LABORATOIRE INTERFACES CAPTEURS
ET MICRO-ÉLECTRONIQUE

École Doctorale IAEM - Lorraine
Département de Formation Doctorale Électronique - Électrotechnique

THÈSE DE DOCTORAT

Présentée pour obtenir le grade de docteur de
l'Université Paul Verlaine - Metz

Discipline : Systèmes Électroniques
Spécialité : Microélectronique

CONCEPTION ARCHITECTURALE HAUT DÉBIT ET SÛRE DE FONCTIONNEMENT POUR LES CODES CORRECTEURS D'ERREURS

Par

HOUSSEIN JABER

Soutenu le 9 décembre 2009 devant le jury composé de :

MICHAEL NICOLAIDIS	Directeur de Recherche au CNRS, TIMA	Rapporteur
MOHAMAD SAWAN	Pr., École Polytechnique de Montréal	Rapporteur
FRANCIS BRAUN	Pr., Université Louis Pasteur de Strasbourg	Examinateur
AHMED BOURIDANE	Pr., Université de Newcastle	Examinateur
FABRICE MONTEIRO	Pr., Université Paul Verlaine de Metz	Examinateur
STANISLAW J. PIESTRAK	Pr., Université Paul Verlaine de Metz	Co-directeur de thèse
ABBAS DANDACHE	Pr., Université Paul Verlaine de Metz	Directeur de thèse

LICM - 07 Rue Marconi, Technopôle, 57070 Metz, France
Tel : +33 (0)3 87 31 56 57 - Fax : +33 (0)3 87 54 73 07 - www.licm.fr

JE DÉDIE CE MÉMOIRE A :

MES CHERS PARENTS

&

MA FAMILLE JABER.

Remerciements

Les travaux présentés dans ce manuscrit ont été réalisés au sein du laboratoire LICM (Laboratoire Interface, Capteurs et Microélectroniques). Je remercie Monsieur Abbas DANDACHE, Professeur et Directeur du LICM, de m'avoir accueilli au sein de son laboratoire.

Je tiens également à remercier Messieurs Stanislaw PIESTRAK et Fabrice MONTEIRO, Professeurs au laboratoire LICM, pour m'avoir encadré mes travaux ainsi que leurs conseils avisés et tout le temps qu'ils m'ont consacré lors de ce travail.

Je remercie vivement Monsieur Michael NICOLAIDIS, Directeurs de Recherche au CNRS, TIMA-Grenoble, et Monsieur Mohamad SAWAN, Professeur à l'École Polytechnique de Montréal, pour l'honneur qu'ils m'ont fait et pour l'intérêt qu'ils ont porté à mes travaux en acceptant d'être les rapporteurs de ce mémoire de thèse.

Je voudrais remercier Monsieur Francis BRAUN, Professeur à l'Université Louis Pasteur à Strasbourg, pour avoir accepté d'être le président de ce jury. Je voudrais aussi remercier Monsieur Ahmed BOURIDANE, Professeur de l'Université de Newcastle, d'avoir accepté d'être membre du jury de thèse.

Merci aussi à tous mes collègues et amis de longue date du laboratoire. Je leur exprime ma profonde sympathie et leur souhaite beaucoup de bien. En particulier je cite : Abbas, Ali, Ahmad, Fadel, Farah, Jeannot, Hassan, Kinda, Mokhtar, Mohamad, Mohsin, Mazen, Mehdi, Nicole, Nassima, Ibrahim, Wael, ...

Table des matières

INTRODUCTION GÉNÉRALE **7**

I. ETAT DE L'ART ET ÉTUDE THÉORIQUE **13**

1 Codes correcteurs d'erreurs **13**

1.1 Introduction 13
1.2 Chaîne de transmission 14
1.2.1 Performance d'un système de transmission : gain de codage 16
1.3 Propriétés et architectures séries des codes 18
1.3.1 Techniques de codage 18
1.3.2 Codes en blocs 18
1.3.3 Codes convolutifs 30
1.3.4 Codes concaténés 36
1.3.5 Turbo-codes 36
1.4 Architectures parallèle-pipeline 37
1.4.1 Codeur parallèle haut débit pour les codes CRC 37
1.4.2 Décodeur parallèle-pipeline pour les codes cycliques en blocs 38
1.4.3 Achitecture de décodeur RS rapide et à faible complexité 40
1.4.4 Codeur parallèle haut débit pour les codes convolutifs 41
1.5 Conclusion 42

2 Sûreté de fonctionnement et tolérance aux fautes **45**

2.1 Introduction 45
2.2 Concepts de base 45
2.2.1 Service d'un système 46
2.2.2 Sûreté de fonctionnement 46
2.3 Autres propriétés de la SdF 50
2.4 Défaillance du service 51
2.5 Méthodologie d'une étude de la SdF 52
2.5.1 Solutions de la SdF aux systèmes embarqués 53
2.6 Tolérance aux fautes, principes et mécanismes 55

2.7 Principe de la tolérance aux fautes 55
2.7.1 Détection d'erreurs 56
2.7.2 Recouvrement 61
2.7.3 Correction d'erreurs 61
2.8 Circuits auto-contrôlables 63
2.8.1 Circuits sûrs en présence de fautes (*Fault-Secure*) 63
2.8.2 Circuits auto-testables (*Self-Testing*) 64
2.8.3 Circuits totalement auto-contrôlables (*Totally Self-Checking*) 65
2.8.4 Circuits fortement sûrs en présence de fautes (*Strongly Fault-Secure*) 65
2.9 Évaluation de la SdF 65
2.9.1 Modèles de fautes 66
2.9.2 Injection de fautes 70
2.10 Conclusion 74

II. ETUDE ARCHITECTURALE ET IMPLANTATION 77

3 Codeur convolutif récursif haut débit 77
3.1 Introduction 77
3.2 Architectures parallèles-pipeline 77
3.2.1 Principe des techniques parallèles-pipeline 78
3.2.2 Inconvénients des architectures existantes 80
3.3 Étude du codeur convolutif haut débit 81
3.3.1 Présentation du codeur convolutif MTO haut débit 82
3.3.2 Présentation du codeur convolutif OTM haut débit 83
3.4 Implantation du codeur convolutif haut débit 83
3.4.1 Implantation du codeur haut débit de type MTO (CP_{mto}) 85
3.4.2 Implantation du codeur haut débit de type OTM (CP_{otm}) 86
3.5 Résultats expérimentaux 88
3.5.1 Performances du codeur CP_{mto} 89
3.5.2 Performances du codeur CP_{otm} 92
3.6 Applications aux filtres récursifs RII 95
3.6.1 Architectures séries d'un filtre RII 96
3.6.2 Architectures parallèle-pipeline d'un filtre RII 98
3.6.3 Résultats expérimentaux 101
3.7 Conclusion 102

4 Méthodologie de conception de codeurs en bloc SdF 105
4.1 Introduction 105
4.2 Introduction de la méthodologie 105

4.3 Élaboration de la méthodologie 109
4.3.1 Schéma général d'un codeur FS 110
4.4 Implantations de codeurs FS parallèles 112
4.4.1 Algorithme de codage parallèle 113
4.4.2 Autre propriété de détection 116
4.4.3 Possibilités de simplification 117
4.5 Aspects fonctionnels et architecturaux 120
4.5.1 Analyse fonctionnelle et validation par injection d'erreurs 121
4.5.2 Aspect architectural 124
4.6 Implantation de codeurs FS parallèles-pipeline 129
4.6.1 Principe de réalisation d'un codeur FS parallèle-pipeline 129
4.6.2 Résultats expérimentaux 132
4.7 Décodeurs FS parallèle-pipeline 134
4.7.1 Architecture d'un décodeur FS parallèle-pipeline 135
4.7.2 Résultats expérimentaux 137
4.8 Conclusion 140

CONCLUSION GÉNÉRALE **143**

A Liste des acronymes **167**

B Liste des publications **171**

INTRODUCTION GÉNÉRALE

Introduction générale

Au cours des dix dernières années, la demande pour des systèmes de transmission numériques fiables s'est considérablement accrue [1]. L'explosion de l'échange d'informations, et les nouvelles possibilités offertes par le traitement numérique du signal ont accentué cette tendance.

Les systèmes de communication modernes exigent des débits de plus en plus élevés afin de traiter des volumes d'informations en augmentation constante. Ils doivent être flexibles pour pouvoir gérer des environnements multinormes, et évolutifs pour s'adapter aux normes futures. Pour ces systèmes, la qualité du service (QoS) doit être garantie et ce malgré l'évolution des technologies microélectroniques qui augmente la sensibilité des circuits intégrés aux perturbations externes (impact de particules, perte de l'intégrité du signal, etc). La tolérance aux fautes devient un critère important pour améliorer la qualité de service. Aux contraintes liées au traitement de l'information s'ajoute la nécessité de protéger les informations émises dans des environnements perturbés (par exemple erreurs de transmission) où traités dans des systèmes susceptibles d'être parasités par des fautes temporaires causées par les radiations cosmiques SEU (*Single Event Upset*) [2]. En effet, la qualité du service rendu en termes de communication se décline en deux mots clés : fiabilité et rapidité.

La qualité d'une transmission numérique dépend principalement de la probabilité d'occurence d'erreur dans les symboles transmis [3]. Cette probabilité étant fonction du rapport "*signal sur bruit*", une amélioration de la qualité de transmission peut être envisagée en augmentant la puissance d'émission et en diminuant le facteur de bruit du récepteur. Malheureusement, cette solution implique des coûts énergétiques et technologiques importants, ce qui en limite sensiblement l'emploi. Le contrôle des erreurs par codage est ainsi indispensable. L'utilisation de techniques de traitement numérique du signal, et notamment le codage des informations à transmettre, permet la détection et/ou la correction d'éventuelles erreurs de transmission. Comme ces techniques permettent de contrôler les erreurs induites par le bruit du canal de transmission, elles sont nommées "*codages de canal*". Parmi les principales techniques existantes, les codages en bloc et les codages convolutifs sont prédominants. Les codages en bloc sont utilisés notamment dans les réseaux Ethernet [53], dans les normes de transmission sans fils telles que bluetooth [16], et dans les normes de transmission HDTV (*High Definition TeleVision*) [18] et DVB-C (*Digital Video Broadcasting-Cable*) [19]. Le codage convolutif est très présent dans les systèmes de communication numérique sans fil [33–36].

La stratégie de base du codage consiste à ajouter une quantité contrôlée de redondance à la série d'informations à envoyer. La procédure de génération de redondance traite les informations, soit par blocs (codage en bloc) ou au contraire de manière continue (codage convolutif), soit comme entité

indépendante ou à l'inverse en tant que structure concaténée avec un autre code, soit plus récemment sous forme d'élément constituant dans un code LDPC (*Low Density Parity Check*) [46] ou turbo-code [42]. L'ajout de la redondance par le codeur permet au décodeur de détecter et de corriger le cas échéant un nombre fini d'erreurs de transmission. L'ensemble codeur/décodeur est considéré comme critique pour garantir le bon fonctionnement de la chaine de transmission. Le nombre d'erreurs affectant la transmission des informations dépend du moyen de transmission. Le débit des erreurs et leur distribution temporelle diffèrent selon que le moyen de transport est une ligne téléphonique, une ligne numérique, un lien satellite ou un canal de communication sans fil.

Il est dès lors évident que l'introduction de techniques de codage de canal induit une augmentation de la complexité du traitement numérique du système de communication. L'importance de cette augmentation est fonction du niveau de protection envisagé par l'opération de codage : une protection plus efficace contre les erreurs de transmission implique l'utilisation de méthodes de codage plus complexes, rendant par conséquent les procédures de décodage plus onéreuses.

Différents codes détecteurs d'erreurs EDC *(Error Detecting Codes)* ou codes correcteurs d'erreurs ECC *(Error Correcting Codes)* ont été utilisés pendant des années pour accroître la fiabilité des systèmes des transmissions [6]. De nombreuses architectures parallèles-pipeline ont été conçues pour les codes correcteurs d'erreurs afin d'augmenter leur débit du fonctionnement [53, 54, 57, 58, 60, 61, 64].

Dans ce contexte, les codeurs et décodeurs sont des circuits critiques pour contrôler l'utilisation correcte du canal de transmission. Par conséquence, un intérêt particulier doit être consacré à l'aspect fiabilité de ces circuits, c'est-à-dire, à leur sûreté de fonctionnement.

Le comportement erroné d'un circuit peut s'avérer tout à fait inacceptable si celui-ci est mis en œuvre dans une application dite critique. C'est la raison pour laquelle les fabricants ont besoin d'outils efficaces pour analyser le comportement de leurs circuits lorsqu'une ou plusieurs erreurs se produisent pendant leur fonctionnement. La découverte par le concepteur d'un comportement inacceptable le pousserait à modifier sa description initiale pour supprimer cette vulnérabilité. En conséquence, pour minimiser un surcoût éventuel, il est nécessaire de réaliser une analyse tôt dans le flot de conception, particulièrement avant que le circuit ne soit fabriqué. L'analyse de sûreté peut être menée grâce à des techniques d'injection de fautes. Le principe est de comparer le comportement nominal du circuit (sans injection de fautes) avec son comportement en présence de fautes, injectées lors de l'exécution d'une application. Des campagnes d'injection de fautes peuvent être réalisées selon plusieurs approches, en particulier la simulation ou l'émulation pour des approches haut niveau. La simulation, plus couteuse en temps, peut cependant permettre des analyses plus fines et complètes que l'émulation.

Le contexte de cette thèse se situe au niveau des systèmes de transmission fiable de données à haut débit. En effet, les systèmes actuels requièrent des débits de transmission très élevés, des techniques de modulation variées et des protocoles de communications complexes. Ces applications sont caractérisées par l'emploi, entre autres, de fonctions complexes et coûteuses en temps de traitement. Certaines de ces fonctions, telles que la modulation et la démodulation numériques, le filtrage, concernent l'adaptation du signal à la nature du support de transmission. D'autre fonctions, telles que le codage

canal, ont pour but de renforcer la robustesse des informations transmises. La conception architecturale sûre de fonctionnement pour ces fonctions est un facteur déterminant, puisque notre objectif est non seulement de transmettre les données à haut débit mais aussi de les protéger contre des altérations non désirées.

Cette thèse s'inscrit dans deux axes de recherche menés au sein du laboratoire LICM. Le premier concerne la conception architecturale d'une chaîne de transmission à haut débit et faible coût. Le deuxième vise à développer une méthodologie de conception architecturale de codeur/décodeur sûr de fonctionnement.

L'objectif principal de ce travail donc est de concevoir une architecture rapide et fiable pour un système de codage à faible coût et à haut débit permettant une protection optimale des données. Le but imposé par le cahier des charges est de proposer un système de codage qui satisfasse les quatre points suivants :

- protection optimale des données ;
- traitement rapide (haut débit) ;
- architecture sûre de fonctionnement ;
- utilisation de technologies à faible coût.

Des systèmes complexes de codage/décodage, modulation/démodulation, et des méthodes d'accès adaptées sont donc apparus pour exploiter au mieux les capacités des médias de transmission. Les architectures d'émetteur et de récepteur qui en découlent ont elles aussi dû évoluer pour supporter les cadences de traitement plus élevées. Concernant l'augmentation de la vitesse de traitement, deux solutions sont envisageables : une solution directe qui consiste à choisir une technologie cible rapide en fonction des besoins sans prendre en compte le facteur coût, solution qui ne rentre pas dans le cadre de nos objectifs globaux. Une deuxième solution consiste à trouver de nouvelles architectures permettant d'atteindre de hauts débits sur des cibles technologiques à faible coût type FPGA *(Field Programmable Gate Array)*, ce qui est en adéquation avec les objectifs que nous nous sommes fixés. Ils présentent de nombreux avantages : en particulier leur faible coût, mais aussi le fait d'offrir une capacité d'évolution importante aux systèmes, permettant par conséquent de s'adapter rapidement aux changements de protocoles, fréquents dans le domaine des télécommunications. En outre, ils s'intègrent parfaitement dans la chaîne de conception d'un système où la réutilisation de blocs fonctionnels devient primordiale avec l'augmentation de la complexité de ceux-ci et des coûts et temps de développements inhérents. La solution proposée consiste à développer un modèle RTL (*Register Transfert Level*) d'une architecture rapide pour le codeur/décodeur du code en bloc ou convolutif indépendamment d'une technologie cible particulière et d'en évaluer les performances en terme de débit de traitement et de surface consommée après synthèse sur FPGA.

Plan du mémoire

Ce manuscrit est composé de quatre chapitres.

Le premier chapitre aborde les concepts concernant les codes correcteurs d'erreurs nécessaires à la bonne compréhension des travaux effectués dans le cadre de cette thèse. Tout d'abord, quelques généralités sur les communications numériques sont présentées, puis les codes en blocs et les codes convolutifs sont décrits avec leur propriétés principales. Cette partie se focalise sur les principes de base du codage et du décodage pour différentes des codes et présente leur implantations architecturales séries traditionnelles. Ensuite des solutions parallèles-pipeline récentes pour différents codes présentées permettant d'augmenter leur performances (réduction du chemin critique, augmentation du rendement, etc.).

Le second chapitre introduit les concepts de base et la terminologie de la sûreté de fonctionnement. Il ensuite met en lumière les défaillances, les erreurs et les fautes pouvant se produire dans de tels systèmes et présente des exemples de techniques des recours possibles face à de telles fautes, qu'il s'agisse de techniques matérielles de détection seule, ou de détection et correction. Nous présentons certaines méthodes de tolérance aux fautes existantes ainsi que leurs spécificités, utilités et qualités.

Le troisième chapitre introduit une nouvelle architecture rapide avancée pour les codeurs convolutifs MTO (*Many To One*) et OTM (*One To Many*). La première partie de ce chapitre décrit les désavantages de techniques de parallélisation précédentes. Ensuite, nous présentons une nouvelle architecture parallèle-pipeline décrite par les équations qui régissent le fonctionnement du codeur parallèle-pipeline. Pour évaluer les performances du codeur haut débit, un modèle générique décrit en VHDL au niveau d'abstraction RTL (Register Transfert Level) a été synthétisé sur des FPGA de type Stratix-II d'Altera. Un codeur récursif pouvant être vu comme un filtre à réponse impulsionnelle infinie RII, une adaptation de la nouvelle méthode de parallélisation des codeurs est appliquée aux filtres récursifs RII.

Le quatrième et dernier chapitre porte sur l'étude et la validation d'une méthodologie de conception d'architectures de codeurs cycliques en blocs sûres de fonctionnement FS (*Fault-Secure*). Nous démontrons une propriété permettant de minimiser le surcoût matériel nécessaire pour implanter le codeur FS. Pour évaluer le niveau de tolérance aux fautes de la méthode proposée, la technique mise en œuvre consiste à réaliser différentes compagnes d'injection de fautes simple SEU et multiples MBU (*Multiple Bit Upset*) au niveau d'abstraction RTL, et de vérifier à chaque simulation le taux de détection d'erreurs du codeur FS. Différents codeurs, correspondant à différents codes cycliques, ont été synthétisés sur des FPGA de type Stratix-II d'Altera pour évaluer la performance et la consommation en surface de ces codeurs FS. Finalement, nous présenterons une version FS de décodeurs parallèles-pipeline cycliques en bloc capables de contrôler leur fonctionnalité à chaque cycle d'horloge et implantés eux aussi sur FPGA.

I. ETAT DE L'ART ET ÉTUDE THÉORIQUE

Chapitre 1

Codes correcteurs d'erreurs

1.1 Introduction

Depuis leur apparition, les nouvelles technologies de communication exigent des normes de plus en plus strictes en termes de qualité de service, la diversité et les volumes croissants de données échangées/traités nécessitant des systèmes de plus en plus rapides. À ces contraintes liées au traitement de l'information s'ajoute la nécessité de prendre en compte la sensibilité accrues des technologies face aux sources perturbatrices externes. Il s'agit notamment de protéger les informations contre les altérations induites par l'environnement lors de la transmission ou par les fautes temporaires de type SEU (*Single Event Upset*) [2] apparaissant dans les systèmes de traitement et causés par les radiations cosmiques. De la qualité du service rendu en termes de communication se dégagent deux mots clés : fiabilité et rapidité. À signaler que différents codes détecteurs d'erreurs EDC *(Error Detecting Codes)* et d'autres codes correcteurs d'erreurs ECC *(Error Correcting Codes)* ont été utilisés pendant des années pour accroître la fiabilité des systèmes de transmissions [6], et plusieurs architectures parallèle-pipeline ont été conçues pour les codes correcteurs d'erreurs pour augmenter leur débit [58, 61, 64].

Ce chapitre vise à définir le domaine d'application considérée dans le cadre de cette thèse, et plus spécifiquement dans le cadre des codes correcteurs d'erreurs ECC. Dans un premier temps, nous nous intéressons à préciser la place de l'application dans une chaîne de transmission numérique. Ensuite, nous exposons les propriétés des codes ECC en blocs et convolutifs pris en considération dabs ce mémoire. Ensuite, nous présentons pour les deux type de codes, différentes architectures de codeurs/décodeurs série existantes. Finalement, nous avons analysons l'intérêt des différents travaux réalisés dans le but d'améliorer les performances par la mise en œuvre d'ensembles codeur/décodeur parallèles-pipeline.

1.2 Chaîne de transmission

De nos jours, la transmission de l'information d'un émetteur à un destinataire se base principalement sur des techniques de transmission numérique. Il est en effet aisé de manipuler un signal numérique et de lui appliquer les traitements souhaités. Un exemple de modélisation de chaîne de transmission numérique est donné dans la figure 1.1.

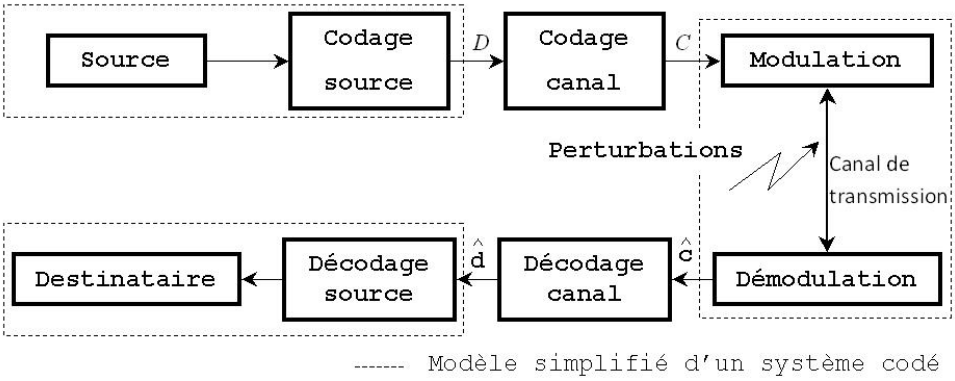


FIG. 1.1 – Modélisation d'une chaîne de transmission numérique

▷ **La source :** La source du message émet l'information sous la forme de symboles binaires.

▷ **Le codage/décodage source :** Le codage source consiste à transformer le message de la source en une séquence d'information « $D(x)$ » de façon à : 1– minimiser la taille du message en éliminant les redondances naturelles de l'information source (algorithme de compression) ; 2– retrouver le message original à partir de la séquence de substitution « $D(x)$ » (algorithme réversible). Les symboles émis par la source sont convertis à partir d'un alphabet de symboles (ordinairement des bits) afin que ceux-ci puissent être récupérés au cours de la réception sans modification à partir des données binaires (codage sans perte) ou alors avec une distorsion (codage avec perte). Le codeur source réduit la redondance contenue dans le message et minimise ainsi la quantité d'information utile à sa représentation. Le décodage source réalise l'opération duale, le message d'information est décompressé afin de retrouver son équivalent à partir de la séquence de substitution « $D(x)$ » avant la transmission. Il est à noter que les limites théoriques du codage source sont fixées par le premier théorème de Shannon [7, 8].

▷ **Le codage/décodage canal :** Le codage canal a pour rôle de protéger l'information émise contre les perturbations du canal de transmission susceptible de modifier son contenu. Il s'agit donc de rajouter de la redondance de manière à détecter et éventuellement corriger les erreurs lors de la réception si la stratégie adoptée le permet. L'information $D(x)$ issue du codage source est transformée en séquence codée $C(x)$. Comme le décrit le théorème fondamental du codage canal, pour se rapprocher de la capacité du canal de transmission, il est nécessaire de coder l'information avant de la transmettre. Au niveau du récepteur, le décodage canal consiste dans un premier temps à détecter la présence d'erreurs dans l'information et puis dans un deuxième

temps de les corriger. Les codes correcteurs d'erreurs ont été utilisés pour la détection et la correction des erreurs induites par le canal la transmission. Différentes types de codes ont été utilisés ; parmi ceux on distingue les codes en blocs et les codes convolutifs. Pour un code en blocs, la trame d'entrée de k symboles d'information est convertie en une séquence de sortie sur n symboles (avec $k \leq n$), Le bloc de sortie sur n symboles dépend uniquement des k symboles de la trame d'entrée et du rendement de code $r = k/n$. Par contre, le codeur d'un code convolutif a un effet mémoire et prend pour entrée un symbole de k bits et fournit en sortie un symbole de n bits. Les n bits en sortie sont calculés par une combinaison linéaire entre les k bits en entrée et les m blocs mémorisés (précédents). Le rendement du code convolutif $r = k/n$ et sa longueur K_c est le nombre maximum de bits associés à une sortie qui peuvent être affectés par un bit quelconque à l'entrée. Par la suite, nous nous sommes interessés au cas du code de rendemment $r = 1/n$.

- ▷ **Le modulation/démodulation :** La modulation agit sur les paramètres d'un signal porteur afin de transmettre les données codées. Dans le cas de la modulation numérique, le message codé est transformé à partir d'un alphabet dont l'entrée correspond à une partie du signal à transmettre (i.e. un symbole). Le signal porteur est une sinusoïde dont on peut faire varier l'amplitude, la fréquence ou la phase indépendamment (ASK (*Apmplitude Shift Keying*), FSK (*Frequency Shift Keying*), PSK (*Phase Shift Keying*)) ou simultanément QAM (*Quadrature Amplitude Modulation*), en fonction de l'information à émettre. Le démodulateur joue le rôle dual du modulateur et transforme donc le signal reçu en un train binaire. Avec les nouvelles techniques (notamment avec l'étalement de spectre) on module souvent plusieurs caractéristiques en même temps.

- ▷ **Le canal de transmission :** Il représente la liaison entre l'émetteur et le récepteur et peut être de différentes natures selon le type de grandeur qu'il permet de véhiculer. Le canal de transmission est caractérisé par sa capacité et sa bande passante. Il existe plusieurs modèles théoriques du canal de transmission en fonction des types d'erreurs les plus fréquents, nous nous limiterons au canal à Bruit Blanc Additif Gaussien (BBAG) et à un canal binaire symétrique [15].
  - Le canal BBAG : il s'agit d'un canal à entrée binaire et sortie analogique. La sortie se représente par une variable aléatoire continue y [9] :

$$Y \; = \; x \; + \; b \tag{1.1}$$

où x est le symbole binaire émis et b est une variable aléatoire gaussienne centrée de variance σ^2 correspondant au bruit du canal. La variance est fonction du rapport signal à bruit :

$$\sigma^2 \; = \; 1/2 \; (E_b/N_0)^{-1} \tag{1.2}$$

où E_b est l'énergie moyenne utilisée pour transmettre un symbole binaire et N_0 est la densité spectrale de puissance monolatérale du bruit additif. Les échantillons transmis au décodeur

de canal sont, en général, numériques, quantifiés sur Q bits. Ils résultent d'une conversion analogique/numérique.

– Le canal binaire symétrique : la valeur Q définie précédemment prend deux valeurs dans le cas d'un canal binaire symétrique (CBS) sans mémoire et stationnaire. C'est le modèle le plus simple utilisé dans la théorie des codes correcteurs d'erreurs. Il sert de point de comparaison avec d'autres modèles. La décision en sortie du canal est ferme et non plus pondérée sur les échantillons reçus. Les entrées et les sorties de ce canal discret sont binaires. Les erreurs de transmission sont mutuellement indépendantes et apparaissent sur les entrées avec une probabilité p identique et invariante dans le temps. Les probabilités de transition du CBS sont présentées sur la figure 1.2

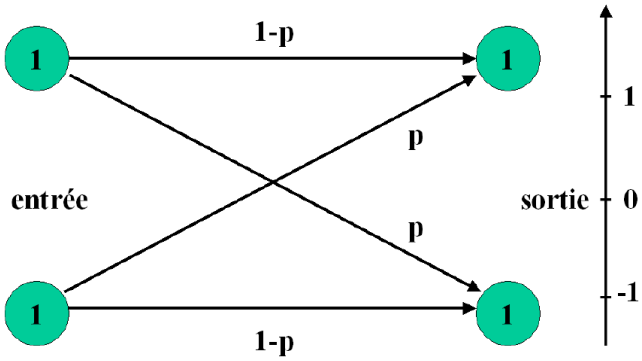


FIG. 1.2 – Graphe de transition du canal binaire symétrique

La prise en considération du CBS est à la base du développement d'algorithmes de décodage algébriques basés sur le modèle mathématique des codes en blocs [10, 11]. Pour des sorties de canal analogique, les algorithmes employés sont de type probabiliste. Ils sont dédiés à des codes convolutifs [12] et à des codes en blocs [13, 14].

