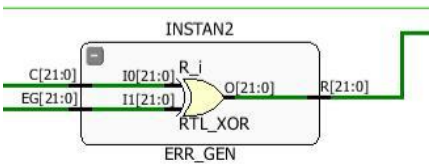


Figure V.27 : bloc de l'erreur génération pour le code hsiao (22 :16)

Bloc de décodage : ce bloc (figure V.28) corriger l'erreur par calculer le syndrome à l'aide de mot code **R[21 :0]**, ensuite le syndrome traverse dans **LUT** pour choisir l'erreur vecteur correspondant à le syndrome, finalement un processus est effectué par l'opération XOR pour corriger l'erreur **BC[15 :0]**.

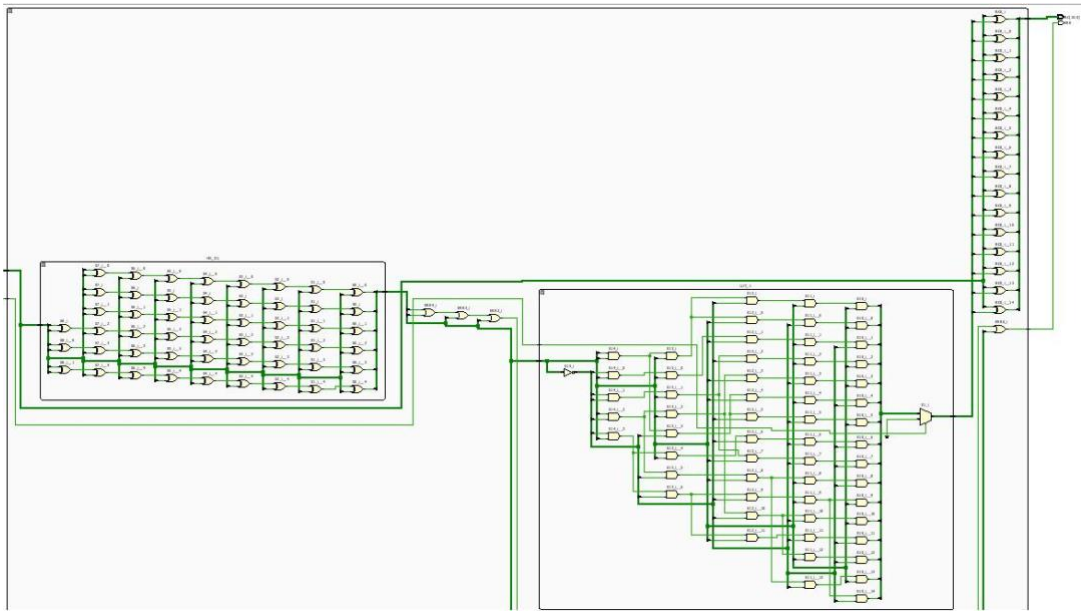


Figure V.28 : bloc de décodage pour le code hsiao (22.16)

V.6 Discussion des résultats

Après avoir effectué des simulations et conçu des schémas RTL pour chacun des codes de correction hamming (12,8), hsiao (22,16) et quasi-cyclique (16,8), nous avons analyser les performances structurelles et fonctionnelles de chaque code.

Le code de hamming présente une structure simple et un nombre limité de portes logiques, ce qui le rend adapté aux applications nécessitant une correction d'erreur unique et une détection d'erreur double avec une faible surcharge logique.

Le code de hsiao offrait une amélioration du nombre de bits protégés et la même capacité SEC-DED, mais montrait une plus grande complexité structurelle dans le schéma RTL en raison de la distribution des bits de parité d'une manière qui réduisait la consommation d'énergie et le nombre de permutations.

D'autre part, le code quasi-cyclique a montré une complexité structurelle importante par rapport aux deux codes précédents, mais il a fourni de meilleures performances en termes de capacité à corriger de multiples erreurs grâce à sa nature matricielle et à sa structure cyclique partielle, le rendant adapté aux systèmes critiques dans l'espace ou les communications. Il apparaît donc que le choix du code de correction dépend principalement des exigences de l'application en termes d'espace, de puissance de calcul et du nombre d'erreurs à corrige.

Conclusion général

Au terme de ces travaux, nous avons abordé de manière approfondie les défis liés à la fiabilité des systèmes embarqués dans les microsatellites, notamment face aux effets délétères des rayonnements ionisants en orbite. Ces rayonnements peuvent engendrer des erreurs transitoires telles que les SEU (Single Event Upsets), affectant notamment les mémoires SRAM et les circuits logiques. La compréhension des différents types d'orbites, de sous-systèmes critiques et des zones de vulnérabilité a permis de cerner avec précision les besoins en mécanismes de protection contre ces perturbations.