1.2.1 Performance d'un système de transmission : gain de codage

La qualité d'une transmission numérique est caractérisée par la probabilité d'occurence d'erreur par élément binaire transmis. Elle est notée P_{E_b} (ou P_{E_s} dans le cas de symboles q-aires). Cette probabilité est fonction du rapport signal sur bruit (SNR) E_b/B. Le tracé de la courbe reliant les points de la P_{E_b} (à rapport signal/bruit donné) reflète directement la qualité de la transmission. L'obtention de ces points nécessite des simulations complexes qui permettent de mesurer le P_{E_b} à travers le taux d'erreur binaire (TEB). L'estimation du TEB est obtenue par simulation de la transmission de N symboles binaires et l'évaluation après décodage du rapport n_e/N où n_e est le nombre de symboles erronés après décodage en réception. La pratique a montré que l'obtention d'au moins une centaine d'erreurs est nécessaire pour avoir une estimation correcte de la p_{E_b}.

En l'absence de codage sur un canal BBAG et pour une modulation BPSK, la probabilité d'erreurs

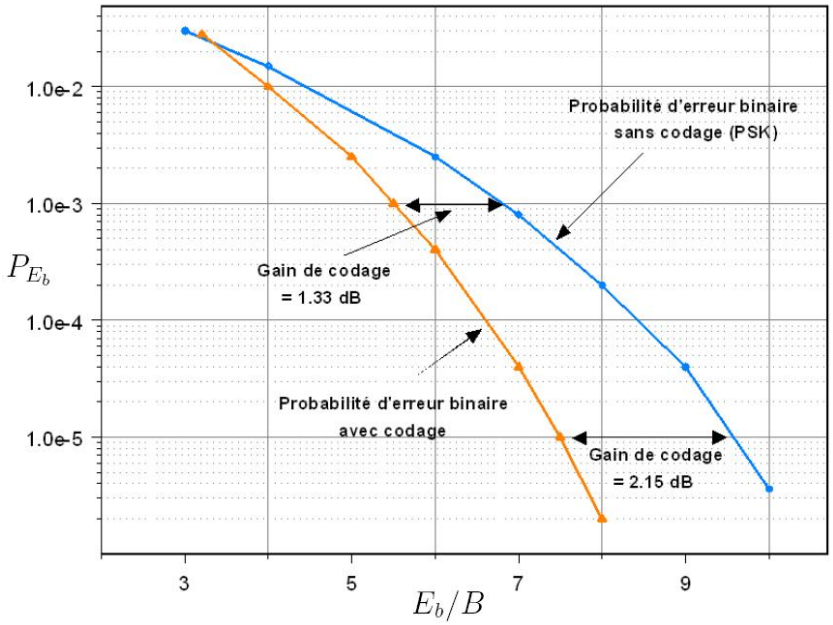


FIG. 1.3 – P_{E_b} en fonction du E_b/B pour un code Golay (23,12) avec décision ferme pour une modulation de type BPSK

par élément binaire transmis s'écrit [15] :

$$P_{E_b} \;=\; 1/2\; f_{ec}(\sqrt{E_b/B}) \tag{1.3}$$

où $f_{ec}(x)$ est la fonction d'erreur complémentaire définie par :

$$f_{ec}(x) \;=\; 2/\sqrt{\pi}\; \int_{+\infty}^{x} \exp^{-t^2} \cdot dt \tag{1.4}$$

L'efficacité du code correcteur d'erreur est déterminée en effectuant la comparaison des courbes en sortie du décodeur de canal et en sortie du canal (en l'absence de codage). La distance entre les deux tracés donne le gain de codage G (fig. 1.3), il s'exprime en décibel dB. G représente l'économie énergie induite par l'utilisation d'un codage canal. Il est également possible de le voir comme une amélioration de la qualité de transmission.

En résumé, dans les prochaines sections, nous allons expliciter le domaine d'application retenu dans cet mémoire et plus particulièrement les codes correcteurs d'erreurs. Après un rappel sur leur rôle dans une chaîne de communications numériques, les codes en blocs et les codes convolutifs sont détaillés. Puis, une présentation de leurs architectures séries de base, et leur versions parallèle-pipeline existants dans le but d'améliorer la performance, suivie d'une discussion concernant leur problèmes de complexité et le débit du fonctionnement seront proposées.

1.3 Propriétés et architectures séries des codes

Cette partie vise à définir les propriétés des codes en blocs et des codes convolutifs. Comme nous nous intéressons dans ce mémoire à la conception d'architectures rapides et fiables, nous discutons dans la suite de cette section, de leur implantation architecturale, ainsi que de différentes techniques existantes pour les rendre fiables.

1.3.1 Techniques de codage

Le codage consiste en l'utilisation de codes en rajoutant de la redondance au message à transmettre permettant à la réception la détection, la localisation et la correction des erreurs induites par le canal de transmission. Différents types de codes ont été utilisés jusqu'à présent, parmi lesquels on distingue : les codes en blocs et les codes convolutifs. Parmi les codes en bloc citons les codes CRC (*Cyclic Redundancy Check*) [52, 53], BCH (*Bose Chaudhuri Hocquenghem*) [10, 11], et RS (*Reed Solomon*) [17, 18] qui présentent une propriété fondamentale de linéarité puisqu'un code linéaire forme un espace vectoriel linéaire, et une propriété cyclique [6] ; ainsi il est possible d'additionner deux mots de codes pour produire un troisième mot code. Le processus de codage et décodage est simplifié puisqu'il est possible de définir n'importe quel mot code comme une combinaison linéaire d'autres mots codes. La distance de Hamming est équivalente au nombre d'éléments non nuls dans le mot code. En pratique, la plupart des techniques de codage utilise des codes linéaires. Le décodeur effectue l'opération inverse du codeur. Pour un code en blocs, des méthodes algébriques sont utilisées pour la résolution de systèmes d'équations, citons par exemple le cas du décodeur cyclique de type Meggitt [6]. Pour un code convolutif, les méthodes de résolution se basent sur le parcours d'un treillis, par exemple un décodage avec le maximum de vraisemblance pour l'algorithme de Viterbi [6].

Les codes convolutifs, comme les codes blocs, peuvent devenir systématiques si le message $D(X)$ est directement contenu dans le mot code $C(X)$. D'autre codes, appelés codes concaténés, sont construits en concaténant plusieurs codes convolutifs ou codes blocs séparés par des blocs d'entrelacement. C'est le cas notamment des turbo-codes résultant de la concaténation de deux ou plusieurs codes convolutifs [42]. D'autres systèmes ont opté pour des processus de codage équivalent aux turbo-codes tels les codes LDPC (*Low-Density Parity-Check*) [46] qui sont par ailleurs les codes adoptés dans le nouveau standard de transmission de vidéo numérique par satellite, le DVB-S2 [47].

1.3.2 Codes en blocs

Le codage en blocs consiste à fractionner la trame d'information en plusieurs blocs de taille fixe k, pour ensuite transformer chacun des messages D_i en un mot de code C_i de taille n en appliquant une loi linéaire (fig. 1.4). La redondance associée à chaque bloc est de taille r, où $k + r = n$. Le rendement d'un code bloc est donné par la formule suivante :

$$R = \frac{k}{n} \tag{1.5}$$

où, k et n représentent respectivement les nombres de bits en entrée et en sortie du codeur.

Les codes en blocs linéaires (ou codes de groupe) constituent un faible pourcentage de l'ensemble des codes en blocs. Cependant, il s'agit des codes en blocs les plus utilisés en pratique.

Historiquement [6] le premier code bloc est le « code de Hamming » qui a été inventé par Richard Hamming en 1946, puis amélioré et généralisé par Marcel Golay qui introduit à son tour les codes connus sous son nom : « codes de Golay ». On retrouve les codes de Hamming dans la norme Bluetooth [16]. Le code de Golay binaire a été utilisé par la NASA dans la sonde spatiale Voyager I. Une

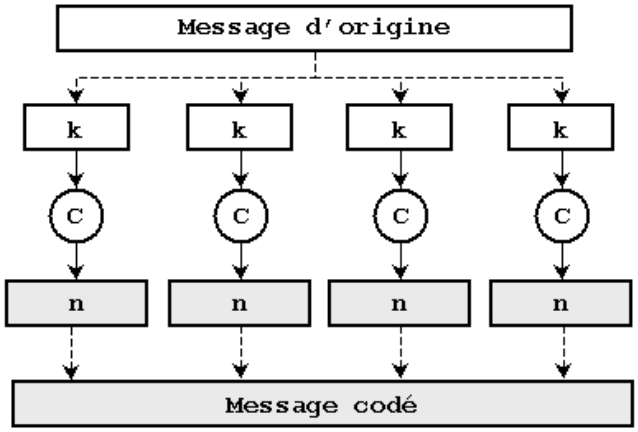


FIG. 1.4 – Codage en bloc

autre classe importante des codes en bloc sont les codes de Reed-Muller introduit par Muller en 1954. Les codes LDPC, apparus en 1957, sont des codes à faible densité et forment une classe de codes en blocs qui se caractérisent par une matrice de contrôle creuse. Ils ont une facilité d'optimisation de la structure du code et sont référencés comme une famille de codes performants pouvant atteindre la capacité de certains canaux standards comme le canal binaire à effacement [20–22]. Ils constituent une alternative intéressante aux turbo-codes également très performants sur de nombreux canaux standards. Les codes CRC sont apparus en 1957. Ils sont généralement utilisés pour la détection d'erreurs (protocoles X25, Ethernet, FDDI (*Fiber Distributed Data Interface*), ATM-AAL5 [1]) [53], la correction s'effectuant par retransmission. Les mots de codes générés par les CRC peuvent être décodés en utilisant les décodeurs cycliques de type Meggitt [6]. Les codes BCH découverts par Hocquenghem en 1959 ainsi que Bose et Chaudhuri en 1960 sont la généralisation des codes de Hamming pour la correction d'erreurs multiples. Ils ont été étendus au cas non-binaire (code RS) par Reed et Solomon en 1960. Grâce à leur nature non binaire l'utilisation des codes RS est adéquate dans le cas de canaux où les erreurs apparaissent par paquet [2] [17]. On retrouve par exemple le code RS(255,233,33) qui est utilisé par la NASA dans les communications spatiales. Ils sont largement utilisés dans les disques compacts, les DVD, la transmission HDTV [18] et préconisés par le standard CDPD (*Cellular Digital Packet Data*) ainsi que par la norme DVB-C (*Digital Video Broadcasting-Cable*) [19].

[1] AAL (*ATM Adaptative Layer*) est la troisième couche du protocole ATM (*Asynchronous Transfer Mode*). L'AAL-5, également connu sous le nom de SEAL (*Simple and Efficient Adaptation Layer*), est une des couches d'adaptation utilisées pour la transmission des données et des signalisations à débits variables (VBR : *Variable Bit Rate*) [207].

[2] Pour des petits paquet d'erreurs seulement, sinon il faut utiliser de l'entrelacement.

▷ **Codes linéaires systématiques** : Un mot de code d'un code en blocs linéaire $C(n,k)$ (avec $k < n$) construit sur un corps de Galois GF *(Galois Field)* [3] se compose de :
- k symboles composés de la séquence d'information à transmettre répartis dans l'ensemble du message.
- $n-k$ symboles calculés à partir d'une combinaison linéaire d'une partie prédéterminée des symboles d'information et eux aussi répartis dans le message. Il s'agit des symboles de parité ou de redondance.

$C(n,k)$ est un sous-espace vectoriel de dimension k de l'espace engendré par $GF(2^n)$, n correspond à la longueur du code, k à sa dimension et k/n au rendement du code.

Le code est dit systématique si les k symboles représentant le message sont transmis tels quels. Les $n-k$ symboles restants sont alors dans ce cas les symboles de parité (figure 1.5)

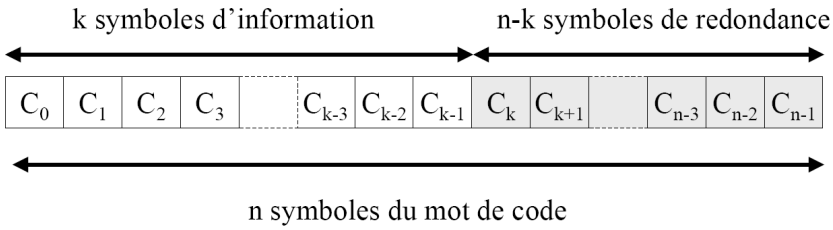


FIG. 1.5 – Représentation d'un mot de code pour un code systématique

L'addition et la soustraction sont deux opérations identiques puisque le corps $GF(2^n)$ est obtenu à partir du corps $GF(2)(=0,1)$. Le formalisme matriciel est utilisé pour expliciter la fonction de codage, la matrice génératrice $[G]$ du code $C(n,k)$ se compose de k lignes et de n colonnes telle que [23] :

$$C \;=\; D \cdot [G] \tag{1.6}$$

où D est le message d'information de dimension k et C est le mot de code de longueur n généré. Une matrice de contrôle (ou de parité) H peut également être associée au code telle que :

$$[H] \cdot [G]^t \;=\; [G] \cdot [H]^t \;=\; [0] \tag{1.7}$$

donc pour tout mot de code de C :

$$S(C) \;=\; [C] \cdot [H]^t \;=\; D \cdot [G] \cdot [H]^t \;=\; [0] \tag{1.8}$$

Cette dernière équation est importante puisqu'elle pose les bases de la détection et de la correction des erreurs de transmission. Le vecteur $S(C)$ est appelé syndrome de C. Un syndrome nul indique que le mot reçu est un mot de code mais il ne garantit pas qu'il s'agisse du mot de code émis. En effet, un mot de code peut très bien se substituer à un autre en fonction du motif

[3] Plus de détail sur le corps de Galois $GF(2^n)$, et la construction de corps de Galois d'ordre m $GF(2^m)$ se trouvent dans [6].

d'erreurs (ME). Par exemple, si R est le mot reçu, E le motif d'erreurs et C le mot de code émis, alors $R = C \oplus E$ et $S(R) = E \cdot [H]^t$. Le syndrome dépend donc uniquement du motif d'erreurs.

Une autre caractéristique importante des codes en blocs linéaires est la distance minimale de Hamming (d_{min}) qui désigne le plus petit nombre de bits différents entre deux mots de codes distincts. Ainsi un code en blocs de distance (d_{min}) est capable de détecter les motifs de $(d_{min} - 1)$ erreurs dans un bloc de dimension n et de corriger tous les motifs de t erreurs :

$$t \ = \ \lfloor (d_{min} - 1)/2 \rfloor \tag{1.9}$$

où le symbole $\lfloor \ \rfloor$ représente la partie entière. t est appelé le pouvoir de correction du code.

▷ **Codes cycliques :** Les codes cycliques bénéficient de toutes les propriétés des codes en blocs linéaires en plus de la propriété cyclique. Pour rappel, pour tout décalage cyclique d'un mot de code, le mot généré est aussi un mot de code. si $C = (\ c_0,\ c_1,\ \cdots,\ c_{n-2},\ c_{n-1})$ est un mot de code alors le décalage cyclique de i produit $C = (\ c_{n-i},\ c_{n-i+1},\ \cdots,\ c_0,\ \cdots,\ c_{n-i-1})$ qui est aussi un mot de code.

Chaque mot de code $C = (\ c_0,\ c_1,\ \cdots,\ c_{n-1})$ du code $C(n,k)$ est associé à un polynôme $C(x) \ = \ c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$.

Comme tous les codes, un code cyclique est défini par son polynôme générateur [4] $G(X)$, où $G(x) \ = \ g_0 + g_1 x + \cdots + c_{n-k} x^{n-k}$. Le polynôme générateur d'un code cyclique C est le mot de code de degré le plus bas, ce polynôme est unique.

Deux modes de codage existent : le codage par multiplication et l'autre par division. Pour un codage par multiplication, tout mot de code $C(x)$ peut donc s'écrire sous la forme :

$$c(x) \ = \ d(x) g(x) \tag{1.10}$$

où $D(x) \ = \ d_0 + d_1 x + \cdots + d_{k-1} x^{k-1}$ est le message d'information à coder. $G(x)$ et également un facteur de $x^n + 1$.

Le codage par division est utilisé pour le codage systématique, et consiste à coder le message $D(x)$ à partir du polynôme générateur $G(x)$, de la manière suivante :

– multiplication de $D(x)$ par x^{n-k}.
– division de $x^{n-k} D(x)$ par $G(x)$:$x^{n-k} D(x) = Q(x) G(x) + R(x)$ où $R(x)$ est le reste de la division.
– addition de $x^{n-k} D(x)$ et $R(x)$:

$$C(x) = x^{n-k} D(x) + R(x) \tag{1.11}$$

[4] Par convention les polynômes liés à une boucle récursive seront identifiés par la notation « G », alors que dans le cas d'une branche directe ils seront notés par « H ».

donc

$$C(x) = x^{n-k}D(x) + [x^{n-k}D(x)] \mod G(x) \tag{1.12}$$

L'équation ci-dessus donne le mot de code $C(x)$ sous forme systématique et les composantes de $R(x)$ sont les symboles de redondance (ou de parité).

Comme nous nous sommes intéressés à concevoir des architectures rapides de fonctionnement, il est utile de présenter les différentes architectures séries de base utilisées, qui correspondent aux deux modes de codage existants : codage par multiplication et codage par division. Ces architectures sont implantées selon les équations 1.10 et 1.12.

▷ **Codage par multiplication** : Le codage par multiplication consiste à multiplier le bloc d'information par un multiplicateur fixe, le mot de code étant le résultat de la multiplication. L'opération de codage est définie par la multiplication polynomiale $C(x) = D(x) \times H(x)$, ce qui se traduit par une convolution discrète (équ. (1.13)) des coefficients du bloc d'information $D(x)$ et du polynôme générateur $H(x)$.

$$c_i = \sum_{j=0}^{m} d_{i-j} \cdot h_j \tag{1.13}$$

où c_i, d_i et h_j sont respectivement les coefficients de $C(x)$, $D(x)$ et $H(x)$.
Du point de vue architectural, les entrées et les sorties à l'instant t sont égales à $(X)_t$ [5] et $(Y)_t$ respectivement, $(X)_t$ contenant la valeur d'entrée d_i et $(Y)_t$ contenant la valeur de sortie c_i.

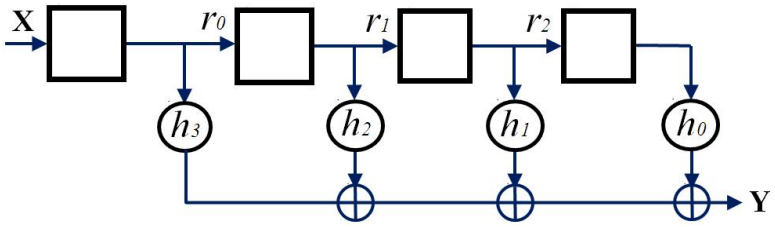


FIG. 1.6 – Multiplieur série MTO pour $H(x) = h_0 + h_1x^1 + h_2x^2 + h_3x^3$

La figure 1.6 représente l'architecture série dite *Many To One* (MTO) d'un circuit multiplieur pour un polynôme générateur $H(x) = h_0 + h_1x^1 + h_2x^2 + h_3x^3$. Le principal inconvénient de ce type d'architecture réside dans le fait que le chemin critique de cette structure dépend fortement du nombre de coefficients h_i non nuls. En effet ce dernier fixe le nombre de portes XOR misent en cascade [6], ce qui a pour effet d'augmenter le chemin critique en fonction du nombre de coefficients h_i qui prennent pour valeur 1. Cependant cet inconvénient peut être évité en adoptant une structure dite *One To Many* (OTM).

[5] La notation $()_t$ est utilisée pour désigner le contenu d'une ligne au début du cycle d'horloge t. Il sera désormais utilisé dans l'ensemble de ce document.

[6] La multiplication se traduit par une connexion simple ou pas de connexion selon que le coefficient correspondant est un 1 ou 0.

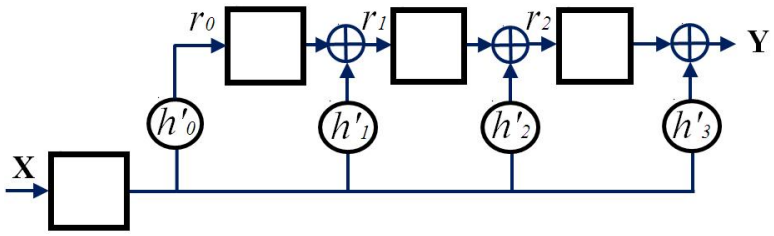


FIG. 1.7 – Multiplieur série OTM pour $H'(x) = h'_0 + h'_1x^1 + h'_2x^2 + h'_3x^3$

Comme le montre la figure 1.7, le chemin critique de cette structure est équivalent à celui d'une seule porte XOR, quel que soit le nombre de coefficients h'_i non nuls du polynôme générateur $H'(x)$. La seule condition à respecter lors du passage d'une structure à l'autre est d'intervertir les coefficients selon l'équation (1.14).

$$h_i = h'_{j-i}, \quad \text{pour } i \in \{0, 1, \cdots, j\} \tag{1.14}$$

Pour de raison d'homogénéité des notations, nous utiliserons dans la suite la notation h pour le cas OTM.

▷ **Codage par division** : L'algorithme de division du bloc d'information (dividende) $D(x)$ par le polynôme générateur (diviseur) $G(x)$ dans $GF(2)$ revient à une succession de tests sur le bit de poids fort du dividende.

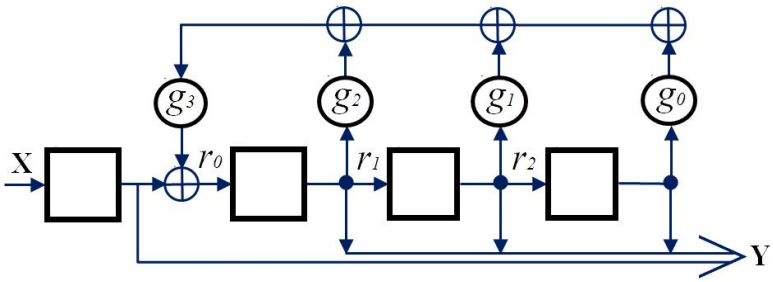


FIG. 1.8 – Diviseur série MTO pour $G(x) = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

La figure 1.8 représente l'architecture série MTO d'un diviseur pour un polynôme générateur $G(x) = g_0+g_1x^1+g_2x^2+g_3x^3$. Par analogie avec le multiplieur, le point faible de cette structure réside dans le nombre de coefficients g_i non nuls qui fixe le nombre de portes XOR mises en cascade. À la fin du codage, le reste de la division est disponible dans le registre constitué des bascules ($r_0,\ r_1,\ \cdots,\ r_{m-1}$).

La structure OTM du diviseur série (fig. 1.9) permet de réduire le chemin critique à une seule porte XOR. Le passage d'une structure à l'autre se fait en respectant les mêmes conditions que pour le codeur multiplieur (équ. (1.14)).

Décodeur cyclique : Après la transmission, le mot code reçu est $C'(x) = C(X) + E(X)$. $E(X)$ désigne les bits erronés par le bruit du canal. Ces erreurs de bits peuvent être détectées si la capacité

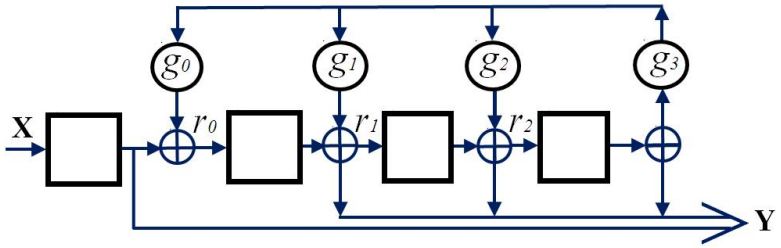


FIG. 1.9 – Diviseur série OTM pour $G(x) = g_0 + g_1x^1 + g_2x^2 + g_3x^3$

de correction des erreurs du code n'est pas dépassée. Pour décoder le mot code reçu C', le syndrome S doit être calculé comme suit :

$$\begin{aligned} S(X) &= C'(X) \bmod G(X) \\ &= [C(X) + E(X)] \bmod G(X) \\ &= E(X) \bmod G(X) \end{aligned} \tag{1.15}$$

où $C(X) \bmod G(X) = 0$.

En effet, le syndrome de S calculé ne dépend que de l'erreur E. Si $S(X) = 0$, $C'(X)$ est supposé sans erreur. Si $S(X) \neq 0$, la correction de $C'(X)$ est nécessaire. L'évaluation de l'erreur où les bits erronés sont estimés à partir du syndrome calculé précédemment. L'estimation de l'erreur est faite par une fonction de décision telle que la fonction de détection du motif d'erreur EPD (*Error Pattern Detection*) ou par une fonction à logique majoritaire MLV (*Majority Logic Voting*).

La figue 1.10 représente l'architecture série d'un décodeur cyclique de type Meggitt [6].

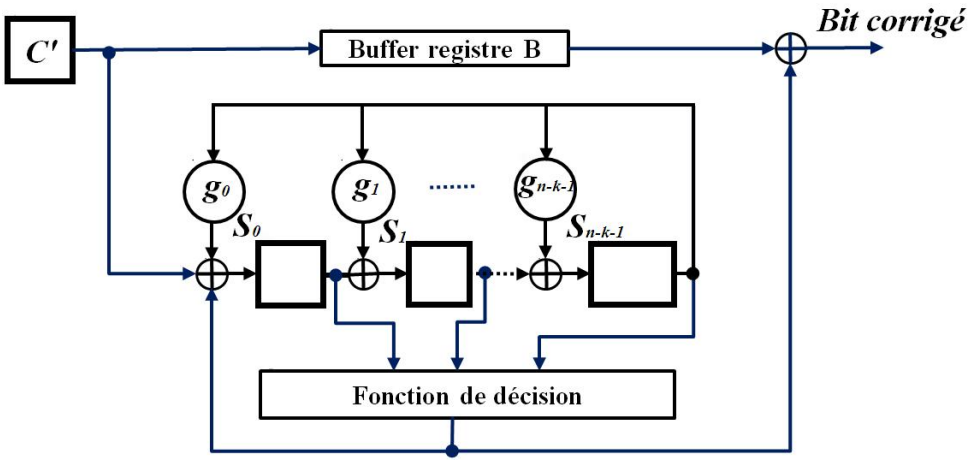


FIG. 1.10 – Architecture série du décodeur cyclique

Le fonctionnement du décodeur est décrit par les 3 étapes suivantes :

1. Le mot code reçu C' est décalé simultanément dans le buffer du registre B et le registre de syndrome S. Après n cycles d'horloge, le registre de syndrome contient le syndrome $S(x)$ de $C'(x)$.

2. Dès que le syndrome a été calculé, le mot code reçu dans le buffer peut être corrigé. À chaque cycle d'horloge, un nouveau bit sortant bu buffer est corrigé si nécessaire. La valeur de la correction est évaluée par la valeur du syndrome. L'équation suivante décrit l'opération performante à chaque cycle d'horloge :

$$\begin{aligned}(S)_{t+1} &= (S)_t \cdot T + F[(S)_t] \cdot (1,0,0) \\ (B)_{t+1} &= (B)_t \cdot T_R\end{aligned} \tag{1.16}$$

où $(S_t) = (s_0, s_1, \cdots, s_{n-k-1})$ et (B_t) sont respectivement les contenus des registres du syndrome et le buffer à l'instant t, F étant la fonction de décision, T and T_R les matrices de transition et de décalage respectivement.

3. L'étape précédente est répétée jusqu'a ce que le mot reçu $C'(x)$ soit décodé.

Le décodeur à logique majoritaire est un décodeur efficace pour décoder certains codes cycliques. La complexité de ce type de décodeur augmente de façon exponentielle avec le nombre d'erreurs corrigées [24]. De différents codes ont été conçus pour une liste de critères données, parmi ceux-ci citons : évaluation d'erreur, capacité de correction/détection d'erreur, complexité et performance du codeur/décodeur, etc...

Codes CRC, BCH, et RS

Parmi les codes cycliques en blocs on distingue les codes cycliques redondants CRC, les codes BCH, et les codes RS. Nous parlerons plus en détails pour chacun de ces codes sur leur proriétés et de leurs implantations, en particulier, pour les architectures séries du codeur et décodeur.

▷ **Les codes de redondance cyclique (CRC)** sont des codes systématiques, la procédure de codage est réalisée selon l'équation 1.12 et consiste à décaler de $n-k$ positions le bloc d'information $D(x)$ c'est-à-dire de prémultiplier $D(x)$ par x^{n-k}, ensuite de calculer le reste $R(x)$ de la division polynomiale de la séquence $x^{n-k} \cdot D(x)$ par un polynôme générateur $G(x)$ connu de l'émetteur et du récepteur.
La matrice [7] G correspondant au polynôme générateur $G(x)$ peut s'écrire sous la forme :

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k} & \cdots & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & 0 & g_0 & g_1 & \cdots & g_{n-k} \end{pmatrix}$$

où $n-k$ représente la taille de la mémoire du codeur.
A partir des deux circuits diviseurs vus précédemment (fig.1.8 et 1.9) il est possible de réaliser un codeur cyclique systématique. Il suffit de court-circuiter l'entrée et la sortie pour générer les

[7] A noter que la représentation matricielle et la représentation polynômiale de G sont équivalentes. La représentation matricielle s'avère fastidieuse lorsque le nombre de coefficients de G augmente.

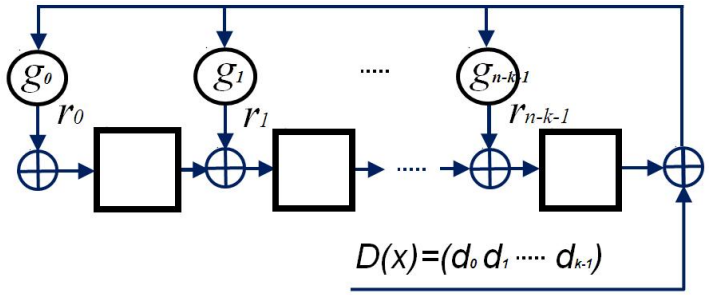


FIG. 1.11 – Implantation série de la division du polynôme par $G(x)$

bits d'information (sortie systématique) et de réaliser les étapes nécessaires au bon fonctionnement du codeur. La figure 1.11 donne l'exemple d'un codeur CRC de polynôme générateur $G(x) = g_0 + g_1 x + \cdots + g_{n-k-1}x^{n-k-1} + x^{n-k}$, où les données $D(X)$ sont premultipliées par x^{n-k} (circuit avec prémultiplication). Dans ce cas, à la fin des k cycles d'horloge la redondance $R(x)$ est disponible dans les registres $(r_0, r_1, \cdots, r_{n-k-1})$.
Au niveau du récepteur, la séquence reçue $C'(x)$ est divisée par $G(x)$. Si le reste de la division $R'(x)$ est nul, alors aucune erreur n'est détectée ; dans le cas contraire la séquence reçue est erronée et une phase de correction est nécessaire. Les mots de codes générés par les CRC peuvent être décodés en utilisant un décodeur cyclique 1.3.2.

▷ **Les codes BCH** sont des codes cycliques, ils portent le nom de leurs inventeurs Bose, Ray-Chaudhuri et Hocquenghem [25, 26]. Il s'agit de codes relativement performants, simples à mettre en œuvre et pour lesquels il existe un ensemble d'algorithmes de décodage algébrique de faible complexité. L'ensemble codeur/décodeur permet de construire un code cyclique et de corriger un nombre de t erreurs dans un bloc de n symboles codés transmis. Les notions d'algèbre dans le corps de Galois nécessaires à une compréhension plus approfondie des codes BCH ne sont pas présentées dans cette étude. Le lecteur intéressé pourra se référer à [27] pour plus d'informations. Seuls les codes BCH binaires primitifs sont évoqués dans cette section. Leur longueur de codage n peut s'écrire $n = 2m - 1$ (pour $n \geq 3$).
Pour un code $C(n, k)$ BCH binaire primitif de distance construite $d_{min} \geq 2t + 1$, le polynôme générateur $g(x)$ admet $2t$ racines $(\alpha, \alpha^2, \cdots, \alpha^{2t})$ et s'écrit sous la forme :

$$g(x) = PPCM\{m_1(x), m_2(x), \cdots, m_{2t}(x)\} \tag{1.17}$$

où PPCM est le plus petit commun multiple et $m_i(x)$ est le polynôme minimal de α^i.
α est un élément primitif de $GF(2^m)$ donc tous les éléments non nuls de $GF(2^m)$ sont des puissances successives de α ($GF(2^m) = \{0, \alpha^0, \alpha^1, \cdots, \alpha^{2m-2}\}$), avec $\alpha^{2m-2} = \alpha^0 = 1$ et α la racine n-ième de l'unité.
Les caractéristiques d'un code BCH binaire primitif ayant un pouvoir de correction t sont donc les suivantes :
– $n = 2^m - 1, \ (m \geq 3)$

– $n - k \leq mt$
– $d_{min} \geq 2t + 1$

Afin d'obtenir des codes ayant une distance de Hamming plus importante, une méthode classique consiste à ajouter un bit de parité globale (la somme modulo 2 de tous les autres bits). Il s'agit alors d'un code BCH étendu de paramètre $(n + 1, k, d + 1 = 2t + 2)$ obtenu à partir d'un code BCH primitif. La distance du code étendu est augmentée de 1 et devient paire. Le bit de parité généré n'augmente pas le pouvoir de correction du code mais facilite la détection des motifs d'erreurs non corrigibles. Le rendement associé est légèrement inférieur à celui du code primitif mais le comportement à fort rapport signal à bruit est meilleur.
Les symboles de redondance permettent le décodage des messages d'information reçus du canal. S est le vecteur associé aux composantes du syndrome $S = (s_1, s_2, \cdots, s_{n-k})$ et $S(x)$ est son polynôme. Les valeurs des composantes du syndrome peuvent s'exprimer à partir des symboles du mot reçu R et du corps de Galois considéré :

$$s_i = R(\alpha^i) = \sum_{j=0}^{n-1} r_j(\alpha^i)^j \ \ avec 1 \leq i \leq 2t \tag{1.18}$$

La fonction de décodage consiste à détecter et à corriger les erreurs en fonction du code BCH qui a été construit dans la partie émettrice.

Décodage des codes BCH : Le décodage dur consiste à exploiter les données binaires issues du canal obtenues par seuillage. Les algorithmes de décodage utilisent les mots reçus du canal et en particulier les symboles binaires de redondance pour estimer le mot émis. Les algorithmes utilisés varient en fonction du pouvoir de correction t du code choisi. En fait, plus le pouvoir de correction augmente plus les algorithmes de décodage correspondants sont complexes.
Un décodage ayant une complexité raisonnable est le décodage par syndrome. Il s'agit d'une version simplifiée d'un décodage optimal. En effet, l'exploitation des symboles de redondance permet de réaliser seulement 2^{n-k} comparaisons par rapport aux 2^k nécessaires classiquement. Ce type de décodage est particulièrement intéressant pour des codes à rendement élevé. Le syndrome est constitué de $n - k$ composantes non-nulles en présence d'erreurs. Le mot reçu s'écrit :

$$C'(x) = C(x) + E(x) \tag{1.19}$$

Comme indiqué précédemment (section 1.3.2), le syndrome dépend uniquement du motif d'erreurs. C'est pourquoi il est donc envisageable de concevoir une correspondance entre la valeur du syndrome et l'erreur estimée $\widetilde{E}$. Le processus de décodage se fait alors en trois étapes :
– le calcul du syndrome S du mot reçu C' ;
– la détermination de l'erreur estimée $\widetilde{E}$;
– le décodage du mot C' à l'aide de l'addition $C' + \widetilde{E}$.

Ce processus s'applique à des codes ayant un pouvoir de correction de $t = 1$.

Pour des pouvoirs de correction supérieurs à $1(t > 1)$, des méthodes de décodage dites algébriques ont été proposées [10, 28]. Ces algorithmes ne permettent pas de corriger plus de t erreurs dans un mot, il s'agit de décodage dit à distance bornée. Pour toutes ces méthodes, le processus de décodage vise à résoudre l'équation-clef [11] :

$$S(x)\sigma(x) \ = \ \omega(x) \bmod (x^{n-k}) \tag{1.20}$$

où $S(x)$ est le syndrome sous forme polynomiale, $\sigma(x)$ est le polynôme localisateur d'erreurs de degré $\leq t$ et $\omega(x)$ est le polynôme évaluateur d'erreurs. La résolution de cette équation nécessite trois étapes élémentaires :
– le calcul des composantes du syndrome S ;
– le calcul du polynôme localisateur d'erreurs $\sigma(x)$ et du polynôme évaluateur d'erreur $\omega(x)$;
– la détermination des positions des erreurs.
La méthode directe pour déterminer le polynôme localisateur d'erreurs utilise l'algorithme de Peterson [28] et peut corriger jusqu'à trois erreurs par mot reçu avec une complexité raisonnable $(o(t^2))$. Au delà $(t > 3)$, des méthodes itératives plus complexes sont nécessaires. Elles reposent sur l'algorithme de Berlekamp [11]. Il faut y adjoindre l'algorithme de Chien pour déterminer les racines (c'est à dire les positions des erreurs) et une vérification du mot de code élaboré. Une version modifiée et plus légère de l'algorithme de Peterson [28, 29], dite PGZ (*Peterson-Gorenstein-Zierler*) est destinée à des pouvoirs de correction plus faible typiquement $t = 1$ et $t = 2$. Dans le cas d'un code BCH binaire, la détermination des coefficients du polynôme évaluateur d'erreurs n'est pas nécessaire.

▷ **Les codes Reed-Solomon** sont des codes cycliques et plus précisément des codes BCH non binaires, ils portent le nom de leurs inventeurs Irving Reed et Gustave Solomon. Il s'agit de codes adaptés à la correction de paquets d'erreurs [30]. L'ensemble codeur/décodeur permet de construire un code cyclique et de corriger un nombre de t symboles q-aire erronés, fixés par l'algorithme de décodage, dans un bloc de n symboles q-aire codés transmis. Les codes RS font partie de la famille des codes séparables à distance maximale MDS (*Maximum Distance Separable*), ils sont optimaux au sens du critère de la distance minimale, c'est-à-dire, des codes parfaits, ou des codes maximisant la distance entre les mots codes et ainsi abaissant la probabilité de confusion lors de décision lors de décodage.
Les codes RS sont constitués de symboles q-aire (avec $q = p^m$). Généralement des éléments binaires sont considérés c'est à dire $p = 2$ et par conséquent $q = 2m$. Chaque symbole q-aire d'un code RS est représenté par un m-uplet d'éléments binaires. Le code est défini par son polynôme générateur $g(x)$ dont les coefficients sont exprimés dans $GF(q) \ = \ \{0, 1, \alpha, \alpha^2, \cdots, \alpha^{q-2}\}$.
Les caractéristiques d'un code Reed-Solomon ayant un pouvoir de correction de t symboles q-aires sont les suivantes :
– $n = q - 1, \ (m \geq 3)$ est la longeur du code.

– $k = n - 2t$ est la dimension du code.
– $\delta = 2t + 1$ est la distance de Hamming.
– $\delta = n - k + 1$

Les codes RS ont une meilleure distance minimale à rendement de codage fixé par rapport à des codes BCH binaires où $\delta \leq (n - k + 1)$.
La procédure d'un codage systématique s'effectue selon l'équation 1.12 par un polynôme générateur $G(x)$. L'équation 1.21 illustre la méthode de construction du polynôme générateur d'un code Reed-Solomon. En effet, il n'y a qu'à prendre les produits des binômes pour $(x + \alpha^i)$, pour i allant de 1 jusqu'à $2t$. On note que dans cette construction, les α^i correspondront aux racines du polynôme $G(x)$, et les g_i sont des symboles membre de $GF(2^m)$.

$$G(x) \ = \ (x + \alpha)(x + \alpha^2) \cdots (x + \alpha^{2t}) \tag{1.21}$$

α étant un élément primitif dans $GF(2^m)$.

Décodage des codes RS : Comme pour le décodage des codes BCH binaires, le décodage revient à résoudre l'équation-clef 1.20. Dans ce cas, la résolution de cette équation nécessite quatre étapes élémentaires :
– le calcul des composantes du syndrome S ;
– le calcul du polynôme localisateur d'erreur $\sigma(x)$ et du polynôme évaluateur d'erreur $\omega(x)$;
– la détermination des positions des erreurs ;
– le calcul des amplitudes des erreurs. La valeur de l'erreur est fournie sur q bits, il s'agit de la correction à apporter au symbole erroné.

Classiquement, la correction des codes RS intervient sur un ou plusieurs octets $(t = 8 \, et \, GF(2^8))$.
Les processus de décodage utilisent les mêmes algorithmes tels que celui de Berlekamp pour calculer le polynôme localisateur d'erreurs et de Chien pour déterminer les racines. L'algorithme PGZ est le plus adapté pour ces faibles pouvoirs de correction (un ou deux symboles).
Les symboles d'un code Reed-Solomon étant q-aires, la localisation des erreurs n'est pas suffisante comme dans le cas des codes BCH. Il est nécessaire de pouvoir estimer également leurs amplitudes afin d'effectuer les corrections associées.
Un autre algorithme similaire à Berlekamp c'est l'algorithme d'Euclide. C'est un algorithme récursif qui permet de localiser les erreurs σ et de calculer leurs amplitudes ω, il est basé sur le calcul d'un PGCD (plus grand diviseur commun) entre deux polynômes $r_0(t)$ et $r_1(t)$ qui sont initialisé à x^{2t} et $S(x)$ respectivement. Une suite de divisions successives est réalisée dans le «champ de Galois» $GF(q)$ jusqu'a ce que le degré du reste soit inferieur à t. La procédure de calcul du polynôme de localisation des erreurs $\sigma(x)$ et le polynôme d'amplitude $\omega(x)$ selon l'algorithme d'Euclide se trouvent dans [208].

1.3.3 Codes convolutifs

Les codes convolutifs, introduits en 1955 par Elias, constituent une classe extrêmement souple et efficace de codes correcteurs d'erreurs. Ce sont les codes les plus utilisés dans les systèmes de télécommunications fixes et mobiles. Théoriquement, ils ont les mêmes caractéristiques que les codes en blocs sauf pour leurs valeurs de dimension et longueur. Les codes convolutifs s'appliquent sur des séquences infinies de symboles d'information et génèrent des séquences infinies de symboles codés.

Un code convolutif diffère d'un code bloc par le fait que chaque bloc de n éléments en sortie ne dépend pas seulement des k entrées à un instant donné mais aussi des m blocs précédents (fig 1.12).

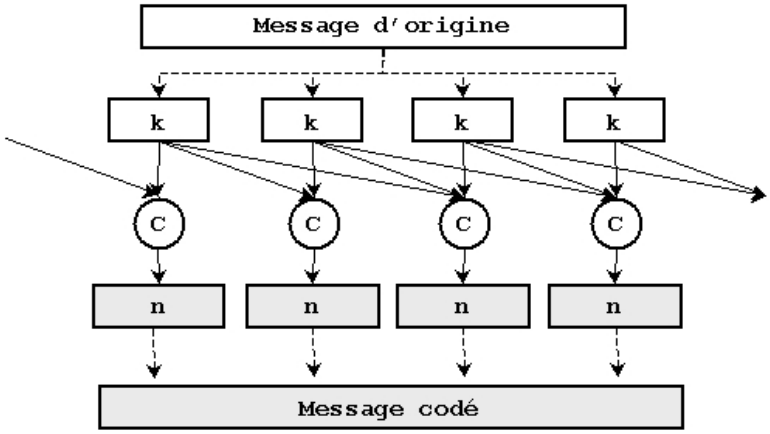


FIG. 1.12 – Codage convolutif

Toutes les propriétés des codes en blocs s'appliquent à ce type de codage ainsi qu'à l'opération inverse de décodage. La contrainte principale du décodage convolutif réside dans le fait que le mot de code est très long, ce qui a tendance à compliquer le circuit décodeur. Les algorithmes de décodage les plus répandus sont :

– **le décodage séquentiel** où le nombre d'itérations nécessaires pour le décodage est considéré comme une variable aléatoire : bien que la plupart des trames soient décodées rapidement, il se peut que le décodage soit défectueux ou la cause d'effacements, ce qui lui vaut l'appellation de méthode probabilistique de décodage. Ce type de décodage est approprié au cas des canaux sans mémoire [31].

– **L'algorithme de Viterbi** proposé en 1967 par A. J. Viterbi [32], il est basé sur le principe du maximum de vraisemblance. Cet algorithme est une méthode optimale de décodage pour les codes convolutifs, ses performances dépendant de la qualité du canal de transmission utilisé. Par contre la complexité des systèmes de décodage augmentant exponentiellement avec la longueur de contrainte du code utilisé restreint leur emploi aux applications où le code a une petite longueur de contrainte. Il est utilisé dans les applications des systèmes sans fil [33–36] : les modems séries V.3x, GSM, satellite DVB, TV portable comme le DVB-H *(Digital Video Broadcasting - Handhelds)* qui a émergé de Nokia et normalisée par le groupe européen ETSI

European Telecommunications Standards Institute sous le norme EN 302,304. Décodage par seuil (logique majoritaire).

– **Le décodage par logique majoritaire** appliqué aux codes convolutifs a été proposé par Massey en 1963 [6]. Il diffère des décodages de Viterbi et séquentiel par le fait que la décision finale prise pour un bloc donné est seulement basé sur la longueur de contrainte du bloc en cours de décodage plutôt que sur l'utilisation de toute la séquence reçue, ce qui conduit à des performances de décodage inférieures aux deux autres méthodes. Ses points forts résident dans la simplicité de son implantation et sa rapidité de décodage. On le retrouve dans des applications telles que la téléphonie ou la radio HF [37]. Il est aussi utilisé dans les systèmes de télécommunication à haut débit [6].

Représentation des codes convolutifs

Un codeur convolutif [8] non récursif peut être vu comme un filtre à réponse impulsionnelle finie (FIR : *Finite Impulse Response*) possédant k entrées et n sorties. Son état est déterminé par le contenu de ses registres à décalage. En pratique, un codeur convolutif peut être considéré comme une machine d'état et être représenté par un diagramme d'état, une structure en arbre ou une représentation en treillis.

Diagramme d'état

Le diagramme d'état (fig.1.13) est une représentation du fonctionnement du codeur ne faisant pas apparaître explicitement le temps.

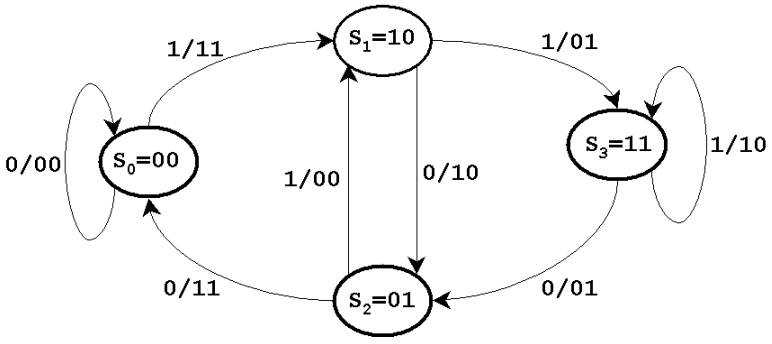


FIG. 1.13 – Exemple d'un diagramme d'état

Il représente les transitions possibles entre les états. Les valeurs des sorties du codeur sont indiquées sur chacune des transitions. Tous les états internes possibles du codeur sont représentés par des nœuds S_j. Pour un codeur de rendement $1/n$ possédant une mémoire de taille m, il existe 2^m états

[8] Le terme « codeur convolutif » indique par défaut le cas non récursif sinon il sera mentionné explicitement que le codeur est récursif, auquel cas il est considéré comme un filtre à réponse impulsionnelle infinie IIR (*Infinite Impulse Response*).

internes possibles. Chaque nœud est connecté à un autre via une branche et le passage se fait par une transition y/x_0x_1, où y correspond au bit d'entrée et x_0x_1 représente la séquence correspondante en sortie.

Structure en arbre

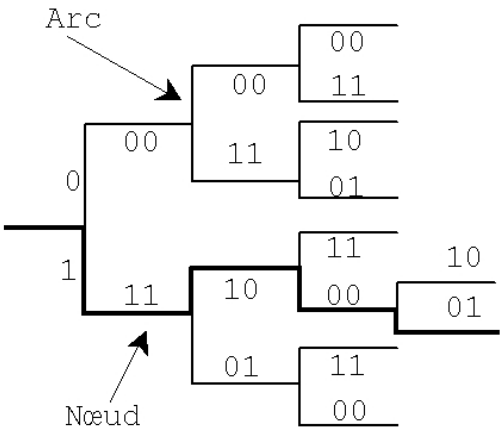


FIG. 1.14 – Exemple d'une structure en arbre $(R = 1/2,\ m = 2)$

Un arbre (fig. 1.14) est une structure partant d'un point appelé racine, il se compose d'arcs et de nœuds. Les arcs sont des traits verticaux dont le sens est déterminé par le bit d'information. Par convention, le 0 est représenté par un arc montant et le 1 par un arc descendant. Les nœuds sont des traits horizontaux indexés par les n sorties correspondant au bit d'entrée. Le chemin en gras sur la figure 1.14 correspond au mot de code 11100001 généré par la séquence d'entrée 1011.

Représentation en treillis

Contrairement aux deux précédentes, la représentation en treillis met en évidence le paramètre temporel. Chaque nœud $S_{(j,i)}$ correspond à un état particulier S_j du codeur à un instant i, où i représente l'indice du temps. Chaque branche est indexée par les bits qui se présentent en entrée et en sortie du codeur e/s. Chaque mot de code est associé à un chemin unique du treillis qu'on appelle « séquence d'état », dénotée par $\mathbf{s} = (s_0, \cdots, s_L)$. Un chemin complet commence à l'état $s_0 = S_{(0,0)}$ et se termine à l'état $s_L = S_{(0,L)}$.

Code convolutif de rendement 1/n

Dans le cas d'un codeur convolutif de rendement $1/n$, la taille m de la mémoire devient celle du registre unique du codeur. Le nombre maximum de bits associés à une sortie pouvant être affectés par un bit quelconque à l'entrée est appelé longueur de contrainte K_c du codeur. La longueur de contrainte

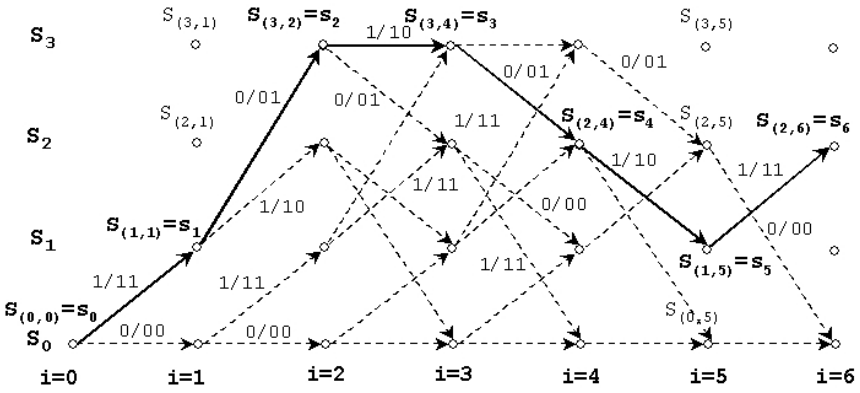


FIG. 1.15 – Exemple d'une représentation en treillis $(R = 1/2,\ m = 2)$

est $K_c = 1 + m$ et chaque bit de parité se calcule suivant l'équation (1.22).

$$c_i^{(s)} = \sum_{j=0}^{m} d_{i-j} \cdot h_{sj} \tag{1.22}$$

Le mot de code C est formé en muliplexant les sorties $C^{(s)}$

$$C = \text{Fmux}[C^0, C^1, \cdots, C^{n-1}] = (c_0^{(0)} c_0^{(1)} \cdots c_0^{(n-1)}, c_1^{(0)} c_1^{(1)} \cdots c_1^{(n-1)}, \cdots) \tag{1.23}$$

où $C^i = (c_0, c_1, c_2, \cdots)^{(i)}$ avec $i \in \{0, 1, \cdots, n-1\}$.

La figure 1.16 représente un codeur convolutif de rendement $R = 1/2$ et de mémoire $m = 3$ pour les polynômes générateurs $H_0 = h_{00} + h_{01}x + h_{02}x^2 + h_{03}x^3$ et $H_1 = h_{10} + h_{11}x + h_{12}x^2 + h_{13}x^3$. $(X)_t$ contenant les valeurs d'entrée d_i et les sorties $Y^{(0)}$ et $Y^{(1)}$ contiennent respectivement les valeurs $C^{(0)}$ et $C^{(1)}$ de l'équation 1.23.

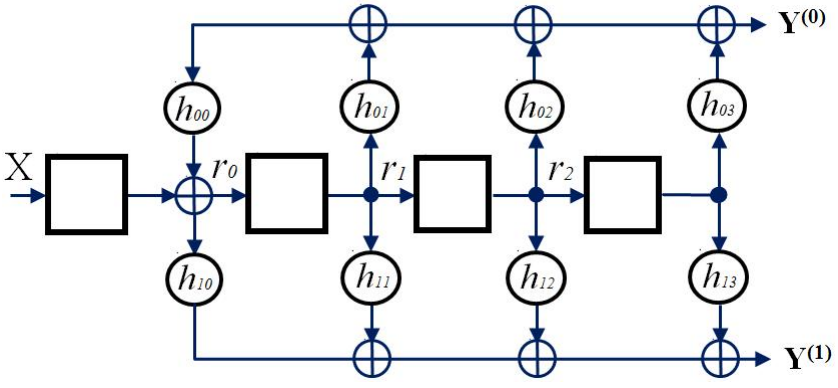


FIG. 1.16 – Exemple d'un codeur convolutif $(R = 1/2,\ m = 3)$.

Le codeur doit être initialisé à zéro avant chaque opération de codage. Les bits d'information sont ensuite propagés à travers les registres. En sortie, les bits de codage sont recueillis toujours dans le même ordre (équ. (1.23)). À la fin on complète le message d'information par les m bits de queue pour la terminaison du treillis.

La structure d'un codeur convolutif de rendement $1/n$ est similaire à celle d'un multiplieur. En effet, le codeur convolutif se comporte comme n multiplieurs dont chacun multiplie la même séquence par son propre polynôme générateur $H_i(x)$. Le chemin critique du système dépend du polynôme générateur qui possède le plus de coefficients non nuls. Les codes convolutifs peuvent, comme les codes blocs, devenir systématiques en dédiant une sortie aux bits d'information.

Codes convolutifs récursifs

La récursivité est obtenue en appliquant une boucle de retour au codeur. Deux architectures du codeurs convolutif sont possibles MTO [9] et OTM [10].

▷ **Architecture série MTO (*Many To One*) :** La forme canonique série de l'architecture MTO d'un codeur convolutif avec une sortie de parité est représentée sur la figure. 1.17. La récursivité est obtenue en appliquant une boucle de retour au codeur. Les entrées et les sorties à l'instant t sont égales à $(X)_t$ et $(Y)_t$ respectivement.

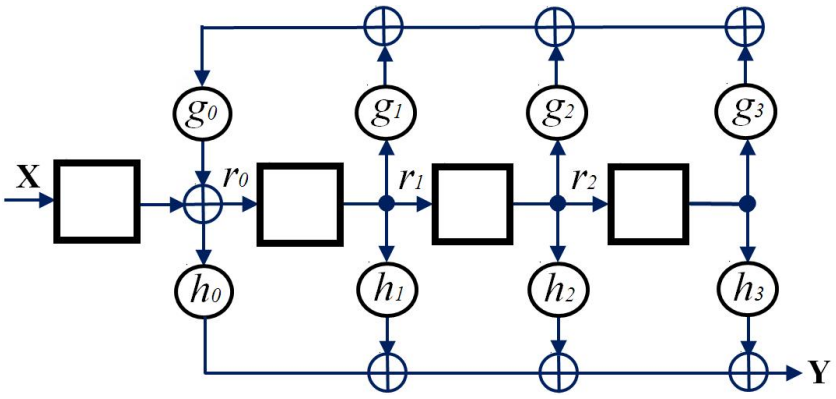


FIG. 1.17 – Forme canonique du codeur convolutif MTO pour $m = 3$.