Dans ce cadre, l'implémentation de codes correcteurs d'erreurs s'est révélée indispensable. Trois types de codes ont été étudiés et comparés :

Le code de Hamming, simple à mettre en œuvre, permet la correction d'une erreur et la détection de deux, tout en demandant peu de ressources.

Le code de Hsiao, dérivé du code de Hamming, est optimisé pour réduire la complexité matérielle, notamment grâce à un faible nombre de bits à 1 dans sa matrice de parité, ce qui diminue le nombre de portes logiques nécessaires.

Le code quasi-cyclique (QC), plus complexe, permet des niveaux de protection bien plus élevés, notamment en environnement fortement irradié, grâce à sa capacité à corriger plusieurs erreurs avec une structure matricielle régulière favorable à une implantation matérielle efficace.

Ces codes ont été implémentés en VHDL, simulés et vérifiés à l'aide de l'environnement Vivado. Les résultats obtenus, notamment l'analyse des diagrammes RTL et la synthèse FPGA, ont permis d'évaluer la consommation de ressources logiques, la latence et la faisabilité physique de chaque solution.

En fin, une perspective intéressante consiste à étendre le code Hsiao (22,16) vers des versions de plus grande dimension, telles que Hsiao (38,32), afin de l'adapter aux architectures 32 bits ou 64 bits plus courantes dans les systèmes embarqués récents. Cette évolution permettrait de protéger des blocs mémoire plus larges tout en conservant les avantages du code Hsiao en matière de faible complexité matérielle. Par ailleurs, il serait pertinent de comparer ces variantes à des codes plus puissants comme les SEC-DED-DAEC, capables de corriger deux erreurs adjacentes, une configuration souvent rencontrée dans l'environnement spatial à forte densité de particules.

Références

[1] Vincent pouget, «Simulation Experimentale Par Impulsions Laser Ultra-Courtes Des Effets Des Radiations Ionisantes Sur Les Circuits Integres.», theses.fr,p 20-24,2000.

[2] Philippe CHEYNET, «Etude de la robustesse du contrôle intelligent face aux fautes induites par les radiations», theses.hal.science, p20, 1999.

[3] Andréas, «Single Event Effects – The Achilles heel of modern aerospace electronics»,https://www.engineeringpilot.com/post/single-event-effects-the-achilles-heel-of-modern-aerospace-electronics, 2021.

[4] Damien leroy, «Étude Des Modes De Perturbation Et De Susceptibilité Des Circuits Numériques Aux Collisions De Particules Et Aux Attaques Laser.», http://docnum.univlorraine.fr/public/UPVM/Theses/2006/Leroy.Damien.SMZ0628.pdf, p 41-46,2006.

[5] Ygor Quadros de Aguiar, «Single-Event Effects, from Space to Accelerator Environments.», https://library.oapen.org/handle/20.500.12657/94673, p 31-32,2025.

[6] Yann clavet, «Définition de solutions de filtrage planaires et multicouches pour les nouvelles générations de satellites de télécommunications», theses.hal.science, p 10-11, 2006.

[7] Deal guyane, «L'observation de la Terre par satellite», https://www.applisat.fr/generalites-satellites/observation-de-la-terre-par-satellite, p 1.

[8] Thomas estier, «nanosatellite», thomas.estier.net, p 10-11, 1999.

[9] Gary quinsac, sciences pour exoplanètes et systèmes planétaires, sesp.esep.pro.

[10] Louis J. Ippolito, Jr, «Satellite Communications Systems Engineering», p 44-45, 2008.

[11] Sebastien De Dijcker, «Implémentation de la gestion des télécommandes et des télémétries au sein de l'ordinateur de bord du nanosatellite OUFTI-1», p 10, 2011.

[12] André MONTPETIT, «note sur la notion d'equivalence entre deux codes lineaires», p 178-184, 1986.

[13] Richard W. Hamming, «Error detecting and error correcting codes», p 4-8, 1950.

[14] Vipin Balyan, caleb hillier, «error detection and correction on-board nanosatellites usinghamming codes», p 5-7 , 2019.

[15] Todd k. Moon, «Error correction coding : mathematical methods and algorithms», p 130-158, wiley 2021

[16] mong sim, yanyan zhuang, «design of two interleaved error detection and corrections using hsiao code and crc», researchgate.net, p 2, 2020.
[17] Y. bentoutou, «a real time edac system for applications onboard earth observation small satellites», 2011.

[18] Sadia Ahmad, Minahil Zahra, Salma Zainab Farooq, Adnan Zafar, «comparison of edac schemes for ddr memory in space applications», 2013.

[19] Hervé Le Provost, «Initiation au langage VHDL», p 3-15, 2008.

[20] stephen brown, jonathan rose, «FPGA and CPLD Architectures : A Tutorial», 1996.