La mémoire du codeur dans l'exemple est $m = 3$. À partir de cette forme canonique, deux configurations du codeur MTO sont possibles. Un codeur convolutif non-résursif (CNR) où un codeur convolutif récursif (CR). Le choix du type de codeur (CNR or CR) sera fait d'après l'équation (1.27).

$$g_0 = 1 \Rightarrow \begin{cases} (g_1, g_2, g_3) = 0, & \text{codeur CNR} \\ (g_1, g_2, g_3) \neq 0, & \text{codeur CR} \end{cases} \tag{1.24}$$

A chaque cycle d'horloge, le codeur transite vers un nouvel état en fonction de l'état précédent et du bit d'information présent en entrée. Il fournit simultanément un nouveau bit codé. Cette opération est décrite par les équations architecturales (1.25) et (1.26) [50].

[9] Many To One
[10] One To Many

$$
\begin{aligned}
(r_0)_{t+1} &= (u)_t + g_0 \cdot [g_1 \cdot (r_0)_t + g_2 \cdot (r_1)_t + g_3 \cdot (r_2)_t] \\
(r_1)_{t+1} &= (r_0)_t \qquad (1.25) \\
(r_2)_{t+1} &= (r_1)_t \\
(Y)_{t+1} &= h_0 \cdot [(X)_t + g_0 \cdot (g_1 \cdot (r_0)_t + g_2 \cdot (r_1)_t + g_3 \cdot (r_2)_t)] \qquad (1.26) \\
&\quad + h_1 \cdot (r_0)_t + h_2 \cdot (r_1)_t + h_3 \cdot (r_2)_t
\end{aligned}
$$

où r_i représente la bascule i du registre R, g_i et h_i sont respectivement les coefficients du polynôme de la boucle de retour $G(x)$ et du polynôme de la branche directe $H(x)$. Y étant la sortie du codeur MTO série.

▷ **Architecture série OTM (*One To Many*) :** De la même façon, l'architecture série d'un codeur convolutif de type OTM, est représentée sur la figure 1.18 pour une taille de mémoire $m = 3$. Le choix du type de codeur (CNR or CR) sera fait d'après l'équation (1.27). $(X)_t$ et $(Y)_t$ sont les entrées/sorties du codeur à l'instant t.

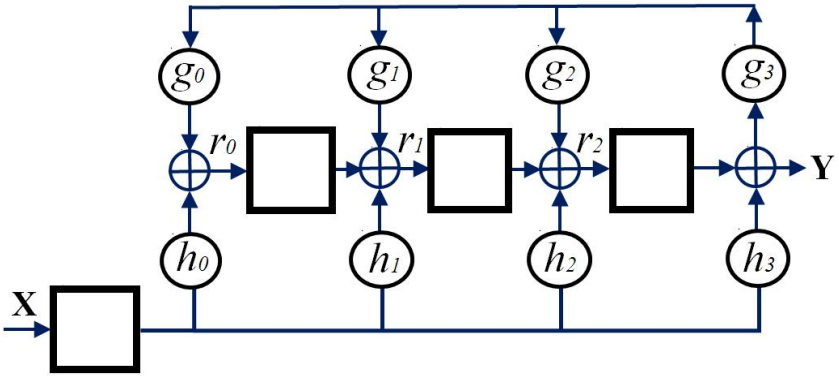


FIG. 1.18 – Forme canonique du codeur convolutif OTM pour $m = 3$.

$$
g_3 = 1 \Rightarrow \begin{cases} (g_0, g_1, g_2) = 0, & \text{codeur CNR} \\ (g_0, g_1, g_2) \neq 0, & \text{codeur CR} \end{cases} \qquad (1.27)
$$

Pour les équations architecturales du codeur OTM sont données par les équations (1.28) et (1.29) [50].

$$
\begin{aligned}
(r_0)_{t+1} &= h_0 \cdot (X)_t + g_0 g_3 [(r_2)_t + h_3 \cdot (X)_t] \\
(r_1)_{t+1} &= h_1 \cdot (X)_t + g_1 g_3 [(r_2)_t + h_3 \cdot (X)_t] + (r_0)_t \qquad (1.28) \\
(r_2)_{t+1} &= h_2 \cdot (X)_t + g_2 g_3 [(r_2)_t + h_3 \cdot (X)_t] + (r_1)_t \\
(Y)_{t+1} &= (r_2)_t + h_3 \cdot (X)_t \qquad (1.29)
\end{aligned}
$$

où r_i représente la bascule i du registre R, g_i et h_i sont respectivement les coefficients du

polynôme de la boucle de retour $G(x)$ et du polynôme de la branche directe $H(x)$. Y étant la sortie du codeur OTM série.

1.3.4 Codes concaténés

La concaténation de codes permet d'augmenter la puissance des systèmes de codage au prix d'une augmentation de la complexité globale, autrement dit, moins complexe qu'un codeur "simple" mais de capacité équivalente. La concaténation peut se faire de trois façons : parallèle, série ou hybride (parallèle et série) et sur deux ou plusieurs niveaux [38]. Dans le cas d'une structure série à deux codes, l'information est codée deux fois. Une première fois par le premier code appelé code externe, puis une seconde fois par le deuxième, dit code interne [8].

Les deux codes utilisés sont en général complémentaires : les codes convolutifs par exemple sont inadaptés aux erreurs qui apparaissent par paquets alors que les codes RS sont adéquats pour ce type d'erreurs. Dans ce cas, le décodeur convolutif s'occupera des erreurs aléatoires pour de faibles rapports signal sur bruit tandis que le décodeur RS s'occupera des erreurs par paquets pour des rapports signal sur bruit élevés. Ce type de concaténation série a été proposée par David Forney en 1966 [14] puis standardisé en 1987 pour les communications spatiales dans les réseaux DSN (*Deep Space Network*) par les agences spatiales NASA (*National Aeronautics and Space Agency*) et ESA (*European Space Agency*), préconisant l'utilisation d'un code RS ($q = 8,\ k = 223,\ n = 255,\ t = 16$). Il est aussi utilisé dans les systèmes de diffusion par satellite (norme DVB-S2) [39], la diffusion hertzienne terrestre (norme DVB-T) et la diffusion vidéo numérique portable DVB-H *(Digital Video Broadcasting - Handhelds)* qui est normalisée par le groupe européen ETSI (*European Telecommunications Standards Institute* [40]).

Les codes produits, inventés par P. Elias en 1954 [41], sont construits par la concaténation série de deux ou plusieurs codes en blocs linéaires à faible pouvoir de correction. En général, les codes utilisés sont les codes BCH et Hamming.

1.3.5 Turbo-codes

Les turbo-codes ont été découverts par C. Berrou et al. en 1993 [42]. Un turbo-code est la concaténation d'un ou de plusieurs codes (convolutifs ou blocs) séparés par des blocs d'entrelacements. L'entrelaceur est un élément essentiel de codage turbo, en son absence les performances se trouvent fortement affectées. Une détection par maximum de vraisemblance peut être envisagée pour le décodage des turbo codes. Le module de décodage associé à chaque code est constitué par un algorithme MAP connu sous le nom de Log MAP et Max Log MAP à entrée-sortie souples [48]. Dans [49], une classe regroupe tous les codes capables d'être décodés par un décodeur de type SISO (*Soft-Input Soft-Output*), la concaténation série proposée par Forney étant exclue de cette classe car le code RS ne peut pas être décodé par SISO.

Cette famille de codes offre de hautes performances permettant de s'approcher très près de la limite théorique de Shannon [43]. Ils marquent une révolution dans la communauté de la théorie de

l'information. Depuis leur introduction, de très nombreux travaux de recherche relatifs aux turbo-codes, et plus généralement aux codes concaténés, ont été effectués afin de comprendre et d'analyser leur comportement et d'améliorer leurs performances. Jusqu'à nos jours, aucune explication théorique et mathématique n'arrive à expliquer parfaitement les performances de cette famille des codes. Coté application, les turbo codes ont été adoptés par quelques standards émergeants telles que le DVB-RCS [44] pour les liaisons satellitaires montantes. Des schémas de codage turbo ont été aussi proposés par la comité CCSDS *(Consultative Committee for Space Data Systems)* pour les applications de télémétrie et de télémesure spatiales [45]. Enfin les turbo-codes ont fait l'objet de nombreux dépôts de brevet par France Télécom dont certains conjointement avec TDF (Télédiffusion De France).

1.4 Architectures parallèle-pipeline

Du point de vue de la conception architecturale, de nombreux codes en blocs et convolutifs ont fait l'objet d'études pour améliorer leur débit et d'autres pour diminuer la complexité du matériel en même temps. Des architectures parallèles-pipeline, par exemple, ont été proposées pour les codes CRC [53, 54, 57], les codes BCH et reed-solomon [58, 60, 61]. Des architectures parallèles-pipeline pour le codeur convolutif de type MTO et OTM ont été présentés dans [64]. Une amélioration de la technique de parallélisation pour ces architectures est proposée par nous dans [67, 68]. Une architecture haut débit pour le décodeur cyclique est proposé dans [51].

1.4.1 Codeur parallèle haut débit pour les codes CRC

Une méthode de pipeline amélioré pour la mise en oeuvre d'un CRC parallèle est proposée dans [52]. Cette méthode offre un meilleur compromis (surface, performance) que celle proposée dans [167]. Elle permet de réduire efficacement le chemin critique tout en réduisant le surcoût matériel. L'architecture série d'un CRC de polynôme générateur $G(X) = 1 + x + x^8 + x^9$ est présentée dans la figure 1.19. Le nombre d'itérations dans le boucle (la partie pointillée) est de $2T_{XOR}$, où T_{XOR} est le délai d'une porte XOR.

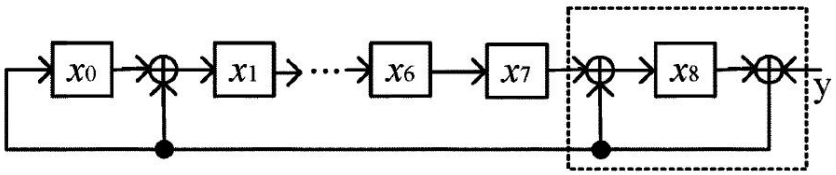


FIG. 1.19 – Architecture CRC série pour $G(X) = 1 + x + x^8 + x^9$

Afin de réduire le nombre d'itérations de la boucle de $2T_{XOR}$ à un seul T_{XOR}, une modification de l'architecture de la boucle est appliquée (voir figure 1.20). Dans cette figure, une version pipeline à 2 étages, le nombre d'itérations de *loop1* et *loop2* est de T_{XOR} et $5/8T_{XOR}$ respectivement.

Après avoir pipeliné et resynchronizé l'architecture de la figure 1.20, une nouvelle architecture à 3 étages de pipeline est presentée dans la figure 1.21. Cette figure necéssite pour l'implanter de

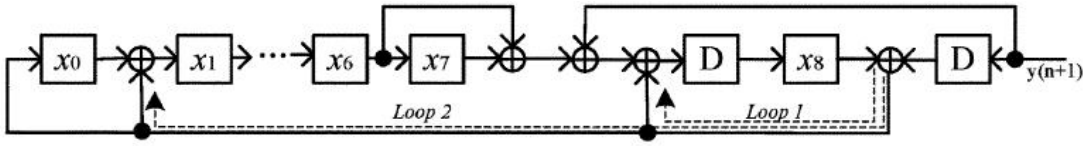


FIG. 1.20 – Architecture CRC série pour $G(X) = 1 + x + x^8 + x^9$

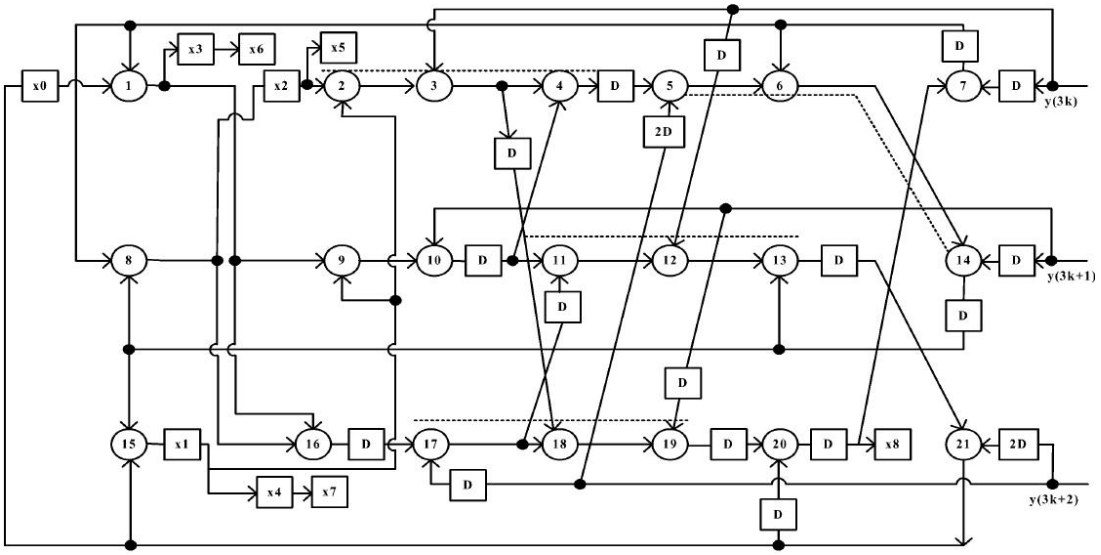


FIG. 1.21 – Architecture parallèle-pipeline d'un CRC pour $p = 3$

seulement 21 XOR pour un chemin critique reduit jusqu'a 3 XOR ; plus de détail pour l'implantation d'un CRC parallèle se trouve dans [52].

1.4.2 Décodeur parallèle-pipeline pour les codes cycliques en blocs

Dans la figure 1.22, une architecture parallèle-pipeline du décodeur cyclique (section 1.3.2) est proposée dans [51] pour les transmissions haut débit. Le fonctionnement du décodeur parallèle est décrit comme suit :

- À chaque cycle d'horloge, p bits d'entrée sont injectés dans la partie droite du décodeur pour calculer le syndrome $S_1 \cdots S_p$.
- Dès que le syndrome a été calculé, la valeur en S' est transférée dans S_0 grâce à la connexion *entrée a*. Les registres S'_i et S' sont réinitialisé, et le processus de décodage est commencé. L'opération est décrite par les équations suivantes :

$$\begin{aligned}(S'_i)_{t+1} &= (S'_{i-1})_t \cdot T + F[(S_i)_t] \cdot (1,0,0) \\ (S')_{t+1} &= (S')_t \cdot T^p + (S'_{p-1})_t\end{aligned} \tag{1.30}$$

  T étant la matrice résultant de l'équation $[(\cdots) * x] \mod G(x)$ [51].
- $p-$permutation cyclique est appliquée au registre S_0, l'erreur estimée est injectée dans les étages du pipeline S'_i, est par conséquent, p-bits d'entrée peuvent être corrigés par les résultats de p F.

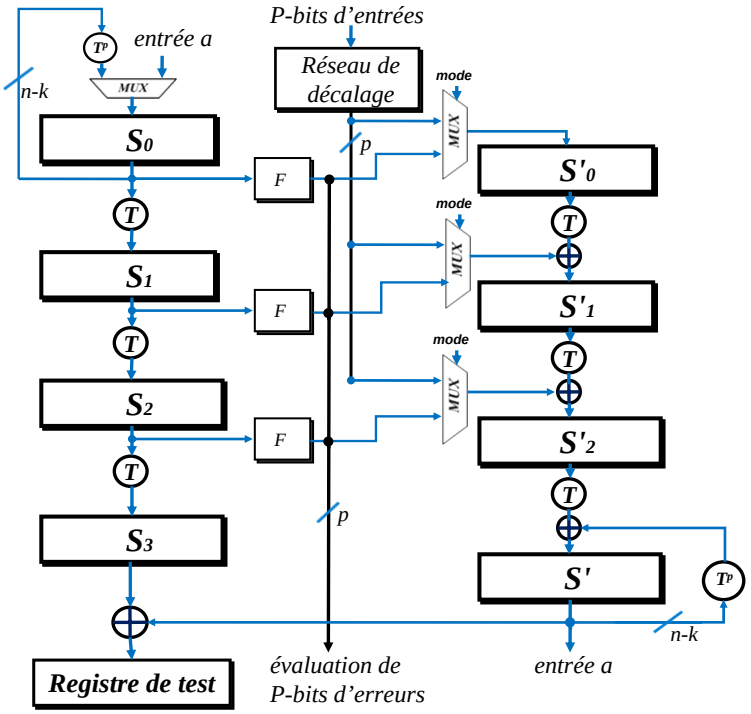


FIG. 1.22 – Architecture parallèle-pipeline du décodeur cyclique pour $p = 3$

Les équations correspondantes sont les suivantes :

$$\begin{aligned}(S_0)_{t+1} &= (S_0)_t \cdot T^p \\ (S_i)_{t+1} &= (S_{i-1})_t \cdot T, \quad i \in 1, \cdots, p-1 \\ (S)_{t+1} &= (S_{p-1})_t\end{aligned} \tag{1.31}$$

– F est une fonction de décision utilisée pour l'estimation de l'erreur ; elle peut être une fonction de la détection de l'erreur EPD (*Error Pattern Detection*) ou la fonction à logique majoritaire MLV (*Majority Logic Voting*) [6].
– Un multiplexeur est utilisé dans la partie droite du décodeur pour choisir l'entrée dans le mode de calcul de syndrome ou dans le mode de décodage.

À la fin de décodage, les valeurs S_p et S' doivent être équivalentes, et par conséquent la valeur enregistrée dans le *Registre de test* doit être égale à zéro.

Cette architecture permet d'atteindre avec succès un débit de données très élevés. En dépit de ce succès, le fonctionnement de cette dernière n'est pas garanti contre les fautes transitoires SEU. Supposons par exemple, lors de la phase de calcul de syndrome, une faute SEU affecte l'un des registres S'_i, alors à la fin de cette phase une valeur erronée de S' est transférée dans S_0 par la connexion *entrée a*. Donc, le *Registre de test*, utilisé pour nous indiquer la fin de décodage, est toujours différente de zéro ($S_p \neq S'$), et par conséquence la phase de décodage ne termine jamais. Une amélioration de cette architecture est proposée dans de cette thèse pour la rendre FS (*Fault-Secure* - dont la définition sera présentée prochainement) afin de contrôler le fonctionnement de cette dernière.

1.4.3 Achitecture de décodeur RS rapide et à faible complexité

Une architecture de décodeur Reed-Solomon (RS) haut débit et à faible complexité a été présentée dans [61]. Cette architecture permet de résoudre l'algorithme d'Euclide récursif modifié PrME dans le but de trouver le polynôme localisateur d'erreurs $\sigma(x)$ et le polynôme évaluateur d'erreur $\omega(x)$ (équation 1.20). La figure 1.23 montre le schéma bloc de l'architecture pipeline de l'algorithme PrME. 4 blocs sont nécessaires pour l'implanter : unité de mesure de degrée, unité arithmétique polynômiale, unité de détection de degrée parallèle, et un réseau de registres à décalage connectés par différentes boucles récursives.

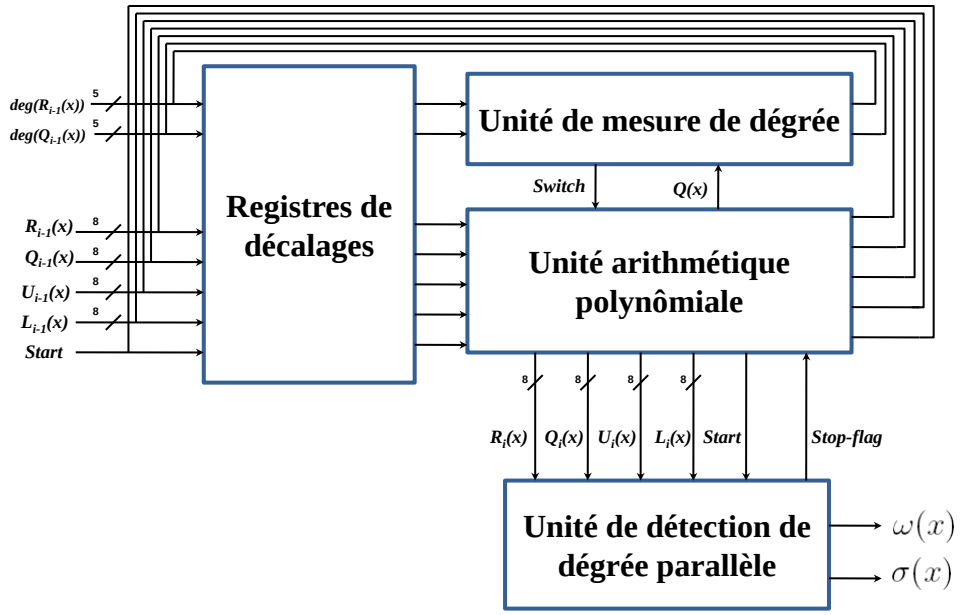


FIG. 1.23 – Schéma bloc d'un décodeur RS haut débit

La structure récursive permet de mettre en oeuvre l'algorithme PrME avec une faible complexité. Le pipeline et la parallélisation permettent de recevoir l'entrée à un très haut taux de fibre optique, et les résultats sont livrés à des taux élevés de minimum de délai. Dans [61], un calendrier de temps (*timing chart*) est appliqué au décodeur RS, après n cycles d'horloge les polynômes $\sigma(x)$ et $\omega(x)$ sont fournis en parallèles au bloc Chien pour touver les racines. Les résultats sont obtenus avec une latence fixe de $2n + 12$ cycles d'horloge. Le décodeur RS pipeline performe à un débit de $80\mathrm{Gb/s}$ tout en réduisant la complexité. Cette architecture à été implantée en utilisant la technologie CMOS de $0,13\mu m$ à tension d'alimentation de $1,2\mathrm{V}$, pour un nombre de portes logiques de $393K$ et à une fréquence d'horloge de $625\mathrm{MHz}$.

1.4.4 Codeur parallèle haut débit pour les codes convolutifs

Le schéma fonctionnel du codeur haut débit est représenté sur la figure 1.24, il est constitué de trois blocs [50]. Le bloc de génération des états anticipés (GEA) fournit à chaque cycle d'horloge un

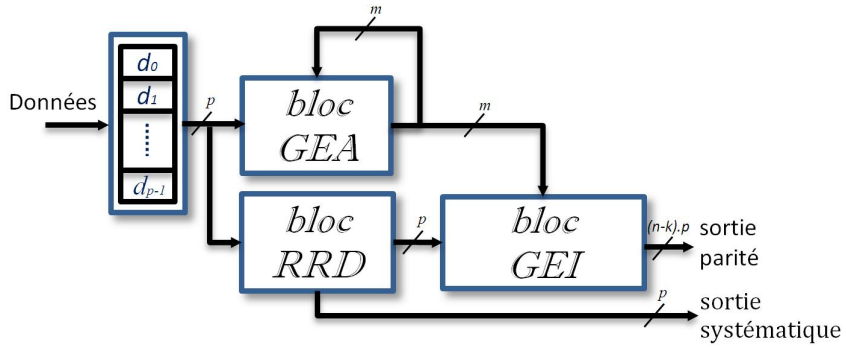


FIG. 1.24 – Schéma fonctionnel du codeur systématique parallèle

nouvel état anticipé $E_{k\cdot p+p}$ calculé à partir de l'état $E_{k\cdot p}$ précédent et de p bits des données présents à l'entrée, où $E_{k\cdot p+p}$ et $E_{k\cdot p}$ sont les valeurs du registre R du codeur série au temps $k \cdot p + p$ et $k \cdot p$ respectivement. Le bloc de génération des états intermédiaires (GEI) présente une structure pipeline de p-étages, constituée de codeurs série connectés entre eux selon une topologie prédéfinie. Chaque

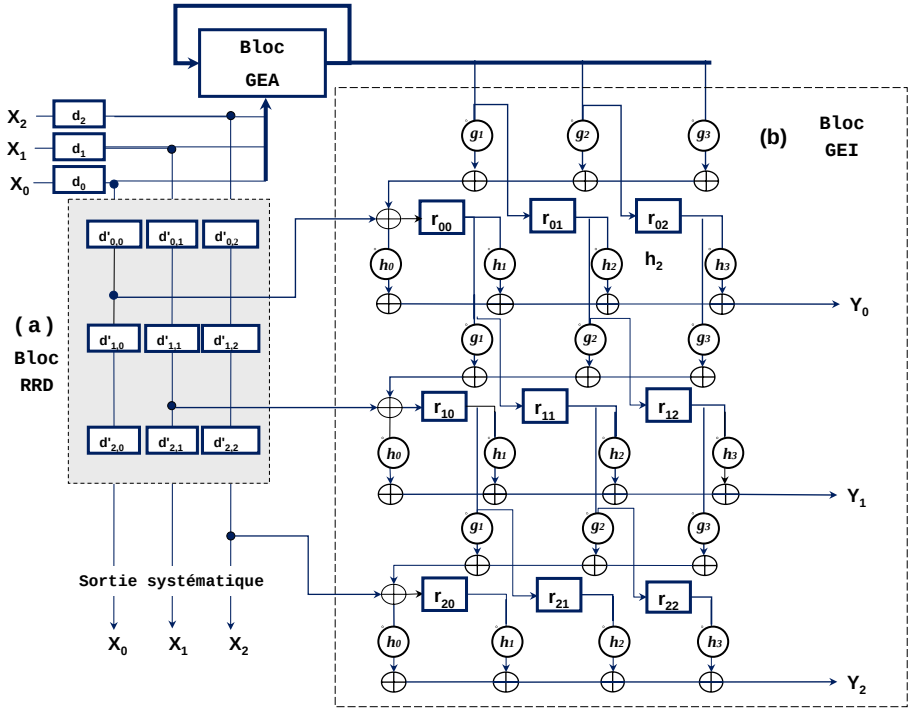


FIG. 1.25 – Codeur parallèle MTO pour $R = 1/2$ et $m = 3$

codeur, représentant un étage du pipeline, permet le calcul d'un état intermédiaire et, en fonction du rendement du codeur, la génération d'un ou de plusieurs bits codés. Le réseau de registre a décalage (RRD) permet de décaler les données entrantes qui sont prélevées dans le réseau selon un ordre précis avant d'être injectées dans le bloc GEI.

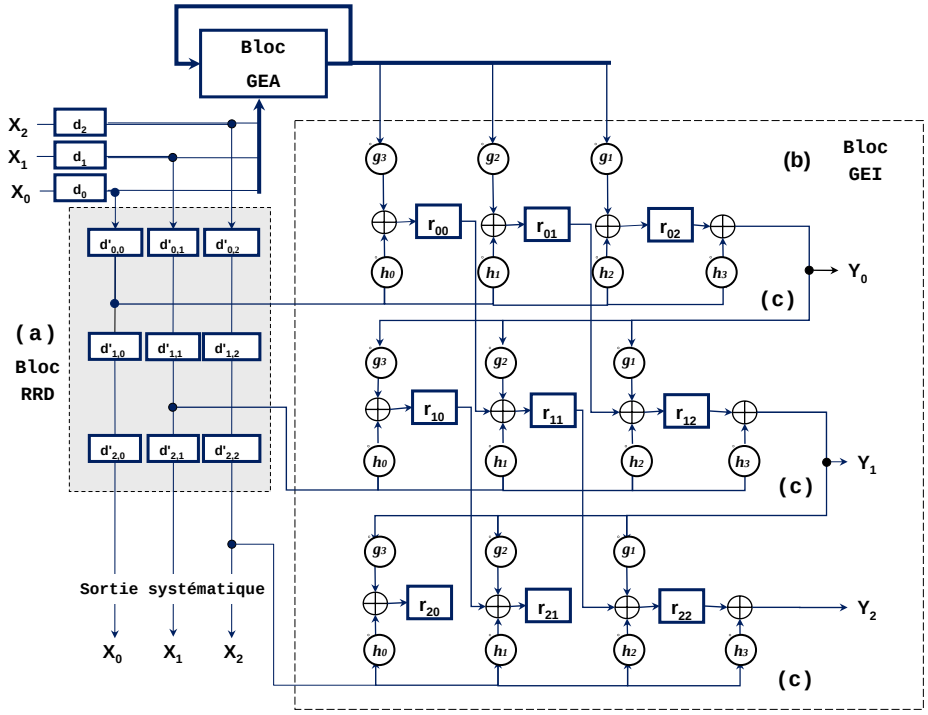


FIG. 1.26 – Codeur parallèle OTM pour $R = 1/2$ et $m = 3$

La figure 1.24 résume le flot de conception du codeur parallèle. Les architectures parallèle-pipeline CP_{mto} et CP_{otm} pour $m = 3$ des codeurs convolutifs séries (voir section 1.3.3) sont représentées sur les figures 1.25 et 1.26, respectivement. Les équations architecturales du codeur OTM et MTO se trouvent en détails dans [64].

Ces architectures permettent d'atteindre un débit important du codeur convolutifs pour les 2 cas MTO et OTM. Malgré ce succès, elles souffrent d'une part de l'augmentation de la surface consommée lorsque le degrée de parallélisation p devient assez élevé (en particulier $p = 32$), d'autre part, la prévention d'un changement en ligne des coefficients des polynômes G et H, puisqu'à chaque nouvel implantation, un pré-calcul des coefficients des matrices est nécessaires, particulièrement dans le bloc GEA.

1.5 Conclusion

Dans ce chapitre, nous avons introduit le domaine d'application de notre étude. Ainsi, les codes en blocs et les convolutifs ont été étudiés dans le cadre de la théorie du traitement de l'information et plus spécifiquement dans le domaine des codes correcteurs d'erreurs. Les codes en blocs linéaires cycliques et plus particulièrement les codes CRC, BCH, et Reed-Solomon q-aires ont été exposés. Puis, les codes convolutifs ont été introduits ainsi que leurs propriétés. De même, une étude approfondie sur les codes convolutifs récursifs de type MTO et OTM a été effectuée.

Des architectures séries de codeur et décodeur ont été présentées pour les deux types de codes.

Citons particulièrement, le décodeur à logique majoritaire pour les codes cycliques en blocs ainsi que les structures MTO et OTM de codeurs convolutifs et leurs implantations respectives. Comme nous nous intéressons par la suite à la conception d'architectures rapides, des techniques parallèles-pipeline ont été abordées dans ce chapitre, utilisées depuis de nombreuses années pour améliorer le débit de fonctionnement.

L'objectif de notre travail est non seulement de concevoir des architectures de codeur/décodeur rapides mais également de les sécuriser, autrement dit, de les rendre sûres de fonctionnement. Dans le chapitre qui suit, nous introduisons les concepts de base et la terminologie de la sûreté de fonctionnement nécessaires à la conception de circuits numériques sécurisées.

Chapitre 2

Sûreté de fonctionnement et tolérance aux fautes

2.1 Introduction

Avant d'aborder l'aspect pratique de la conception d'architectures sécurisées, nous nous intéresserons dans un premier temps à donner des définitions précises concernant la terminologie et les concepts de base de la sûreté de fonctionnement.

La sûreté de fonctionnement (SdF) d'un système peut être définie comme étant la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre. Elle peut être analysée selon différentes propriétés complémentaires : fiabilité, disponibilité, sécurité, maintenabilité, confidentialité-intégrité [73, 76]. Ainsi, le système n'est plus sûr à partir du moment où le service qu'il fournit diverge du service attendu par l'utilisateur.

Nous poursuivrons ensuite l'étude avec la définition des moyens complémentaires permettant de concevoir des systèmes sûrs de fonctionnement, en nous focalisant en particulier sur la tolérance aux fautes autour de laquelle s'articulent nos travaux (de thèse). La tolérance aux fautes permet à un système de continuer à délivrer un service conforme à sa spécification en présence de fautes. Dans cette partie nous identifions les principales méthodes existantes et leurs caractéristiques pour rendre un système tolérant aux fautes. Finalement, pour évaluer l'efficacité des stratégies de tolérance appliquées, nous présenterons les techniques d'injection de fautes utilisées et poursuivrons avec une discussion sur les modèles de fautes qui peuvent/doivent être injectées pour valider les propriétés de tolérance aux fautes de ces architectures.

2.2 Concepts de base

Cette section a pour objectif d'introduire les principaux concepts de la sûreté de fonctionnement et d'introduire la tolérance aux fautes comme un moyen pour construire un système sûr de fonctionnement. Les propriétés (caractéristiques) de la tolérance aux fautes seront présentées dans cette section

ainsi que les méthodes d'évaluation correspondantes. L'objectif est de donner des définitions précises caractérisant les principes qui entrent en jeu dans la sûreté de fonctionnement pour les systèmes numériques. Un bref rappelde toutes les notions définies dans [73, 74, 76] est donné. Le lecteur pourra se référer à ces travaux pour une description plus détaillée de l'ensemble du domaine.

Dans cette section nous définissons les notions de base de la sûreté de fonctionnement, les attributs et les moyens et les entraves, ainsi que la mise en œuvre et l'analyse.

2.2.1 Service d'un système

Un *système* est une entité qui interagit avec d'autres entités, donc d'autres systèmes qui constituent l'environnement du système considéré, y compris le matériel, le logiciel, les humains.

La *fonction* d'un système est ce à quoi il est destiné. Elle est décrite par la spécification fonctionnelle, qui inclut les performances attendues du système. Le comportement d'un système est ce que le système fait pour accomplir sa fonction, et est représenté par une séquence d'états.

Le *service* délivré par un système est son comportement tel que perçu par ses utilisateurs (séquence d'états externes), un utilisateur est un autre système, éventuellement humain, qui interagit avec le système considéré.

2.2.2 Sûreté de fonctionnement

La *sûreté de fonctionnement* (SdF) d'un système est son aptitude à délivrer un service de confiance justifiée [74] ; elle regroupe les activités d'évaluation de la fiabilité, de la maintenabilité, de la disponibilité et de la sécurité (FMDS) d'une organisation, d'un système, d'un produit ou d'un moyen. Ces évaluations permettent, par comparaison aux objectifs ou dans l'absolu, d'identifier les actions de construction (ou amélioration) de la sûreté de fonctionnement de l'entité. Elles sont prédictives et reposent essentiellement sur des analyses inductives ou déductives des effets des pannes, dysfonctionnements, erreurs d'utilisation ou agressions de l'entité.

La SdF est un ensemble d'outils et de méthodes qui permettent, dans toutes les phases de vie d'un produit, de s'assurer que celui-ci va accomplir la (les) mission(s) pour laquelle (lesquelles) il a été conçu, et ce, dans des conditions de FMDS bien définies. Elle doit être prise en compte tout au long du cycle de vie du produit.

Villemeur [76] définit la sûreté de fonctionnement comme la science des défaillances. Elle inclut ainsi leur connaissance, leur évaluation, leur prévision, leur mesure et leur maîtrise. Au sens strict, elle est l'aptitude d'une entité à satisfaire une ou plusieurs fonctions requises dans des conditions données.

Selon Laprie [74], la notion de sûreté de fonctionnement *(dependability)* d'un système est la «propriété qui permet de placer une confiance justifiée dans le service qu'il délivre». En d'autres termes c'est la propriété, lorsque le système est mis en oeuvre, de se comporter de façon nominale, de rendre un service conformément à une référence prédéterminée. La propriété de sûreté de fonctionnement

est associée à plusieurs attributs [1], les principaux étant :

- disponibilité *(availability)* : le pourcentage du temps pendant lequel le système est prêt à l'utilisation est supérieur à un certain seuil,
- fiabilité *(reliability)* : la continuité de service est assurée pendant une durée minimale,
- innocuité *(safety)* : la probabilité d'occurrence de défaillances jugées critiques est inférieure à un certain seuil,
- maintenabilité *(maintenability)* : l'aptitude d'une entité à être maintenue ou remise en état de fonctionnement, état dans lequel elle peut accomplir une fonction requise.

En effet, le niveau de sûreté d'un circuit dépend de l'environnement dans lequel il se trouve (exposition aux perturbations), et également de la tâche qu'il doit effectuer, pour les erreurs qui sont principalement introduites lors de la conception, un travail important de *vérification* et de *validation* doit être mené tout au long du cycle de développement. La vérification consiste à s'assurer qu'un système a été construit correctement (*bien* construire le système) alors que la validation consiste à s'assurer que le système est conforme aux spécifications (construire le *bon* système). Vérification et Validation (V & V) sont donc deux activités intimement liées.

L'objectif principal de la vérification est de révéler les fautes de conception. Ces fautes peuvent être introduites durant n'importe quelle phase du cycle de développement, il est néanmoins important de les révéler le plus tôt possible, pour des raisons de coût et d'efficacité. Le développement d'un système sûr de fonctionnement passe par l'utilisation combinée d'un ensemble de méthodes qui peuvent être classées en :

- *prévention de fautes* : comment empêcher l'occurrence ou l'introduction de fautes,
- *tolérance aux fautes* : comment fournir un service conforme aux spécifications en dépit des fautes,
- *élimination de fautes* : comment réduire la présence (nombre, sévérité) des fautes,
- *prévision de fautes* : comment estimer la présence, la création et les conséquences des fautes.

La validation recouvre les deux activités, l'élimination et la prévision des fautes, qui peuvent se regrouper sous le concept d'*évitement des fautes* ; la première c'est comment minimiser, par *vérification*, la présence de fautes, et la deuxième comment estimer, par *évaluation*, la présence, la création et les conséquences de fautes.

L'élimination des fautes comporte trois étapes : vérification, diagnostic et correction. La vérification consiste à déterminer si le système satisfait à des propriétés générales ou spécifiques, appelées conditions de vérification [77]. Si des propriétés ne sont pas vérifiées, les deux autres étapes sont entreprises et une nouvelle phase de vérification doit être réalisée afin de s'assurer que l'élimination des fautes ne s'est pas accompagnée de la création de nouvelles fautes. Il existe deux classes de vérification, en fonction de l'exécution ou non du logiciel : statique ou dynamique [73] (cf. figure 2.1).

Parmi les techniques de vérification citons le "model-checking" (vérification de modèles) [78–80]

[1] Parmi ces attributs il existe d'autre tels que la confidentialité et l'integrité.

qui est une technique de vérification formelle qui repose sur une idée simple : si on énumère toutes les situations possibles auxquelles peut mener le système, on pourra s'assurer qu'aucune de ces situations n'est en contradiction avec les comportements désirés sous l'hypothèse qu'il n'y a pas de panne. Le "model-checking" est basé sur deux modèles : un modèle du système représentant tous les états possibles du programme à vérifier et un modèle des propriétés que ce dernier est censé préserver. La technique de vérification consiste alors à prouver la cohérence de deux modèles, et a pour avantage de fournir en général un contre exemple lorsqu'il détecte la violation d'une propriété par le système. Ceci facilite grandement la compréhension et la correction des erreurs.

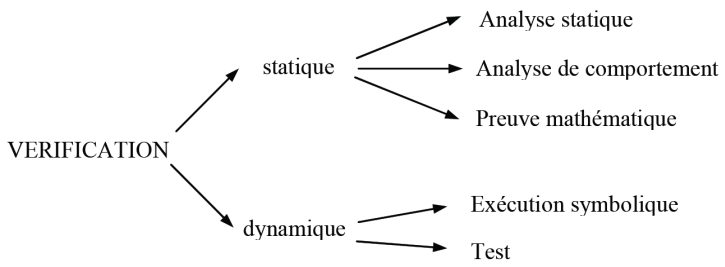


FIG. 2.1 – Les techniques de vérification [73]

Une autre technique, la plus couramment utilisée à PSA Peugeot Citroën est certainement le test [75]. Il consiste à exécuter un programme en lui fournissant des valeurs numériques : les entrées de test. Classiquement, on distingue trois types de test pendant le développement du logiciel :

- **le test unitaire** : il permet de vérifier les fonctions élémentaires indépendamment de leur environnement,
- **le test d'intégration** : il permet de vérifier un ensemble de fonctions et de leur intégration,
- **le test système** : la vérification concerne l'ensemble du système matériel/logiciel et est effectué sur banc de test.

Une analyse plus fouillée de ces définitions ainsi que des rôles et des liens entre élimination de fautes et prévision de fautes peuvent être trouvées dans [73, 74, 81].

La tolérance aux fautes et la prévision des fautes peuvent se regrouper sous le concept d'*acceptation des fautes* : partant du principe qu'il y a toujours des fautes qu'on ne peut pas éviter, on essaie d'évaluer leurs impacts sur le système et de réduire la sévérité des défaillances qu'elles peuvent causer (si possible jusqu'à la suppression des défaillances). Un moyen pour augmenter la fiabilité d'un système est la tolérance aux fautes. Celle-ci consiste à empêcher qu'un défaut de la structure ne «dégénère» en défaillance [2], il peut alors être confinée, corrigé, ou bien l'élément défaillant peut être remplacé par un élément redondant suite à une reconfiguration du système.

La *reconfiguration* est une modification de la structure d'un système défaillant (figure 2.2), de telle sorte que les composants non défaillants permettent de délivrer un service acceptable bien que dégradé (fonctionner en mode dégradé, c'est tenter de fournir le service jugé indispensable, en manquant de

[2]La tolérance s'applique aussi aux situations où la cause de l'erreur est externe, par exemple le modèle SEE (*Single Event Effect*) incluant les modèles SET (*Single Event Transient*) et SEU (*Single Event Upset*).

ressources complètes ou fiables).

Deux types de redondance peuvent être distingués [82] :

- **la redondance statique** : le nombre d'éléments redondants peut être important (redondance massive) et chacun d'entre eux participe à la réalisation de la fonction,
- **la redondance dynamique** : l'élément redondant ne participe à la fonction qu'après détection et réaction à l'erreur.

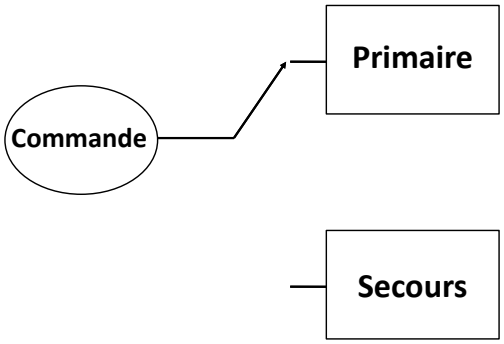


FIG. 2.2 – Reconfiguration dynamique

L'exemple de la figure 2.2 représente un système reconfigurable. En effet, la commande permet de reconfigurer le système en utilisant la ressource secours en cas de défaillance de la ressource primaire. La référence [73] détaille un certain nombre de stratégies de reconfiguration pouvant être mises en oeuvre suite à la détection d'une erreur.

Les approches de tolérance aux fautes sont basées sur le principe de la redondance : la redondance logicielle ou matérielle, en temps ou en information, doit permettre de découvrir une manifestation de la faute. La validation de l'approche de tolérance peut être faite par une méthode appelée technique d'injection de fautes. Celle-ci consiste à observer le comportement du système en présence de fautes, introduites volontairement dans le modèle. Cette technique permet de valider les stratégies de tolérance aux fautes mises en place dans le modèle (validation des redondances, reconfigurations logicielles ou matérielles,...).

De nombreuses approches d'injection de fautes s'appliquent pour évaluer la tolérance d'une telle architecture ; approche à base de simulation, par exemple la simulation au niveau RTL (*Register Transfer Level*) qui consiste à injecter des fautes au niveau transferts de registres [138, 139] et simulation au niveau portes [156] ; d'autres injections basées sur le matériel, citons l'injection au niveau des broches [83–85], par corruption de la mémoire [86], par perturbations de l'alimentation [88], et par laser [90]. Dans la suite, nous ciblons la méthode d'injection de fautes basées sur des simulations de descriptions de haut niveau RTL, ce processus d'injection de fautes étant toutefois entièrement compatible avec la réalisation des expériences permettent de valider nos architectures. Il est cependant nécessaire d'avoir un ou plusieurs modèles de fautes, les fautes qui nous intéressent sont les fautes SEUs (*Single Event Upset*), MBU (*Multiple Event Upsets*), SET (*Single Event Transient*). Ces modèles des fautes sont couramment utilisés avec les modèles RTL [98, 99], qui représentent les phé-

nomènes physiques le plus fidèlement possible. De plus les modèles de fautes doivent être applicables dans le cadre de l'approche choisie pour l'injection, et qui seront davantage détaillés dans la dernière partie de ce chapitre.

En résumé, nous allons présenter une méthodologie globale pour une étude de sûreté, ensuite nous ciblons en particulier la tolérance aux fautes des systèmes numériques autour de laquelle s'orientent dans ce manuscrit nos travaux, et la validation de la tolérance par la simulation RTL. Une grand partie de ce chapitre sera consacrée à ces thèmes qui sont parmi les moyens les plus utilisés dans la littérature pour concevoir des systèmes sûrs de fonctionnement [73].

2.3 Autres propriétés de la SdF

Les attributs de la SdF sont les principales composantes qui définissent la notion de SdF. Elles sont des grandeurs chiffrables qui dépendent les unes des autres et doivent être prises en compte pour toute étude de sûreté. Dans le même temps, certaines des grandeurs peuvent être contradictoires. Par exemple, pour l'amélioration de la disponibilité d'un composant, on néglige parfois la maintenance préventive et la sécurité diminue en conséquence. Les études de la SdF permettent ainsi de chiffrer le compromis optimal entre les diverses grandeurs. Pour caractériser la SdF, en plus des probabilités définies avant, on utilise aussi des grandeurs temporelles moyennes (figure 2.3) :

- **MTTF** ou **MTFF** *(Mean Time To First Failure)* : temps moyen de bon fonctionnement avant la première défaillance,
- **MTTR** *(Mean Time To Repair)* : temps moyen de réparation,
- **MTBF** *(Mean Time Between Failure)* : temps moyen entre deux défaillances consécutives d'un système réparable,
- **MUT** *(Mean Up Time)* : temps moyen de bon fonctionnement après réparation,
- **MDT** *(Mean Down Time)* : temps moyen de défaillance (temps de détection de la panne, durée d'intervention, temps de réparation et temps de remise en service).

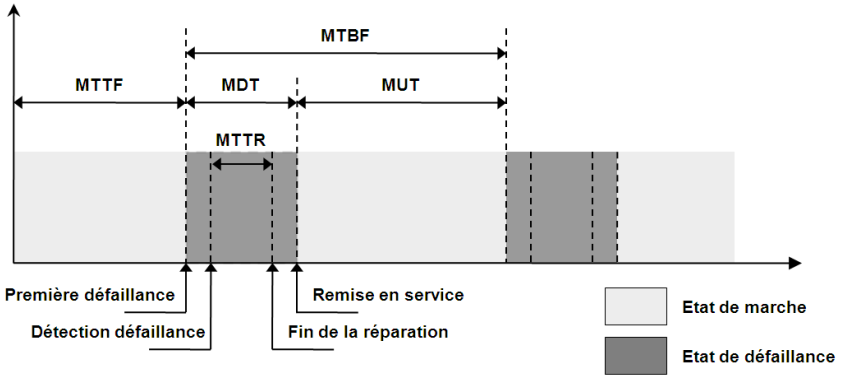


FIG. 2.3 – Diagramme des temps moyens

À signaler que la *défaillance* est l'état d'une entité inapte à accomplir une fonction requise, non comprise l'inaptitude due à la maintenance préventive ou à d'autres actions programmées, ou due à

un manque de moyens extérieurs, citons quelques définitons :

- **Taux de défaillance** : probabilité pour qu'une entité perde sa capacité à accomplir une fonction pendant l'intervalle $[t, t + dt]$, sachant qu'elle a fonctionné entre $[0, t]$,
- **Taux de réparation** : probabilité pour qu'une entité soit réparée ou remplacée pendant l'intervalle $[t, t + dt]$, sachant qu'elle a été en panne entre $[0, t]$,
- **Taux de défaillance à la sollicitation** : probabilité pour qu'une entité refuse de changer d'état lorsque cela lui est demandé sous forme d'une sollicitation.

Nous allons présenter par la suite les origines des défaillances d'un service, et les méthodes d'analyse de défaillance pour déterminer leurs causes et estimer leurs conséquences sur le service rendu par le système.

2.4 Défaillance du service

Le comportement d'un système est perçu par son (ou ses) utilisateur(s) comme une alternance entre deux états du service délivré par rapport au service spécifié :

- service correct, le service délivré étant conforme à la spécification,
- service incorrect, le service délivré n'étant pas conforme à la spécification.

La notion de service rendu est donc déterminante. Si le service est tel que prédéfini en termes de fonctionnalité et de performance on dit qu'il est correct. Les événements qui conduisent aux transitions entre les deux états du service sont, respectivement, la défaillance, transition de service correct à service incorrect, et la restauration, transition de service incorrect à service correct.

Une *défaillance* (failure) survient lorsque «le service délivré dévie du service correct, soit parce qu'il n'est plus conforme à la spécification, soit parce que la spécification ne décrit pas de manière adéquate la fonction du système» [74], et peuvent dépendre de circonstances liées à la conception, la fabrication et/ou à l'emploi, internes ou externes à l'entité. La *spécification* étant une description agréée de la fonction ou du service attendu du système.

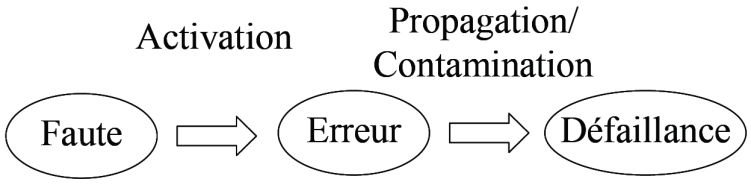


FIG. 2.4 – Schéma faute-erreur-défaillance

Une *erreur*, état incorrect du système, peut entraîner une *défaillance* si elle se propage et devient observable de l'extérieur. L'origine d'une erreur est une *faute* dans le système, telle une valeur logique incorrecte d'un noeud de celui-ci. Cet enchaînement est très souvent illustré par un schéma comme celui de la figure 2.4. Une faute engendre une erreur lorsqu'elle est activée. Or une faute peut

également rester latente si elle se trouve dans une partie non utilisée du circuit. Suivant leur origine, nature, ou phase de création, les fautes sont classées en trois types principaux : *fautes de conception*, *fautes physiques* et *fautes d'interaction*.Dans notre étude nous nous intéressons seulement aux fautes physiques.

Dans la partie qui va suivre nous allons présenter une méthodologie globale d'une étude de la sûreté comprenant les méthodes principales d'analyse de sûreté de fonctionnement.

2.5 Méthodologie d'une étude de la SdF

Une étude complète de la SdF comporte en réalité la phase d'analyse qualitative, suivie par une analyse quantitative [94]. La première phase, ou la phase d'identification du système, est constituée d'une analyse du besoin client, une analyse fonctionnelle du système et une analyse des défaillances pour chaque composant du système. La deuxième phase, ou la phase de quantification, est composée d'une phase de recherche de données nécessaires pour l'étude et d'une phase de modélisation du système et de son comportement, finissant avec une évaluation effective des critères de sûreté.

L'analyse de la SdF peut être qualitative et/ou quantitative. Les principales méthodes sont réalisées suivant deux types de raisonnement logique (figure 2.5) :

- *Approche inductive* : raisonnement du particulier au général. Dans ce cas, on recherche les effets d'une défaillance sur le système ou son environnement,
- *approche déductive* : raisonnement du général au particulier. Dans ce cas on suppose que le système est défaillant et on recherche les causes possibles de la défaillance.

La chronologie d'une étude de sûreté est présentée dans la figure 2.6.

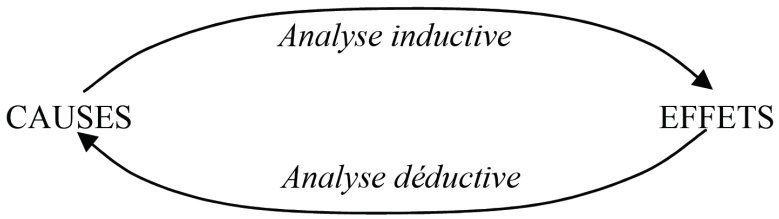


FIG. 2.5 – Différents raisonnements logiques

L'*analyse fonctionnelle* est une démarche qui consiste à rechercher, ordonner, caractériser, hiérarchiser et/ou valoriser les fonctions du produit (matériel, logiciel, processus, service) attendues par l'utilisateur, elle est utilisée au début d'un projet pour créer (conception) ou améliorer (reconception) un produit. Elle est un élément indispensable à sa bonne réalisation afin d'effectuer un dimensionnement correct des caractéristiques du produit, donc c'est une analyse qui précède donc une étude de sûreté de fonctionnement.

L'AMDEC (*analyse des modes de défaillance, de leurs effets et de leur criticité*) est une technique spécifique de la sûreté de fonctionnement, mais aussi et surtout une méthode d'analyse de systèmes statiques (systèmes au sens large composés d'éléments fonctionnels ou physiques, matériels, logiciels,

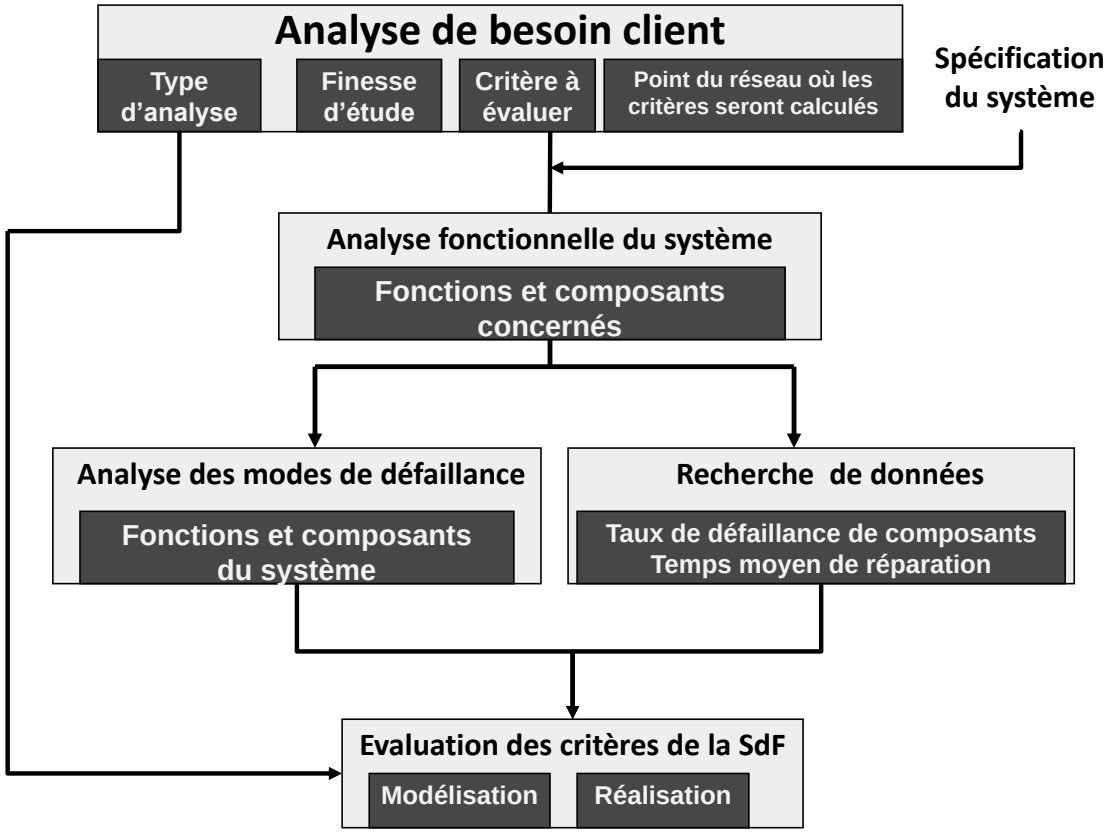


FIG. 2.6 – Chronologie d'une étude de la SdF

humains ...) s'appuyant sur un raisonnement inductif (causes conséquences), pour l'étude organisée des causes, des effets des défaillances et de leur criticité. Le plus grand inconvénient de la méthode est son incapacité à prendre en compte des défaillances multiples (systèmes redondants) [95]. L'AMDEC considère uniquement les défaillances simples.

Pour évaluer les *critères de la SdF d'un système*, une partie préliminaire de *modélisation* du système et de son comportement est nécessaire. Une fois le modèle réalisé, nous pouvons passer à sa *réalisation*. Le choix des méthodes de modélisation et de réalisation dépend fortement de la complexité du problème à étudier, de la précision des résultats qu'on souhaite obtenir mais aussi du type des indices à calculer.

2.5.1 Solutions de la SdF aux systèmes embarqués

Un *système embarqué* est un système électronique, piloté par un logiciel, qui est complètement intégré au système qu'il contrôle. On peut aussi définir un système embarqué comme un système électronique soumis à diverses contraintes, il combine généralement diverses technologies qui relèvent des domaines de la mécanique, de l'hydraulique, de la thermique, de l'électronique et des technologies de l'information.

Lors de la conception d'un système matériel, le concepteur se doit de respecter les exigences exprimées par les différentes parties prenantes [96]. Ces exigences sont soit fonctionnelles soit non

fonctionnelles (les délais, les coûts, la sûreté de fonctionnement, les performances). De nombreuses méthodes et outils ont été développés pour faire face à la complexité croissante des systèmes embarqués, malgré cela, une méthodologie avec une démarche systémique est nécessaire pour passer du simple cahier des charges, qui exprime les besoins et exigences, à la réalisation du produit final.

Les études dues à Lenoir [97] montrent que les principaux défauts de conception sont :

- des besoins mal spécifiés ou des exigences mal formulées,
- une évolution des besoins/exigences dans le temps,
- une modification spontanée, parfois faite avec de bonnes intentions,
- la non accumulation de savoir-faire et manque de retour d'expérience,
- le pari technologique,
- une définition erronée d'interfaces,
- la pression de la concurrence,
- une extension d'exigences fonctionnelles.

Traditionnellement, dans la conception des systèmes, chaque fonction peut être étudiée et développée indépendamment des autres et l'implication de la sûreté de fonctionnement se résumait à la réutilisation de modèles génériques issus du retour d'expérience. Cette approche traditionnelle ne permet pas de prendre en compte les risques liés à l'intégration de plusieurs technologies. Il est donc important de formuler les exigences de sûreté de fonctionnement non seulement localement («*in the small*») mais globalement (niveau système, «*in the large*»). Cela revient à formuler ces exigences au niveau du système complet et, ensuite, à les décliner à des niveaux plus bas (jusqu'aux simples composants). Le «*cycle en V*» (figure 2.7) est généralement utilisé dans la description du cycle de

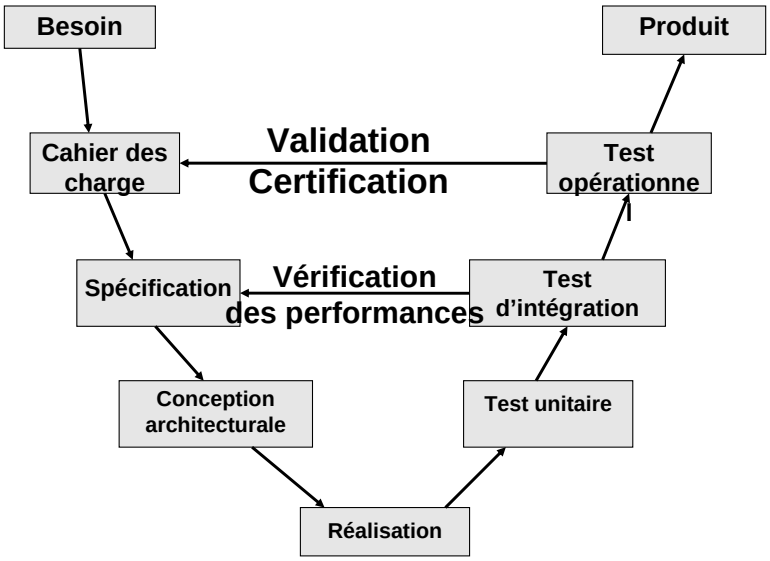


FIG. 2.7 – Cycle de développement en V d'un système embarqué

développement d'un système embarqué. Le cycle est composé de deux branches. La branche descendante, qui correspond à une démarche de raffinements successifs, décrit les phases de conception

allant du général, qui démarre avec l'expression des besoins, au particulier. La branche ascendante détaille les phases d'intégration et de validation correspondant à chaque phase de conception.

La conception d'un système suivant le cycle en V permet de retarder le choix de la technologie de réalisation. Les efforts sont concentrés sur la spécification et la conception. Les phases de spécification et de conception conduisent à définir des niveaux de description de plus en plus détaillés. Les phases d'intégration, de test et de vérification permettent d'évaluer la conformité de la réalisation pour chaque niveau de la conception. A signaler que les mêmes phases de la figure 2.7 ont été retenues dans notre étude de conception afin de valider et réaliser nos architectures.

2.6 Tolérance aux fautes, principes et mécanismes

Cette section résume l'état de l'art des méthodes permettant de détecter ou de tolérer des fautes logiques dans un circuit numérique synchrone. L'objectif est essentiellement d'identifier les principales méthodes existantes et leurs caractéristiques. La plupart de ces méthodes ne peuvent être utilisées directement dans le cadre de ce travail, car elles sont trop coûteuses en termes de consommation en surface, et leur utilisation pour durcir des circuits synchrones contre les fautes est inefficace. En revanche, elles peuvent être utilisées à titre de comparaison pour évaluer les techniques spécifiques aux circuits synchrones proposées dans ce manuscrit, en termes d'efficacité dans la protection contre les fautes et en terme de coût : consommation en surface, performance en vitesse.

2.7 Principe de la tolérance aux fautes

La *tolérance aux fautes* est définie comme l'aptitude d'un système à délivrer son service conforme à la spécification, malgré la présence ou l'occurrence de fautes, qu'il s'agisse de dégradations physiques du matériel, de défauts logiciels, d'attaques malveillantes, d'erreurs d'interaction homme-machine ou autre. Le but est alors de limiter les effets d'une faute, autrement dit d'accroitre la probabilité qu'une erreur soit *acceptée* ou *tolérée* par le système.

Deux grand thèmes pour la mise en oeuvre de la tolérance aux fautes :

♢ **Traitement des erreurs** : ce sont des opérations dans le but d'éliminer les erreurs, si possible avant qu'une défaillance ne survienne. Parmi ces opérations on distingue :

- **Détection d'erreur** : détecter l'existence d'un état incorrect.
- **Diagnostic d'erreur** : permet d'estimer les dommages créés par l'erreur qui a été détectée et par les erreurs éventuellement propagées avant la détection.
- **Recouvrement d'erreur** : remplacer l'état incorrect par un état correct (conforme aux spécifications) :

  ▷ **Reprise** : le système est ramené dans un état survenu avant l'occurence d'erreur ; ceci passe par l'établissement de *points de reprise* : des instants durant l'exécution d'un processus don l'état courant peut ultérieurement nécessiter d'être restauré.

- ▷ **Poursuite** : un nouvel état est trouvé à partir duquel le système peut fonctionner (habituellement en mode dégradé).
- ▷ **Compensation d'erreur** : l'état erroné comporte suffisamment de redondance pour permettre la transformation de l'état erroné en un état exempt d'erreur.

♢ **Traitement des fautes** : ce sont des opérations destinées à éviter qu'une ou des fautes ne soient activées à nouveau :

- **Diagnostic de faute** : déterminer les origines possibles des erreurs, en terme de localisation et de nature.
- **Passivation des fautes** : c'est l'objectif principal du traitement des fautes qui consiste à empêcher une nouvelle activation des fautes. Les composants considérés comme fautifs sont retirés du processus d'exécution ultérieure.
- **Reconfiguration** : cette étape est nécessaire lorsque le système devient incapable de délivrer le même service qu'auparavant. Elle consiste à modifier la structure du système pour lui permettre de délivrer un service dégradé *acceptable* : abandon ou réallocation de certaines tâches aux composants restants.
  - ▷ (par exemple : réaliser la même opération avec des algorithmes différents pour tenter d'éviter les fautes de conception)

Un point commun à toutes les techniques de tolérance aux fautes est l'utilisation de la redondance. L'éventail des cet techniques de redondance utilisées peut être dressé ci-dessous :

- **redondance matérielle** redondance dynamique, redondance hybride, circuits autotestables
- **redondance temporelle** répétition d'une opération
- **redondance d'information** codes détecteurs/correcteurs d'erreurs, duplication des données
- **redondance logicielle** tests d'acceptation, méthodes pour les recouvrements d'erreurs, blocs de recouvrement, programmmation en N-versions

La redondance peut être matérielle (certains modules matériels sont dupliqués, triplés (tripliqués) (TMR : *Triple Modular Redundancy*)), logicielle où l'algorithme est effectué de plusieurs façons différentes, temporelle (certaines parties d'un programme sont exécutées plusieurs fois), d'information (le circuit ou le programme possède une redondance d'information) ou un mélange de ces quatre solutions.

On parlera généralement d'un système résistant/tolérant à n fautes, ce qui signifie que le système a la capacité de détecter (respectivement tolérer) jusqu'à n fautes simultanées, en référence à un modèle de faute donné. Par la suite nous allons détailler les différents modèles de redondance (matérielle, temporelle, information) utilisés pour la détection /correction d'erreurs

2.7.1 Détection d'erreurs

La détection de fautes consiste à déceler et à signaler l'occurrence d'une faute (ou d'une erreur), susceptible d'altérer le service fourni par le circuit. Certaines techniques de durcissement permettent

de rendre un *circuit résistant aux fautes*. Les fautes (ou erreurs) sont détectées par le circuit et celui-ci en informe son environnement.

- **Redondance matérielle**

  ▷ Duplication et comparaison (*DWC : Duplication With Comparison*) [100] : C'est la technique de détection la plus simple comme le montre la figure 2.8. Le coût en surface est de $+100\%$, plus la logique de comparaison. De même pour la consommation $(+100\%)$. Le coût en vitesse est faible car le signal de comparaison est obtenu uniquement avec de la logique et en parallèle du signal de sortie. Cette architecture peut s'appliquer à tous les blocs mais c'est l'une des moins optimisées.

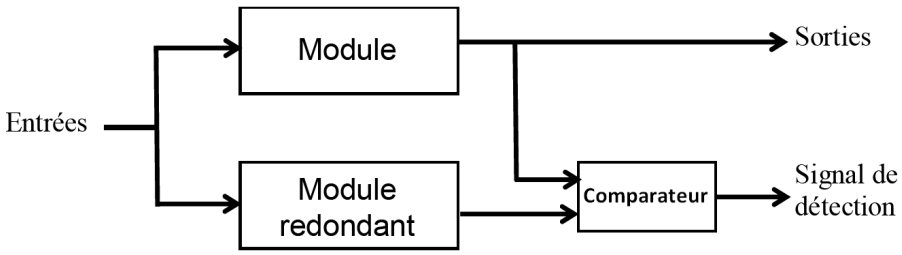


FIG. 2.8 – Architecture Duplication et Comparaison

  Cette technique est la plus importante présentée dans ce document ; elle est considéré comme une référence pour valider tous nos travaux que nous verrons dans les chapitres suivants. Il suffit de comparer pour chaque architecture tolérante aux fautes proposée la surface consommée par rapport à cette méthode classique.

  ▷ Duplication Avec Redondance Complémentaire (*DWCR : Duplication With Complementary Redundancy*) [101] : Cette technique est similaire à la méthode DWC mais les signaux d'entrée et de sortie, les signaux de contrôle (distingués des entrées de données) et les données internes sont de polarités opposées dans les deux modules. Le surcoût en surface et en puissance consommée est de l'ordre de $+100\%$ et il faut ajouter la logique de comparaison. Le surcoût en vitesse est faible. Néanmoins cette méthode augmente la complexité de la conception par rapport à une duplication simple. La réalisation en double rail (DRC : Double Rail Code), dans laquelle les deux sorties sont inversées si il n'y a pas de fautes, peut être vue comme un cas plus simple de redondance complémentaire. Le détecteur double rail est le plus souvent utilisé comme bloc de comparaison (contrôleur). Un exemple d'utilisation se trouve dans [103].

- **Redondance temporelle**

  ▷ Redondance temporelle simple : L'opération est effectuée deux fois de suite et les résultats sont comparés. Cette méthode est simple, et bien adaptée aux fautes temporaires comme les

SEU. Cette architecture entraîne un surcoût en vitesse très important car le temps de calcul est plus que doublé, le temps de cycle augmentant légèrement à cause de la comparaison et deux cycles étant nécessaires au lieu d'un. Le surcoût en surface est limité à la mémorisation du premier résultat et à la logique de comparaison, la puissance consommée augmente peu (mais l'énergie augmente notablement).

▷ Redondance Temporelle au niveau des bascules [124] : Cette technique vise la détection de SET . Le circuit de détection se compose de deux bascules et d'un comparateur par sortie du circuit combinatoire (2.9). Ce circuit permet de détecter une faute transitoire affectant le circuit combinatoire. S'il n'y a pas de faute, le résultat est directement disponible sur la bascule de sortie. Le retard δ est fonction du temps de "set-up" des bascules (Dsetup) et du temps maximum toléré pour le SET (Dtr) : δ = Dsetup + Dtr. En général Dtr vaut quelques picosecondes.

Le surcoût en surface est dû aux bascules supplémentaires, à la logique de comparaison

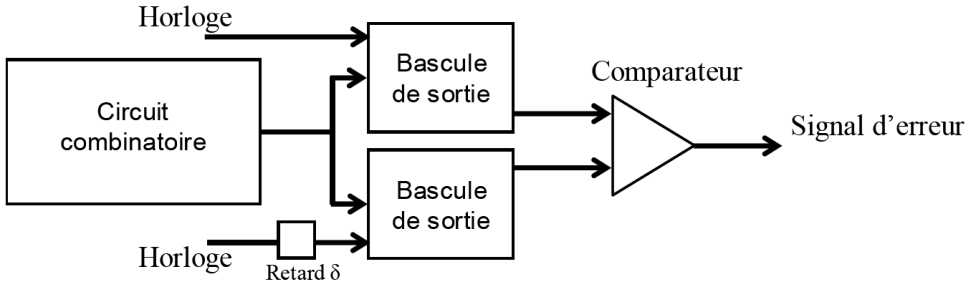


FIG. 2.9 – Architecture de contrôle concurrente basé sur la redondance temporelle

et aux éléments rajoutés pour retarder le signal d'horloge. Les essais ont été effectués sur des additionneurs et des multiplieurs ; pour les multiplieurs le surcoût en surface est compris entre $+1,4\%$ et $+10\%$, pour les additionneurs le surcoût en surface est compris entre $+22,6\%$ et $+70,2\%$.

La méthode de contrôle concurrente basée sur la redondance temporelle s'applique bien aux blocs se trouvant dans le chemin critique. Le surcoût en surface peut être assez faible (surtout pour les multiplieurs), les surcoûts en vitesse et en consommation sont faibles.

- Recalcul avec Duplication et Comparaison (*REDWC : REcomputing Duplication With Comparison*) [101] : REDWC est une méthode de détection de fautes par association de redondance temporelle et de duplication. Elle s'applique aux additionneurs et aux multiplieurs, le principe est de diviser les opérandes sur n bits en opérandes sur $n/2$ bits. Prenons l'exemple de l'addition :

  – Les demi-mots de poids faibles sont additionnés simultanément par deux additionneurs $n/2$ bits, les résultats sont comparés et l'un deux est mémorisé.

– La retenue est réinjectée dans les additionneurs et on additionne les demi-mots de poids forts. Les résultats sont comparés et s'il n'y pas de faute, le résultat final est obtenu par concaténation du résultat obtenu pour les poids faibles et les poids forts.

Cette technique présente un bon compromis entre surcoût en surface (+75%), en vitesse (+31% pour le temps de cycle d'un multiplieur) et en puissance consommée par rapport aux autres techniques temporelles.

- **Redondance d'information**

La redondance d'information consiste à utiliser un code détecteur ou correcteur d'erreur. Quelques exemples de codes détecteurs sont cités ci-dessous.

▷ Codage par parité : La détection par génération d'un bit de parité est basée sur le calcul du nombre des bits à 1 d'un mot de données et permet la détection de toutes les erreurs de multiplicité impaire. La génération et le contrôle du bit de parité se font à l'aide de portes XOR.
Ce mode de détection est simple et entraîne un coût temporel limité. Dans le cas des mémoires, le bit de parité est généré quand les données sont écrites dans la mémoire et il est contrôlé au moment de la lecture. Le code de parité est un code systématique.

Le codage par parité est très utilisé, voici quelques exemples d'application :
– Codage par parité à deux dimensions (lignes et colonnes) pour les mémoires et bancs de registres [105]. L'inconvénient est que deux erreurs sur la même ligne entraînent la corruption de toute la mémoire (c'est le cas de plusieurs codes aussi).
– Pour la détection d'erreurs multiples et dans le cas de bancs de registres, on peut implanter la technique de la parité croisée (Cross-Parity), développée en détails dans [106]. Elle correspond au codage par parité des lignes, colonnes et diagonales qui constituent une mémoire et à la vérification par analyse des trois parités obtenues.
– Prédiction du code de parité notamment pour les applications à faible ou moyenne consommation [103].

▷ Codage "Double Rail" : Le codage "Double Rail" consiste à dupliquer et complémenter chaque mot de données et permet de détecter les erreurs multiples. Le surcoût en surface est très important (+100%) car il y a autant de bits de contrôle que de bits de données.

▷ Codes non ordonnés : Les codes non ordonnés détectent toutes les erreurs unidirectionnelles. Ce sont les erreurs multiples, telles que tous les bits erronés du mot comportent le même type d'erreur (soit uniquement «0» vers «1», soit uniquement «1» vers «0»). Si une telle erreur af-

fecte un mot du code, le mot résultant n'appartient plus au code non ordonné et ainsi l'erreur est détectable. Les codes non ordonnés les plus intéressants sont les codes m parmi n, et les codes de Berger.

– Le code m/n est un code non-séparable composé de mots de n bits comportant m bits à 1. Exemple le code $2/4$ est composé des mots $\{1100, 1010, 1001, 0101, 0011, 0110\}$. Ce type de codage peut-être utilisé dans les circuits de contrôle mais il entraîne une redondance très importante et des circuits de codage/décodage très compliqués.

– Le code de Berger est un code séparable non ordonné où la partie de contrôle représente le nombre de «0» de la partie donnée. Le nombre de bits de contrôle est égal à $\lceil log_2(n+1) \rceil$, pour n bits de la partie donnée.

Ces codes sont utilisés pour détecter les erreurs unidirectionnelles affectant surtout les PLA, les mémoires ROM ou les circuits logiques sans inverseur avec de la logique partagée.

▷ Codage résidu : Le code résidu (arithmétique) correspond à la concaténation de n bits de données et de m bits de contrôle représentant le résidu (modulo r) des données, ce code est noté (n, m). La base r du résidu doit être différente de 2^m, en effet si $r = 2^m$ les fautes simples affectant les bits de poids $\geq 2^m$ ne sont pas détectables. Pour la détection de fautes simples on prend $r = 3$, ce qui fait apparaître deux bits pour $\mathrm{mod}\ 3$ (1 de plus que la parité) de redondance pour le codage de l'information.

▷ Codes arithmétiques : Les codes arithmétiques sont intéressants pour les circuits arithmétiques, parce qu'ils sont préservés par les opérations arithmétiques comme l'addition, la soustraction et la multiplication. Ces codes sont divisés en deux catégories : les codes séparables et les codes non séparables. Dans les codes séparables, sur une base A, les mots codes sont obtenus en associant à la partie donnée X, une partie de contrôle $X' = |X|_A$ (le code résidu), ou $X' = A - |X|_A$ (le code résidu inverse). Dans les codes arithmétiques non séparables, sur une base A, on parle de mots codes égaux au produit des mots d'origine (non codés) par la base A (X.A).

- **Architectures pour séquenceurs (machines à états finis)**

Des techniques de protection particulières peuvent être définies pour certains blocs, notamment les machines à états. Un exemple est la vérification du flot de contrôle par analyse de la signature (CFC : Control Flow Checking) qui donne lieu à deux réalisations possibles, avec ou sans ajustement, comme développées dans [107]. La signature est calculée à partir du code de l'état futur et de la signature courante. Le surcoût en surface pour cette architecture dépend fortement

de la structure de la machine à états considérée. Il peut être très faible, mais peut aussi atteindre des valeurs supérieures à $+100\%$. Le surcoût en temps (chemin critique) est également très variable [108].

2.7.2 Recouvrement

Les différentes techniques ci-dessus permettent la détection de fautes dans un circuit mais pour les applications critiques la détection n'est pas suffisante puisque le résultat en sortie reste faux. Pour effectuer un recouvrement, on peut simplement répéter la même fonction lorsqu'une erreur est détectée, sachant que la faute est supposée transitoire. Cette technique peut s'appliquer à tous les blocs d'un circuit. Selon le type de bloc, il est nécessaire de définir soigneusement les points de reprise possibles.

Dans le cas particulier des microprocesseurs, le recouvrement peut s'effectuer à plusieurs niveaux, selon que l'on considère l'exécution d'une macro-instruction (Branch ou Add par exemples), ou l'exécution des micro opérations qui sont réalisées par le processeur (lecture de la mémoire, chargement de registres, etc.). Ainsi on exécute, lorsque cela est possible, un "**micro-rollback**" après détection d'une faute lors de l'exécution d'une micro-opération, au niveau d'un étage d'un processeur (pipeline ou non) [109, 112]. Mais il est parfois nécessaire d'effectuer un "**stage rollback**" ou un "**pipeline rollback**" (qui coûtent plusieurs cycles d'horloge), lorsque les données nécessaires à l'exécution de la micro-opération ne sont plus disponibles. Le processeur doit alors ré-exécuter un certain nombre d'opérations effectuées auparavant [109].

2.7.3 Correction d'erreurs

Il est possible d'utiliser d'autres techniques de redondance qui permettent la correction des fautes en plus de les détecter. Parmi ceux-ci, nous pouvons citer la technique TMR, les codes de Hamming utilisés pour le test des mémoires [181, 182], et d'autres codes qui peuvent servir à la correction aussi tels que les codes de parité 2-D et les codes arithmétiques [122, 123].

- **Redondance matérielle**

  La méthode **TMR (Triple Modular Redundancy)** correspond à la triplication d'un module avec un circuit de vote majoritaire 2 parmi 3 en sortie comme illustré figure 2.10. Lorsque plus d'un des modules n'est plus fonctionnel, la tolérance n'est plus garantie et la sortie peut être erronée. Le coût en surface et en consommation est élevé à cause du triplement du nombre de modules et du rajout de la logique de vote mais le surcoût en vitesse est faible car le voteur ne comprend que peu de logique combinatoire.
  Cette méthode peut-être appliquée à tout bloc, globalement ou au niveau le plus bas, c'est-à-dire pour chaque bascule, ou bien encore, comme dans le cas des machines à états, pour une sous partie comme la logique de séquencement [107]. On peut étendre la méthode TMR à la

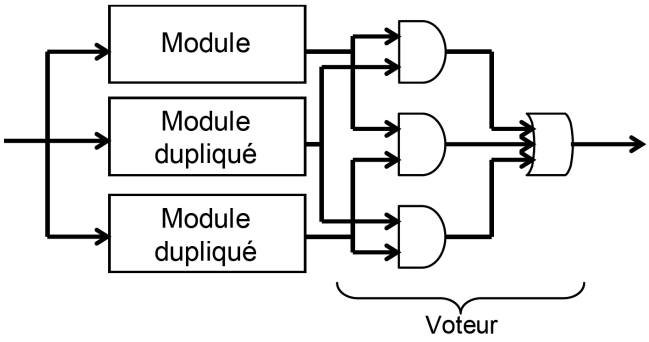


FIG. 2.10 – Architecture TMR

méthode NMR qui utilise $N-1$ modules redondants. L'avantage est que la tolérance est toujours effective si un ou plusieurs modules (jusqu'à $N/2-1$) tombent en panne ; l'inconvénient est un surcoût en surface et consommation encore plus important.

- **Redondance temporelle**

  La méthode est similaire à la technique de redondance temporelle REDWC (duplication et comparaison) présentée précédemment, mais appliquée à la tolérance donc avec triplication et vote [101]. Le surcoût en surface est de l'ordre de $+100\%$ et le surcoût en temps (temps de cycle) est compris entre $+76\%$ (additionneur) et $+131\%$ (multiplieur).

- **Redondance d'information**

  Le codage de Hamming modifié (code Hsiao) est l'un des codages les plus utilisés pour la détection/correction d'erreurs dans les mémoires. Il rajoute des bits de contrôle à certaines positions dans un bloc de bits d'information [102]. Il est tel que $2^m \geq n+m$, où n est le nombre total de bits (bits de données et bits de correction) et m est le nombre de bits de correction. Le résultat est un code séparable qui permet soit la détection d'erreur sur deux bits (code DED : *Double Error Detection*) ou la correction d'erreur sur un bit (code SEC : *Single Error Correction*). On peut obtenir un code SEC/DED en rajoutant un bit de parité, ce qui revient à rajouter une ligne de 1 dans la matrice H.

  Les résultats obtenus dans le cadre d'un durcissement de micro-processeur 8051 [113] montrent que le codage de Hamming est le plus efficace pour la protection de la mémoire interne (faible nombre de blocs logiques de codage/décodage) et le moins efficace pour l'UAL Unité Arithmétique et Logique (grand nombre de blocs logiques de codage/décodage). Ses avantages sont un faible coût en surface (moins de $+50\%$) et la correction d'erreur (pas seulement la détection).

2.8 Circuits auto-contrôlables

La méthode traditionnelle de tolérance aux fautes présentée ci-dessus dans le cas des circuits logiques, et qui consiste à la triplication du circuit par la technique TMR a un surcoût matériel très élevé, dépassant 200%, ce qui est inacceptable pour des applications à faible valeur ajoutée comme les produits grand public. De même la technique de détection concurrentielle consiste en la vérification des résultats fournis par un circuit pendant son fonctionnement normal, et peut être réalisée par une procédure qui consiste à la duplication du système numérique suivie de la comparaison des sorties provenant des deux copies. Cette technique a pour principal désavantage un surcoût matériel très important, plus de 100%. Dans le cas de fautes transitoires, il existe une méthode de tolérance aux fautes moins coûteuse. Il s'agit de la combinaison de la détection d'erreurs concurrentielle avec une procédure de reprise, afin de réaliser la correction d'erreurs détectées.

Toutes ces méthodes restent inefficaces dans différents domaines d'applications d'où l'intérêt de s'orienter vers d'autres types des circuits tels que les circuits auto-contrôlables.

Les circuits « auto-contrôlables » "self-checking" dans la littérature anglaise [114–119] sont une alternative intéressante pour la détection concurrentielle du fait de leur coût beaucoup plus faible. Ils détectent les fautes permanentes, mais aussi les fautes transitoires. Un nombre significatif d'implantations de circuits auto-contrôlables a été proposé dans le passé. Une grande partie de ces circuits offrent une détection d'erreurs à faible coût (par exemple des multiplicateurs auto-contrôlables utilisant des codes arithmétiques [120–123]. Notre but est de pouvoir utiliser ces circuits pour le contrôle concurrentiel de fautes transitoires SEU. Les fautes transitoires seront notre première préoccupation car elles représentent les causes les plus importantes de dégradation de la fiabilité dans les technologies submicroniques avancées [124].

Dans les circuits auto-contrôlables, le circuit complexe est partitionné en plusieurs blocs fonctionnels, chacun de ces blocs étant implémenté d'après le principe présenté dans la figure 2.11. Cette structure consiste en un bloc fonctionnel, fournissant des sorties qui appartiennent à un code détecteur d'erreurs. Ensuite, un contrôleur de code vérifie si les sorties du bloc fonctionnel appartiennent au code et réalise ainsi la détection concurrentielle d'erreurs.

Dans la suite, nous présenterons les classes de circuits auto-contrôlables définissant ainsi leurs propriétés diverses. Ces propriétés ont été introduites par Carter et Sschneider [125], et sont formalisées par Anderson [114].

2.8.1 Circuits sûrs en présence de fautes (*Fault-Secure*)

Pour un ensemble de fautes $F = \{f_1, f_2, f_3\}$, un circuit H est **sûr en présence de fautes**, si pour chaque faute $f \in F$ du code d'entrée Cin, les sorties erronées n'appartiennent pas au code de sortie $Cout$ (figure 2.12).

Cette propriété garantit que, pour l'ensemble de fautes F, le circuit H ne génère pas de sorties

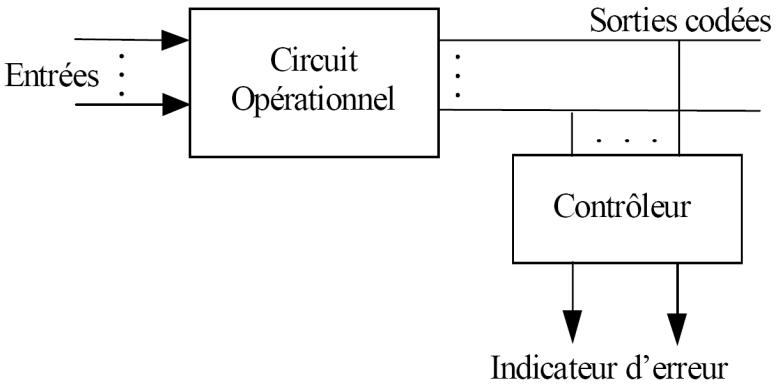


FIG. 2.11 – La structure générale d'un circuit auto-contrôlable

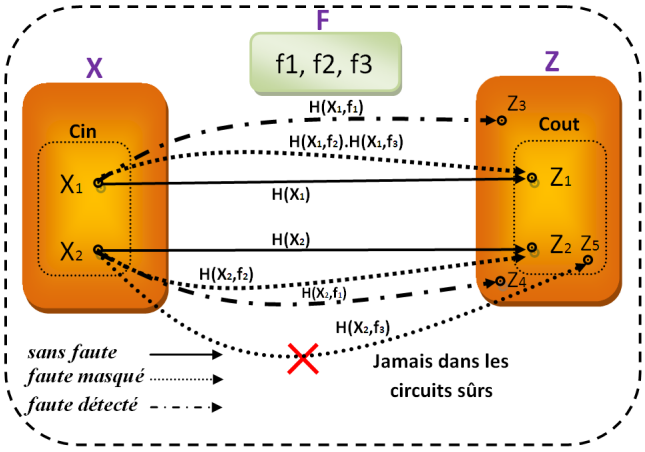


FIG. 2.12 – Circuit sûr en présence de fautes

erronées qui appartiennent au code, et qui ne sont donc pas détectées par le contrôleur (cette propriété est suffisante pour les fautes temporaires). Une autre propriété utile est la propriété d'auto-testabilité.

2.8.2 Circuits auto-testables (*Self-Testing*)

Pour chacune des fautes modélisées affectant le circuit H, il y a au moins un vecteur d'entrée pendant le fonctionnement normal du circuit qui produit des sorties qui n'appartiennent pas au code de sortie.

Cette deuxième propriété évite par exemple l'existence des fautes permanentes. De telles fautes restent indétectables et peuvent être combinées à de nouvelles fautes survenant plus tard, pour donner de fautes multiples. Cependant, en présence de fautes multiples, il n'est pas garanti que le circuit reste sûr en présence de fautes. Il peut donc produire des sorties erronées non détectables. La propriété d'**auto-testabilité** nous évite cette situation. La combinaison de ces deux propriétés dans la propriété totalement auto-contrôlable offre le plus haut niveau de protection contre les fautes temporaires et permanentes.

2.8.3 Circuits totalement auto-contrôlables (*Totally Self-Checking*)

Le circuit H est **totalement auto-contrôlable** s'il est sûr en présence de fautes **et** autotestable pour chacune des fautes de F.

La propriété de sûreté en présence de fautes est la plus importante des deux, parce qu'elle garantit la détection d'erreur pour n'importe quelle faute unique. Par contre, elle est la plus difficile à réaliser. La propriété d'auto-testabilité est facile à réaliser, spécialement pour les fautes de collage logique. Une faute de collage logique qui n'est pas détectable correspond à une redondance logique. Ainsi, ces fautes peuvent être éliminées en utilisant un système de minimisation qui élimine les redondances logiques [126].

Pour la plupart des fautes transitoires, la propriété d'auto-testabilité n'a pas de sens, parce qu'il n'y a pas de fautes dans le circuit, mais seulement des erreurs créées par une source externe. Dans ce cas, la seule propriété qui s'applique est la sûreté en présence de fautes. Pour certains autres types de fautes, comme les courts-circuits, par exemple, il n'est pas toujours facile d'éliminer toute faute qui n'est pas détectable. Dans ce cas, et pour toute application qui nécessite un très haut niveau de sécurité, on pourra imposer des contraintes structurelles au circuit de façon à garantir qu'une faute indétectable ne détruise pas la propriété de sûreté en présence de fautes [127]. Ces techniques utilisent la propriété suivante.

2.8.4 Circuits fortement sûrs en présence de fautes (*Strongly Fault-Secure*)

Un circuit H est fortement sûr en présence de fautes F, si pour chaque faute $f \in F$, soit il est totalement auto-contrôlable, soit en présence de f il préserve la propriété de sûreté en présence de fautes.

Les circuits fortement sûrs en présence de fautes constituent la classe la plus large des circuits satisfaisant la propriété de totalement auto-contrôlable. Par contre, dans le cas de fautes spécifiques, telles que les fautes de court-circuit, les contraintes additionnelles garantissant cette propriété impliquent un surcoût matériel qui n'est pas justifié pour la plupart des applications.

Etant donné que dans cette étude nous ne recherchons pas à protéger les circuits dans n'importe quelle situation pour atteindre un niveau de sécurité très élevé, mais cherchons seulement à éviter une dégradation de la fiabilité dans les circuits VLSI, nous nous intéressons dans la suite uniquement à la propriété de sûreté en présence de fautes qui garantit la détection d'erreurs produites après l'occurrence d'une premiere faute.

2.9 Évaluation de la SdF

Avec l'accroissement de la probabilité des fautes dans les technologies les plus récentes, de plus en plus de concepteurs vont devoir analyser finement l'impact potentiel de ces fautes sur le comportement des circuits qu'ils conçoivent. L'injection de fautes est une des principales approches pour l'évaluation de la sûreté. Un grand nombre de méthodes ont été proposées, se basant essentiellement

sur la validation physique des systèmes, et incluant des injections sur les broches des circuits, des corruptions en mémoire [86], l'injection d'ions lourds, la perturbation des alimentations [88], ou encore l'injection de fautes par laser [136, 137]. Aucune de ces approches ne peut être utilisée pour une évaluation de la sûreté avant que le circuit ne soit effectivement fabriqué.

Par contre, il existe d'autres techniques d'injection permettant une analyse plus tôt dans le flot de conception, typiquement au niveau transfert de registres RTL *(*Register Transfer Level) ou au niveau portes, l'injection de fautes dans une description RTL devient une solution efficace pour analyser très tôt dans le processus de conception les conséquences de fautes sur le comportement d'un circuit numérique complexe. Cette approche a été étudiée depuis quelques années [138, 139] et a plus particulièrement été développée pour des systèmes à base de microprocesseurs [140].

L'injecteur de fautes peut correspondre à une implantation matérielle ou logicielle. Il peut supporter différents types de fautes (avec une sémantique matérielle ou une structure logicielle appropriée), injectées à des emplacements multiples et à des instants différents. Dans tous les cas il faudra définir une liste des modèles de fautes qui peuvent être applicables dans le cadre de l'approche choisie pour l'injection. Nous allons présenter dans ce chapitre une vue d'ensemble des approches employées.

2.9.1 Modèles de fautes

Dans cette section nous allons défini les différentes modèles de fautes susceptibles d'être injectées dans notre méthode d'injection au niveau RTL.

Une faute est considérée comme une déviation d'une fonction prévue au niveau matériel ou logiciel. Elle peut surgir à n'importe quelle étape du processus de conception d'un système : spécification, conception, développement, fabrication, et elle peut éventuellement survenir durant la vie opérationnelle d'un système. Les dysfonctionnements pouvant survenir dans un système sont représentés en fonction de leur origine, avec plus ou moins de précision, par de nombreux modèles de fautes. Un modèle de faute est une abstraction de l'état incorrect d'un circuit, l'injection de fautes permet d'évaluer les conséquences de ces fautes. Il est cependant nécessaire d'avoir un ou plusieurs modèles de fautes qui représentent le plus fidèlement possible les phénomènes physiques.

Les effets singuliers, regroupés sous le nom SEE (*Single Event Effect*), effet d'une particule isolée, correspondent aux phénomènes déclenchés par le passage d'une particule unique, telles que les ions lourds ou des protons énergétiques. Ces effets singuliers sont classés en deux sous-catégories :

- ▷ **Les effets irréversibles**, dégradations permanentes destructives appelées erreurs permanentes ou erreurs matérielles. Parmi ces effets, nous citons le SEL (*Single Event Latchup*), impulsion provoquée par une particule isolée.
- ▷ **Les effets réversibles**, défauts transitoires non destructifs, appelés aussi aléas logiques ou erreurs logicielles. Parmi ces effets, nous citons le SET (*Single Event Transient*), impulsion transitoire provoquée par une particule isolée qui affecte la logique combinatoire du circuit et le SEU (*Single Event Upset*), perturbation par une particule isolée qui change directement les points mémoires ou les registres du circuit.

Il existe différents niveaux de modélisation du phénomène d'erreurs logicielles tels que les fautes transitoires provoquées par la radiation, allant d'un modèle spécifique aux descriptions de circuit de très bas niveau "netlist" niveau transistor ou descriptions physiques) jusqu'à des modèles utilisés dans des descriptions de circuit niveau porte ou niveau RTL. Dans notre étude, nous nous sommes focalisés sur la modélisation des fautes au niveau RTL. À ce niveau, les registres de données et les registres d'état sont facilement identifiables. Les modèles de fautes associés aux phénomènes SEU, MBU, et MCU (*Multiple Cell Upsets*) correspondent respectivement à une inversion de bit unique *(single bit-flip)*, multiple *(multiple bit-flip)* dans de tels registres, ou multiple *(multiple bit-flip)* dans plusieurs registres.

Par contre, il est souvent impossible de modéliser avec précision le phénomène SET car, à ce niveau de description, les noeuds d'une logique combinatoire ne sont pas identifiables. Il est donc impossible d'évaluer avec précision les configurations d'erreur dues aux SETs qui peuvent éventuellement se produire. Cependant, supposer que seulement des inversions de bits uniques peuvent se produire est évidemment très optimiste. En conséquence, des modes de défaillance critiques peuvent ne pas être identifiés pendant cette analyse haut niveau et devraient donc être traités beaucoup plus tard dans le processus de conception. Une solution à cette limitation consiste à étendre le modèle de fautes habituel à un modèle que nous appelons MBF *(multiple bit-flips)*. En effet, une impulsion transitoire qui se propage à travers des portes logiques sera finalement capturée par des éléments mémoires. Cela correspond à une inversion de bits multiple au niveau d'un ou plusieurs points mémoires. Dans ce cas on fait alors abstraction des éventuels masquages et on utilise les modèles d'inversion singulière ou multiple.

Nous décrivons dans la suite quelques modèles de fautes singulières et multiples qui peuvent être injectées lors d'une simulation RTL, nous citons :

***Single Event Upset* (SEU)** : Un SEU correspond au basculement d'un point mémoire, sans effet destructeur.

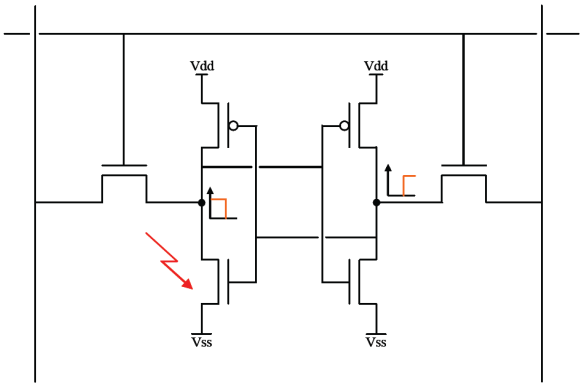


FIG. 2.13 – Single Event Upset (SEU)

La charge Q_{col} est la charge collectée après un impact de particule ou après tout autre type de perturbation. La charge critique Q_{crit} est la charge collectée minimum nécessaire pour créer une

erreur comme définie dans la norme JESD89 [131]. Physiquement, si la charge collectée Q_{col} est plus importante que la charge critique Q_{crit} alors il y a inversion de l'état logique, de '0' vers '1' ou de '1' vers '0'.

La figure 2.13 illustre l'exemple d'une cellule SRAM à 6 transistors. Si une particule fait basculer un nœud la perturbation se propage à travers l'inverseur et fait basculer l'autre nœud. Comme le second nœud commande le premier, les deux nœuds basculent. Le seul moyen de supprimer cette erreur est la réécriture de la cellule mémoire. Les éléments sensibles aux SEU sont tous les éléments mémoires volatiles d'un système : bascules *(Flip-Flops)* et verrous *(latches)* individuels, registres et RAM (SRAM, DRAM ...).

Single Event Transient **(SET)** : Les SETs *(Single Event Transient)* résultent d'une impulsion transitoire créée par le passage d'une particule chargée dans la logique combinatoire d'un circuit numérique. Un cas typique de circuit logique affecté par une impulsion transitoire est présenté dans la figure 2.14. Un SET est injecté dans la logique combinatoire d'un circuit (dans le cas d'un circuit synchrone) et a la possibilité de se propager à travers elle jusqu'à un élément mémoire. En fonction de la topologie de la partie combinatoire, un SET peut très bien se propager vers plusieurs points mémoire.

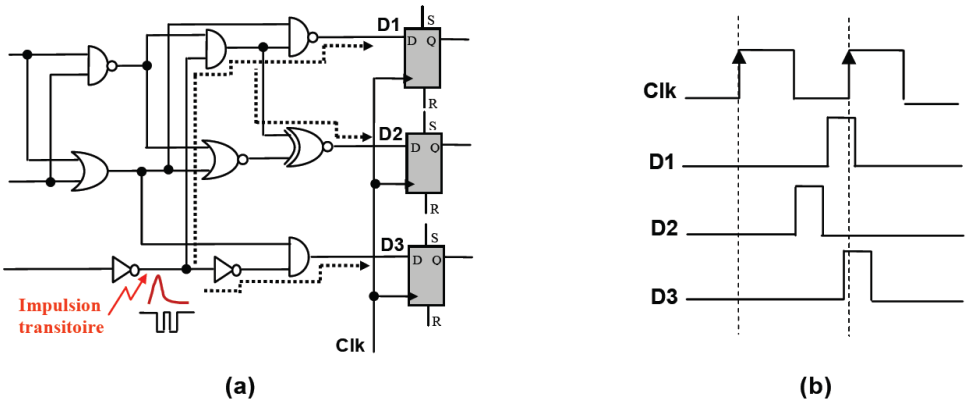


FIG. 2.14 – Propagation d'un SET (a) exemple d'un circuit ; (b) formes d'onde des entrées de bascules

Les trois chemins de propagation montrés dans la figure 2.14(a) sont activés par un vecteur d'entrée adéquat. Ainsi, l'impulsion transitoire peut arriver au niveau des entrées D1, D2 et D3 des bascules en fonction de la longueur du chemin de propagation correspondant. Si elle arrive à l'entrée d'une bascule au moment du front actif de l'horloge du système, elle sera capturée et mémorisée comme une valeur erronée (cas des entrées D1 et D3 dans la figure 2.14(b)).

Dans d'autres cas, l'impulsion transitoire peut être atténuée avant son arrivée aux entrées des bascules. Si sa durée est plus grande que les temps de transition des portes logiques, elle peut se propager aux entrées des bascules sans atténuation [132]. Comme les portes logiques deviennent très rapides dans les nouvelles technologies, avec des temps de propagation très courts, les impulsions transitoires ne seront plus atténuées, même pour des particules d'énergie réduite [133]. Néanmoins, la durée de

l'impulsion finale pourrait être réduite ou augmentée selon les portes traversées.

***Multiple Cell Upset* (MCU) et *Multiple Bit Upset* (MBU)** : Les MCU et les MBU sont la conséquence des phénomènes de diffusion et de collection de charges dans plusieurs cellules mémoires. Ils sont causés par un unique impact de particule. Les cellules appartiennent au même mot dans le cas des MBUs. La faute ne peut alors pas être corrigée par un code correcteur simple (un bit). Dans le cas des MCUs les bits n'appartiennent pas au même mot mémoire [134].

Le nombre de cellules affectées dépend de l'énergie de la particule, de son angle d'incidence et de la façon dont est implantée la cellule mémoire. En conséquence, prévoir l'occurrence ou la probabilité d'occurrence des MBUs avant placement/routage est impossible.

Parmi les modèles de fautes cités ci-dessus, il existe d'autres modèles ; citons par exemple la faute due au bruit et d'autres fautes par attaque volontaire, qui ne sont pas utilisables par la simulation RTL, mais nous le citons juste pour montrer l'existence d'autres modèles de fautes qui correspondent à des techniques d'injection de fautes différentes ; parmi ces modèles nous citons :

Faute due au bruit : Les modèles SET et SEU ont été largement utilisés pour caractériser l'incidence des impacts de particules. Cependant, à notre connaissance, la modélisation des phénomènes liés à la perte de l'intégrité du signal n'a pas encore été étudiée. Bien que des modèles de fautes pour les phénomènes de couplage aient été définis dans [135] ils ne concernent que les interconnexions entre les blocs d'un système.

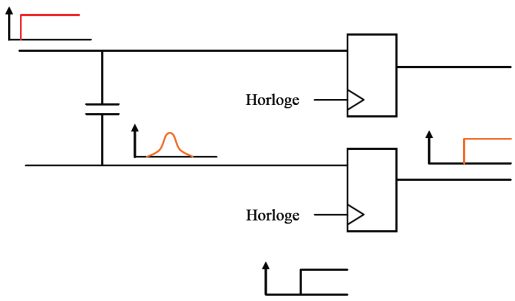


FIG. 2.15 – Faute due au couplage

Le couplage parasite et le bruit de commutation produisent des pics de courant dans les parties de logique combinatoire et dans les interconnexions (figure 2.15). Ils peuvent donc être modélisés avec le modèle de faute SET. En effet un pic créé sur une ligne victime peut se propager à travers la logique combinatoire et être mémorisé par une cellule mémoire, i.e. un SET.

Les effets de la variation de l'horloge sont plus proches du modèle SEU. Considérons un décalage d'horloge *(clock skew)* pour une cellule mémoire comme représenté sur la figure 2.16. δ_1 est le temps de calcul de la logique combinatoire. Si le décalage de l'horloge est δ_2 et si l'entrée change à $T + \delta_1$ ($\delta_2 \geq \delta_1$), alors la nouvelle valeur d'entrée sera mémorisée à $T + \delta_2$. Dans ce cas, le basculement de la valeur mémorisée se produit entre deux fronts d'horloge *idéaux*. Cette faute peut être considérée équivalente à une faute transitoire de type SEU [99].

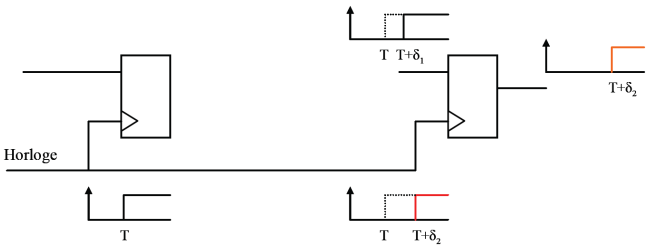


FIG. 2.16 – Faute due au variations de l'horloge

Faute par attaque volontaire : La sécurité est une propriété de plus en plus recherchée pour de nombreuses applications (cartes de crédit, télécommunications...) il devient indispensable de considérer les attaques intentionnelles comme une source de fautes. Par exemple, les attaques par fautes sont utilisées pour modifier le comportement d'un système et ainsi récupérer des données censées rester secrètes. Cette technique est appelée *Differential Fault Analysis* (DFA) [99].

Des fautes transitoires peuvent être injectées en modifiant l'environnement du circuit de différentes façons. Variation de la tension d'alimentation, variation de l'horloge ou du reset, injections optiques (flashes, laser) font partie des techniques d'injection possibles.

Récemment, une nouvelle menace suscite beaucoup d'attention. Elle concerne les circuits relatifs à la sécurité, comme par exemple les primitives cryptographiques ou les cartes à puce. Il a été démontré que des attaques basées sur l'introduction de fautes peuvent permettre à un intrus d'accéder à des informations confidentielles (par exemple une clé secrète) ou de contourner des mécanismes de sécurité. Les menaces principales viennent donc des attaques induisant des fautes dans une zone limitée (puisque des attaques globales sur le circuit peuvent souvent être détectées par des capteurs), mais ces attaques ne se limitent pas à l'inversion d'un seul bit, elles peuvent provoquer des inversions multiples.

2.9.2 Injection de fautes

Un grand nombre de techniques d'injection de fautes ont été proposées pour la validation de circuits. Parmi celles-ci, certaines méthodes d'injection de type matériel ciblent des exemples réels de circuits après fabrication. Ces méthodes peuvent être cataloguées comme suit :

- **Injections de fautes au niveau broches :** peut utiliser un contact direct avec les broches du circuit, commme la technique MESSALINE [83] développée au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS), et les injecteurs de l'outil RIFLE [84] permettent également de détecter si les fautes injectées ont produit des erreurs. L'avantage de ces outils est qu'ils ne sont pas intrusifs et qu'ils ne modifient pas le fonctionnement temporel des circuits, mais le problème majeur est la définition de modèles représentatifs des fautes internes d'un circuit.

- **Corruption de la mémoire :** L'approche présentée dans [86] utilise un injecteur de fautes commercial : l'appareil DEFI. Celui-ci permet d'injecter des fautes (permanentes ou transitoires -

simples ou multiples) dans la mémoire programme d'un processeur pour valider les mécanismes de détection d'erreurs. Elle est trop limitée pour l'analyse de circuits actuels car les fautes ne peuvent être injectées que dans la mémoire programme.

– **Perturbation de l'alimentation :** basée sur l'ajout d'un transistor MOS entre l'alimentation et le port Vcc du circuit à analyser [88]. Elle permet d'injecter des fautes en modifiant la tension de grille de ce transistor. Cette approche se rapproche des phénomènes de chutes de tension *(IR drop)* qui peuvent provoquer des fautes transitoires.

– **Laser :** offre une très bonne contrôlabilité dans le temps et dans l'espace. Certains paramètres comme la fréquence d'opération ou la précision peuvent cependant être des facteurs limitants. La méthode proposée dans [90] offre une fréquence de fonctionnement de 50kHz et une précision spatiale de $0,1\mu m$. Le laser peut également être utilisé pour caractériser la sensibilité d'un circuit aux SEU [144]. La méthode se montre très efficace mais doit être approfondie pour les technologies actuelles.

D'autres approches à base de simulation [137] permettent une analyse plus tôt dans le flot de conception, à tous les niveaux, du plus haut (niveau fonctionnel) au plus bas (le comportement élémentaire, transistors, capacités...). La simulation utilise aujourd'hui beaucoup les langages de description de haut niveau tels que VHDL et Verilog. Ces langages permettent la description de systèmes très complexes aussi bien au niveau comportemental (algorithmes, équations booléennes...) qu'au niveau structurel (niveau portes).

Des travaux ont récemment été menés afin d'effectuer l'analyse de sûreté au niveau système dans un environnement de conception conjointe matériel/logiciel "co-design" [148–150] (description SystemC, Esterel, POLIS ou utilisation d'un modèle comportemental d'architecture (ISA)). D'autres simulations au niveau instructions existent, où deux approches sont possibles : la modélisation ISS *(Instruction Set Simulation)* et la modélisation ISA *(Instruction Set Architecture)*. L'approche ISS permet de d'observer le contenu des registres utilisateur et l'approche ISA considère les transferts de registres [140] ; elle est donc plus détaillée que l'approche ISS qui considère les transactions.

La simulation à haut niveau de description (*High-level Description Language* - HDL) permet de vérifier un circuit de façon fonctionnelle avant synthèse. La simulation niveau portes permet de vérifier également le "timing" du circuit. Le contre poids est que les simulations niveau portes sont approximativement dix fois plus lentes que les simulations RTL, voire davantage.

Dans la suite de ce document, nous nous intéresserons à l'utilisation des langages de haut niveau, plus particulièrement sur les approches d'injection de fautes par simulation RTL car la mise en œuvre d'un tel simulateur est la plus réaliste.

Simulation RTL : L'injection de fautes lors de la simulation du modèle RTL (VHDL ou Verilog) du circuit à analyser peut se faire de deux façons ; soit en utilisant les commandes du simulateur pour forcer certains signaux internes, soit en utilisant une description modifiée du circuit permettant

l'injection de fautes.

Exploitation des commandes du simulateur : Lors d'une simulation RTL pour une description du circuit à analyser il est possible d'utiliser les commandes du simulateur ou les routines d'un langage spécifique (TCL par exemple) afin d'injecter une ou plusieurs fautes. Pour ce faire, un outil d'injection de fautes (MEFISTO) *(Multi-level Error/Fault Injection Simulation Tool)* [138, 139] utilise les commandes du simulateur, la manipulation de signaux et la manipulation de variables pour l'injection. La simulation se déroule normalement jusqu'à ce que le cycle d'injection soit atteint et que tous les *process* soient arrêtés au niveau d'une instruction *wait*. Pour l'injection d'une faute dans un signal (au sens VHDL), celui-ci est déconnecté de sa source et forcé à une nouvelle valeur jusqu'à la fin de la durée de la faute. Dans ce cas, il est possible d'altérer la valeur d'un signal ou d'une variable déclarée dans une description VHDL. Cette technique permet de ne pas modifier le modèle VHDL mais son application dépend des fonctionnalités offertes par le langage de commande du simulateur. La manipulation de signaux cible l'injection de fautes de collage permanentes et transitoires dans une description structurelle et la manipulation de variables vise l'injection de *bit-flips* au niveau comportemental.

Un autre outil developé par l'Université de Turin exploite les mécanismes de débogage du simulateur, en l'occurrence ModelSim, et l'interface proposée par le langage TCL *(Tool Command Language)* [151]. Une liste de fautes est générée par analyse de la description comportementale (VHDL RTL) en fonction du modèle de faute sélectionné. Le "simulateur de fautes" se compose d'un ensemble de routines qui interagissent avec le simulateur. La commande utilisée est le *breakpoint* ; simple pour les variables et double pour les signaux. Cette approche a été étendue à l'injection de "bit-flips" dans les éléments mémoires [152].

Simulation avec description instrumentée : Le laboratoire LAAS a développé une approche à base d'instrumentation, MEFISTO-L, parallèlement à une approche utilisant les commandes du simulateur [153].

L'instrumentation se traduit par l'addition de saboteurs et de sondes. Un saboteur est un composant VHDL additionnel qui, lorsqu'il est activé, altère la valeur ou les caractéristiques temporelles d'un ou de plusieurs signaux alors que les sondes permettent d'observer la valeur des signaux. Les mutants ne sont pas supportés par MEFISTO-L. La description originale est d'abord analysée pour générer les modèles de fautes, les cibles potentielles et les signaux à observer.

De la même façon, Gracia et al. proposent dans [154] des approches basées sur l'utilisation de saboteurs, et l'utilisation de mutants. Un grand nombre de saboteurs est considéré, jusqu'au saboteur "n bits bidirectionnel". L'implantation de mutants peut se faire de manière statique ou dynamique. De façon statique, le mécanisme de configuration du VHDL permet de compiler une configuration spécifique qui reste la même durant toute la simulation.

Cette approche permet d'injecter des fautes permanentes et des fautes transitoires. Les mutants permettent logiquement l'utilisation de modèles de fautes plus complexes mais les temps de simulation augmentent alors considérablement.

La génération de mutants permettant des injections avec des modèles de faute plus complexes

a également été étudiée au TIMA [155]. Les mutants présentés permettent notamment l'injection de transitions erronées dans une machine à états finis ou un organigramme de contrôle. Le ou les mutants générés sont ensuite simulés lors des campagnes d'injection.

Simulation au niveau portes : Parallèlement aux approches de haut niveau, des techniques d'injection ont été développées pour des descriptions au niveau portes (après synthèse). Parmi celles-ci nous pouvons citer FAST, VERIFY.

En 1996, un outil appelé FAST (*FAult Simulator for Transients*) a été proposé par [156]. Il se compose en fait de deux simulateurs distincts. Le premier simule le circuit jusqu'au moment où la faute injectée est capturée par une cellule mémoire. Le second simulateur (extension de l'outil PROOFS) est utilisé à partir de ce point afin d'observer les manifestations de la faute sur les sorties du circuit. L'intérêt de l'approche à deux simulateurs est d'accélérer la simulation après que la faute a été mémorisée puisqu'à partir de ce point il n'est plus nécessaire d'avoir une grande précision temporelle.

Afin d'obtenir le modèle de faute, au niveau portes, le plus proche possible d'un SEU, des simulations SPICE ont été menées. Elles ont permis d'obtenir des données précises comme la durée nécessaire de la faute, les délais des portes, et plus généralement le comportement des bascules *(Flip-Flops)* et des verrous *(latches)*. L'outil VERIFY *(VHDL-based Evaluation of Reliability by Injecting Faults efficientlY)* est basé sur une extension du langage VHDL et permet l'injection de fautes au niveau portes mais également au niveau RTL [99].

D'autres types d'injection existents comme l'**injection de fautes avec instrumentation** ; le principe est le même que pour la simulation qui repose sur la modification de la description initiale afin de permettre l'injection de fautes. Le circuit instrumenté est ensuite synthétisé puis émulé.

La pertinence du prototypage pour l'analyse de la SdF a été mise en avant au TIMA en 1999 [158] ; des modèles comportementaux sont également présentés comme des techniques performantes pour l'analyse de la SdF.

Une approche d'injection de fautes à base de FPGA a été développée par Université de Turin comme présenté dans [159]. Elle permet l'injection de *bit-flips* dans les éléments mémoires d'une version instrumentée du circuit à analyser. Dans [160], l'outil FIFA *(Fault Injection by means of FPGA)* est amélioré et met en œuvre des techniques dédiées aux microprocesseurs. Une autre technique a récemment été développée par le groupe microélectronique de l'université de Madrid, un environnement pour FPGA pouvant mettre en œuvre par un masque est appliqué aux éléments mémoires du circuit pour inverser le contenu de ceux qui sont ciblés [159]. Des efforts importants ont été menés pour intégrer au maximum la partie injection de facon matériel [162] ; par exemple, l'approche dite multiplexée temporellement *(time-multiplexed)* dont le but est de réduire les temps de campagne en arrêtant chaque expérience lorsque les effets de la faute injectée disparaissent.

2.10 Conclusion

Dans ce chapitre, nous avons présenté les notions de base de la SdF, et les techniques de tolérance aux fautes. Parmi ces techniques, nous avons présenté la méthode “duplication avec comparaison”. Cette méthode est utilisée par la suite à titre de comparaison pour évaluer nos architectures en terme de consommation de surface, . La conception des systèmes intégrés sécurisés nécessite non seulement de minimiser le surcoût matériel mais également de pouvoir valider/évaluer l’approche de tolérance retenue. Pour cela, nous avons présenté les approches d’évaluation de la tolérance par injection de fautes.

Concernant les injections basées sur le matériel, l’inconvénient est de devoir fabriquer le circuit avant de pouvoir injecter les fautes. Avec un methode d’injection à base de simulation, il est possible d’obtenir un évaluation de la SdF du système “très tôt” dans le flot de conception. Parmi les approches citées, nous avons ciblé la technique d’injection de fautes par simulation RTL qui est tout à fait appropriée pour valider nos architectures. Pour l’injection de fautes au niveau RTL, nous avons précisé l’ensemble d’erreurs susceptibles d’attaquer les éléments de mémoires, à savoir les effets d’une particule isolée SEE *(Single Event Effect)*. Nous nous sommes intéressés aux fautes transitoires, non permanentes, et en particulier aux perturbations correspondant respectivement à l’inversion de bit unique SEU *(Single Event Upset)* ou multiple MBU (*Multiple Bit Upset*).

II. ETUDE ARCHITECTURALE ET IMPLANTATION

Chapitre 3

Codeur convolutif récursif haut débit

3.1 Introduction

Parmi les objectifs de cette thèse figure la conception d'une nouvelle architecture parallèle-pipeline de codeur pour les codes convolutifs, que nous proposons dans ce chapitre. La première partie de ce chapitre décrit le principe général de fonctionnement du codeur rapide. Ensuite, nous poursuivons l'étude en identifiant les désavantages des techniques de parallélisation habituellement utilisées pour les deux schémas MTO et OTM de codeur convolutif. À partir de ces descriptions, de nouvelles architectures parallèles-pipeline seront proposées pour chacun des schémas MTO et OTM et décrites par les équations qui régissant leur fonctionnement. Pour évaluer les performances de ces codeurs haut débit, un modèle générique décrit en VHDL au niveau d'abstraction RTL (Register Transfert Level) a été synthétisé sur des FPGA de type Stratix-II d'Altera. Les résultats expérimentaux obtenus pour différentes implantations sont présentés en termes de débit (nombre de bits traités par seconde) et de surface consommée (nombre de cellules logiques utilisées) et interprétés.

Un codeur convolutionnel récursif pouvant être vu comme un filtre à réponse impulsionnelle infinie RII, une adaptation de la nouvelle méthode de parallélisation est proposée pour les filtres RII. Les résultats expérimentaux obtenus par une implantation sur FPGA sont présentées dans la deuxième partie de ce chapitre.

3.2 Architectures parallèles-pipeline

Avec l'exigence de débits de données de plus en plus élevés pour les applications de transmission de données, il existe un réel besoin d'architectures nouvelles efficaces (consommant moins de surface et plus performantes en débit) pour les circuits dédiés aux codes détecteur d'erreurs EDC et aux codes correcteur d'erreurs ECC. Comme on peut s'y attendre, la forte augmentation des requis en termes de performance des applications de transmission débouche sur un renforcement des contraintes concernant les architectures de codeur et de décodeur. Une approche en apparence intuitive pour satisfaire ces objectifs consiste à employer des techniques architecturales parallèles-pipeline.

3.2.1 Principe des techniques parallèles-pipeline

Une architecture synchrone série permet de traiter un bit par cycle d'horloge pour une fréquence d'horloge $f_{clk,ser}$, le débit D_S du codeur sera donc $(1\ bit \cdot f_{clk,ser})$. La parallélisation de l'architecture doit permettre de traiter p bits par cycle d'horloge pour une fréquence $f_{clk,par}$, le débit D_p sera alors égal à $(p\ bits \cdot f_{clk,par})$. Pour avoir une augmentation effective du débit, il est souhaitable que la fréquence $f_{clk,par}$ de l'architecture parallèle soit aussi proche que possible de $f_{clk,ser}$:

$$f_{clk,par} \simeq f_{clk,ser} \tag{3.1}$$

Avant de poursuivre, nous introduisons, par un exemple simple, les notions de parallélisme combinatoire et pipeline.

– l'additionneur de la figure 3.1 permet d'effectuer l'addition de deux opérandes de 10bits, la longueur de son chemin critique est ℓ_a ;

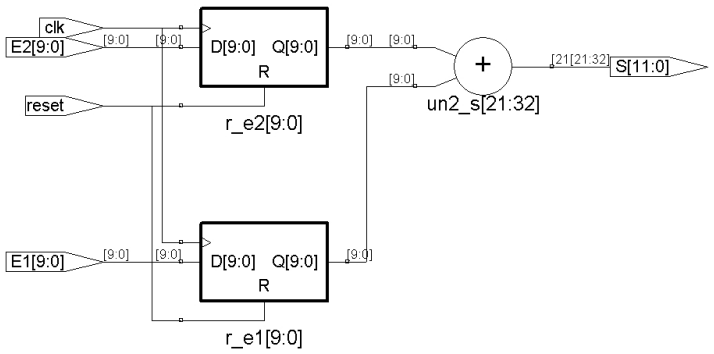


FIG. 3.1 – Additionneur à 2 opérandes.

– la figure 3.2 représente une structure combinatoire d'un additionneur à 4 opérandes, il permet de calculer en parallèle 4 opérandes. Cependant la longueur ℓ_c de son chemin critique est : $\ell_c > \ell_a$. Dans ce cas, on parle d'implantation parallèle purement combinatoire ou de parallélisme spatial ;
– la structure pipeline de l'additionneur représentée dans la figure 3.3 permet de traiter en parallèle 4 opérandes, la longueur ℓ_p de son chemin critique étant $\ell_p = \ell_a$. Dans ce cas on parle d'implantation parallèle pipeline ou de parallélisme temporel.

Lors d'une implantation parallèle purement combinatoire, deux limitations sont à considérer :

– de manière générale, le chemin critique induit par le bloc combinatoire dégrade considérablement la fréquence de fonctionnement $f_{clk,par}$:

$$p \nearrow \Rightarrow\ f_{clk,par} \searrow \Rightarrow\ f_{clk,par} \ll f_{clk,ser} \tag{3.2}$$

– la surface occupée S_o par le modèle parallèle augmente avec le degré de parallélisme.

$$p \nearrow \Rightarrow\ S_o \nearrow \tag{3.3}$$

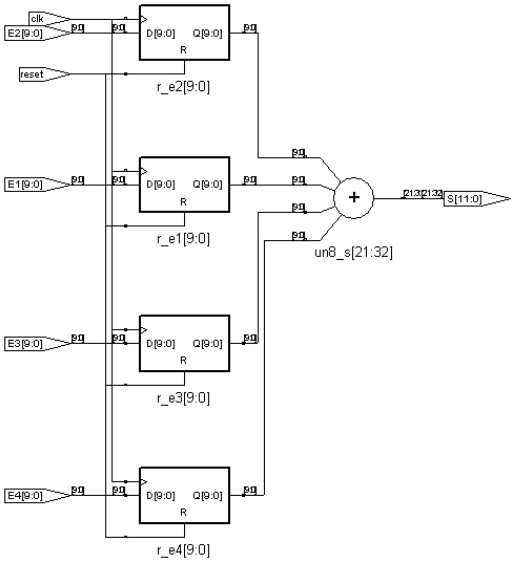


FIG. 3.2 – Structure combinatoire d'un additionneur à 4 opérandes.

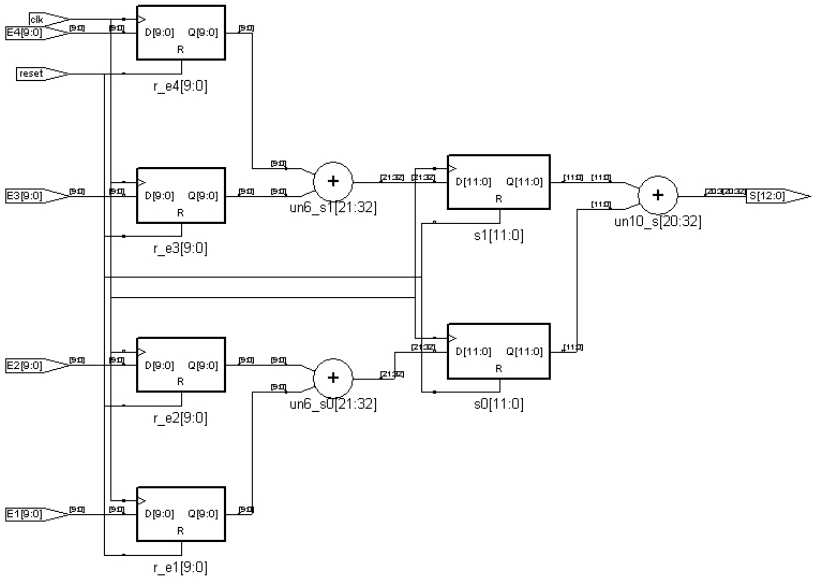


FIG. 3.3 – Structure pipeline d'un additionneur à 4 opérandes.

Notre objectif étant d'augmenter efficacement le débit, la première contrainte rend l'approche purement combinatoire peu pertinente. En effet, l'idéal est d'avoir autant que possible une augmentation du débit proportionnelle au degré de parallélisme (fig. 3.4, cas idéal). Malheureusement, dans le cas général la relation proportionnelle se dégrade rapidement au-delà d'un certain degré de parallélisme limite p_{limite} (fig. 3.4, approche combinatoire). Le débit ne pouvant augmenter de manière constante au delà de p_{limite}, le parallélisme combinatoire devient sans intérêt au-delà de cette valeur.

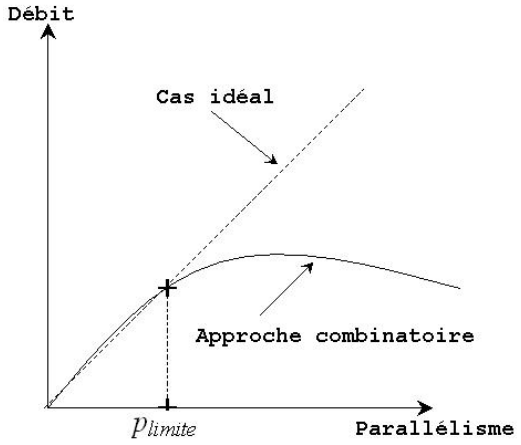


FIG. 3.4 – Courbes caractéristiques du débit en fonction du degré de parallélisme.

3.2.2 Inconvénients des architectures existantes

Plusieures architectures parallèles-pipeline de codeurs et décodeurs ont été proposées par le passé pour divers ECC, y compris pour les codes CRC [51–57], BCH binaires, RS [58–62], et pour les DCCS à logique majoritaire [54, 63]. Des architectures systoliques ont été expérimentés pour réaliser une convolution rapide [64–66], opération centrale dans les codes convolutifs.

Dans [64], une architecture rapide à été introduite, dédiée aux deux types MTO et OTM de codeur convolutif, et s'appuyant sur la technique parallèle-pipeline présentée dans [57]. Bien que toutes ces approches réussissent généralement à satisfaire les contraintes de débit, elles ont généralement tendance à générer une consommation matérielle très importante.

Comme nous l'avons présenté dans le premier chapitre (section 1.4.4), la méthode proposée dans [64] dédiée aux codeurs convolutifs MTO et OTM consiste à combiner à l'approche combinatoire avec une technique pipeline afin de maximiser la valeur de p_{limite} en minimisant d'une part, la longueur du chemin critique généré par le circuit combinatoire et d'autre part, en la rendant indépendant du degré de parallélisme. De cette manière, toute augmentation du degré de parallélisme p entraînera automatiquement une augmentation du débit. Cette approche permet d'atteindre avec succès de très hauts débits de données. Cependant, elle souffre des différentes points faibles que nous résumons comme suit :

– trois blocs sont nécessaires pour implanter un codeur parallèle-pipeline. La surface consommée de ces blocs dépend fortement du degré de parallélisme p. Ceci implique qu'avec un degré p assez élevée (par exemple $p = 32$), la quantité du matérielle requise devient très importante ;

– le chemin critique, situé dans le bloc GEA (voir section 1.4.4), est indépendant de p. Il est liée à n, où n est le dégré du polynôme de retour $G(x)$ [64]. Ceci explique que le chemin critique soit de l'ordre de n quel que soit le dégrée de parallélisme p. En particulier, pour des polynômes de degrés élevés avec et $n \geq p$, le chemin critique est très important, ce qui limite largement l'intérêt du parallélisation ;

– cette approche requiert le pré-calcule des coefficients des matrices nécessaires préalablement à l'implantation, ce qui interdit une configuration en ligne du codeur (changement en ligne des coefficients des polynômes $G(x)$ and $H(x)$).

Pour éliminer ces contraintes, il est nécessaire de modifier la structure parallèle-pipeline existants de façon à :

– minimiser la surface consommé de l'architecture parallèle-pipeline. Cela peut se faire en fusionnant les trois blocs de la structure précédente (voir section 1.4.4) ;
– réduire le chemin critique en le indépendant du degré de polynôme $G(x)$, notamment pour les polynômes de dégrées élevés ;
– éliminer tous les matrices nécessitent un pré-calcule et empêchant par conséquent la configuration en ligne du codeur parallèle-pipeline.

Nous proposons ici une amélioration de l'approche parallèle-pipeline existante pour les codeurs MTO et OTM [67,68]. Deux architectures parallèles-pipeline CP_{mto} et CP_{otm} sont proposées respectivement pour chacune des variantes MTO et OTM du codeur convolutif. Du point de vue expérimental, les deux modèles de codeurs seront modélisés en VHDL et synthétisés sur FPGA.

Après implantation, les performances de chaque codeur seront évaluées en fonction de deux critères directement liés aux paramètres intrinsèques du composant cible. Dans notre cas, la cible est un FPGA de type Stratix-II d'Altera dont la fréquence maximale de fonctionnement est $f_{max} = 500\text{MHz}$. Par conséquent, l'évaluation du codeur se fera selon les deux critères suivant :

– fréquence de fonctionnement $f_{clk,par}$ qui conditionne le débit :

$$D_p \text{ (unité)} = p \times f_{clk,par} \tag{3.4}$$

– surface S_o occupée, mesurée en nombre de cellules logiques utilisées dans le FPGA.

À partir de ces deux paramètres, les courbes de compromis ($D_p,\ S_o$) permettront de comparer les performances des codeurs. Ainsi, pour une surface occupée S'_o le codeur le plus performant sera celui au débit le plus élevé. De même, pour un débit D'_p, le meilleur codeur sera celui à la plus petite surface occupée dans le FPGA.

3.3 Étude du codeur convolutif haut débit

Comme nous l'avons écrit précédemment, notre objectif est de concevoir des architectures rapides et à faible consommation en surface. Dans cette section, nous proposons deux architectures haut débit pour le codeur convolutif MTO et OTM. Les deux techniques parallèle et pipeline sont combinées entre elles pour atteindre un meilleur compromis entre performance (débit) et surface. Par rapport aux précédentes, cette nouvelle approche est très efficace, permettant une forte réduction de

la surface consommée (dans les deux cas MTO et OTM) pour des performances (débit) identiques voire supérieures. Dans la suite, l'architecture proposée parallèle-pipeline MTO (OTM) est noté par MTO(OTM)-PPFC (Parallèle-Pipeline à Faible Complexité), tandis que l'architecture MTO (OTM) précédente est noté par MTO(OTM)-PP (Parallèle-Pipeline).

3.3.1 Présentation du codeur convolutif MTO haut débit

Le schéma fonctionnel du codeur convolutif MTO haut débit est représenté dans la figure 3.5. Il est constitué de deux blocs RDR et GSP [67].

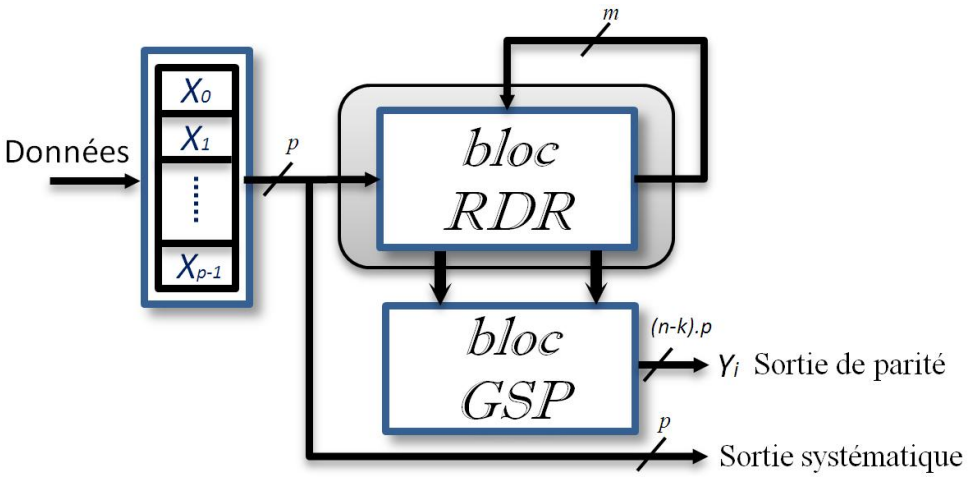


FIG. 3.5 – Schéma fonctionnel du codeur MTO parallèle-pipeline

– **Réseau de Décalage de Retour (RDR)** ce bloc constitue une structure de décalage et accumulation de p bits d'entrées X_i via différentes couches des registres. À un instant donné, un nouvel état $(X_i)_{t+1}$ est calculé par l'addition de chaque bit de l'entrée $(X_i)_t$ au valeur $(V_i)_{t,t+1}$ correspondant. $(V_i)_{t,t+1}$ étant la valeur résultant de la combinaison linéaire des états actuels $(X_i)_{t+1}$ et des états précédens $(X_i)_{t-j}$ (j étant le décalage) par les $(n-1)$ coefficients correspondants du polynôme de la boucle du retour $G(x)$, où n est le dégré du polynôme $G(x)$. Cette combinaison est construite selon une topologie bien déterminée.

– **Génération des Sorties Pipelines (GSP)** dans ce bloc, p états de sortie sont calculés selon une autre combinaison des états $(X_i)_{t-j}$ générés par le bloc RDR et des m coefficients du polynôme de la branche directe $H(x)$, où m est le degré du polynôme $H(x)$.

Les degrés n et m étant pratiquement égaux, on utilise par la suite la notation n pour designer la taille de la mémoire du codeur. À noter que dans le cas d'un codeur MTO parallèle-pipeline, le chemin critique dépend des valeurs non nulles résultant de la multiplication des coefficients de $G(x)$ et de $H(x)$ par les états $(X_i)_{t-j}$ calculés précédemment, autrement dit par les coefficients non nuls de ces polynômes. Ce chemin critique est proportionnelle à la taille de la mémoire du codeur n. Si la taille

mémoire du codeur $n << p$, le chemin critique est faible, et un meilleur compromis débit/surface par rapport à l'architecture MTO existante est atteint. Lorsque le degré n tend à augmenter, le chemin critique correspondant augmente parallèlement. Dans ce cas, le chemin critique est important, et le meilleur compromis débit/surface peut être obtenu avec le codeur parallèle-pipeline de type OTM présenté ci-dessous.

3.3.2 Présentation du codeur convolutif OTM haut débit

Le schéma fonctionnel du codeur convolutif de type OTM est présenté dans la figure 3.6. Il est constitué d'un seul bloc EPC [68].

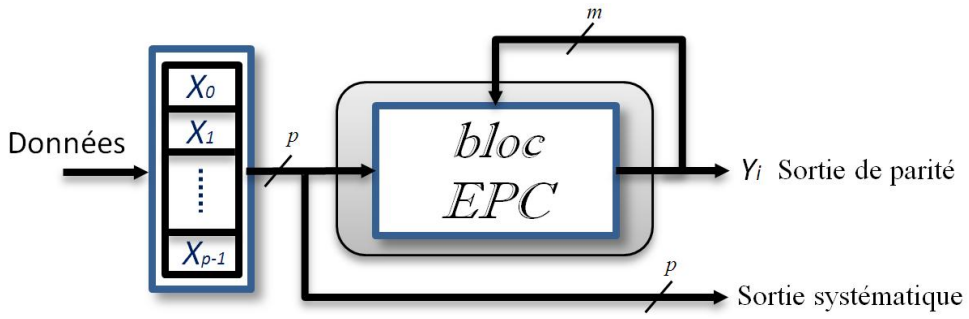


FIG. 3.6 – Schéma fonctionnel du codeur OTM parallèle-pipeline

– **Bloc de génération d'États de Pipeline Cumulées (EPC)** ce bloc génère, à chaque cycle d'horloge, les p sorties $(Y_i)_{t+1}$ du codeur OTM. Chaque sortie est calculée par la combinaison de p bits d'entrées $(X_i)_t$ multipliés par les coefficients du polynôme $H(x)$ et de p bits de sorties $(Y_i)_t$ multipliés par les coefficients du polynôme $G(x)$. Chaque coefficient de $G(x)$ et $H(x)$ est utilisé une seule fois, un registre contenant la valeur de la combinaison calculée à l'instant $(t-1)$ à laquelle va s'ajouter la combinaison actuelle de l'instant t, et ainsi de suite, jusqu'à utilisation de tous les coefficients de $G(x)$ et $H(x)$.

Le chemin critique du codeur OTM est proportionnel à p où p est le degré du parallélisme. Si le degré $p << n$, où n est la taille mémoire du codeur OTM, le chemin critique est faible et un meilleur compromis débit/surface par rapport à l'architecture OTM antérieure est atteint. Lorsque p augmente le chemin critique devient important, jusqu'à $p \geq n$, un codeur haut débit de type MTO sera utile pour avoir un bon compromis débit/surface.

3.4 Implantation du codeur convolutif haut débit

Les modèles de codeurs MTO et OTM parallèles ont été décrits en VHDL au niveau RTL. Il ont été implantés sur des FPGA de type Stratix II d'Altera à l'aide de l'outil Quartus II (version 7.1). La fréquence maximale de fonctionnement (f_{max}) de cette famille de FPGA est limitée à 500 MHz. Tous

les résultats obtenus en termes de fréquence de fonctionnement, pour chaque codeur MTO et OTM, seront comparés aux fréquences respectives des codeurs MTO et OTM correspondants proposés précédemment.

Les modèles de codeur parallèles CP_{mto} et CP_{otm} ont été décrits de façon générique, indépendamment de la technologie cible. Comparativement aux codeurs proposées précédemment, aucun précalcul de valeurs n'est nécessaire pour les deux nouveaux modèles CP_{mto} et CP_{otm}, une configuration en ligne des coefficients des polynômes $G(X)$ et $H(X)$ étant suffisante.

Afin d'évaluer l'efficacité des nouvelles architectures, les performance de celles-ci sont comparées à celles des anciennes architectures pour chaque type de codeur MTO ou OTM. Pour pouvoir les comparer aux architectures proposées précédemment qui nécessitent un précalcul des valeurs des matrices pour chaque implantation, une phase d'initialisation des paramètres génériques (codeur systématique ou non, taille m de la mémoire, valeur des polynômes générateurs, rendement R et degré de parallélisme p) s'effectue avant la synthèse par l'intermédiaire d'un paquetage généré par un programme de pré-calcul écrit en langage C. Le programme de précalcul a pour rôle de :

– calculer, à partir des paramètres de configuration, les matrices de transitions dans les deux cas du codeur parallèle déjà proposée CP_{mto} et CP_{otm} ;
– générer le paquetage de configuration VHDL associé au modèle générique du codeur parallèle.

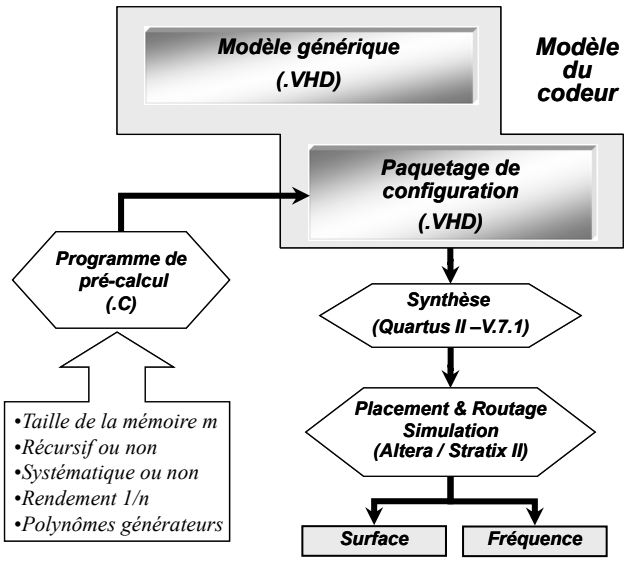


FIG. 3.7 – Flot de conception

Dans la fiqure 3.7, aucun programme de précalcul n'est nécessaire pour les nouvelles architectures. Il n'est nécessaire que pour les anciennes versions de codeurs parallèles. La comparaison de per-

formance entre deux architectures (nouvelle et précédente) sera réalisée sur le débit D (en $\mathrm{Gbits/s}$) calculé selon l'équation suivante :

$$D = p \cdot f_{max} \tag{3.5}$$

où p est le dégré de parallélisme et f_{max} la fréquence maximale de fonctionnement. Notre objectif est non seulement de concevoir des architectures rapides mais également d'optimiser la surface de ces architectures. Une comparaison de la consommation en surface (mesurée en termes de CLBs-*Configurable Logic Blocs*- dans l'FPGA) est donc réalisée.

Notations chaque codeur est définit par les paramètres (k, n, m, p) où k représente le nombre d'entrées du codeur, n représente le nombre de sorties, m est la taille de la mémoire du codeur, p est le niveau du parallélisme du codeur, G représente les coefficients du polynôme de la boucle récursive et H représente les coefficients du polynôme de la branche directe.

3.4.1 Implantation du codeur haut débit de type MTO (CP_{mto})

Les équations 3.6 et 3.7 relient pour le codeur MTO les états des entrées et des sorties de l'architecture série originelle (voir section 1.4.4) à ceux de la nouvelle architecture parallèle-pipeline. Les équations sont définies pour un degré de parallélisme p. Comme le montre l'équation 3.7, les sorties sont obtenus avec une latence de deux cycles d'horloge.

$$((X_0)_t, \cdots, (X_{p-1})_t) = ((X)_{p\cdot t+(p-1)}, \cdots, (X)_{p\cdot t}) \tag{3.6}$$

$$((Y_0)_t, \cdots, (Y_{p-1})_t) = ((Y)_{p\cdot (t-2)+p}, \cdots, (Y)_{p\cdot (t-2)+1}) \tag{3.7}$$

Le fonctionnement du codeur MTO, constitué des deux blocs RDR et GSP (voir section 3.3.1), peut facilement être décrit en introduisant les notations auxiliaires Z_i pour les lignes internes de l'architecture. Les équations décrivant l'évolution du codeur entre t et $t+1$ sont pour un parallélisme p [67] :

$$\forall i \in \{0, \cdots, p-1\},$$
$$(Z_i)_{t+1} = (X_i)_t + \sum_{k=1}^{m} g_k \cdot (Z_{i+k})_{t+1} \tag{3.8}$$
$$\forall i \in \{p, \cdots, p+m-1\},$$
$$(Z_i)_{t+1} = (Z_{i-p})_t \tag{3.9}$$
$$\forall i \in \{0, \cdots, p-1\},$$
$$(Y_i)_{t+1} = \sum_{k=0}^{m} h_k \cdot (Z_{i+k})_t \tag{3.10}$$

L'équation 3.8 représente la contribution des coefficients g_i de la boucle de retour $G(x)$ sur le pipeline de données d'entrées X_i. Les données d'entrées sont retardées à chaque cycle d'horloge selon l'équation 3.9. L'équation 3.10 décrit l'état des sorties Y_i obtenus par la contribution des coefficients h_i de polynôme de la branche directe $H(X)$.

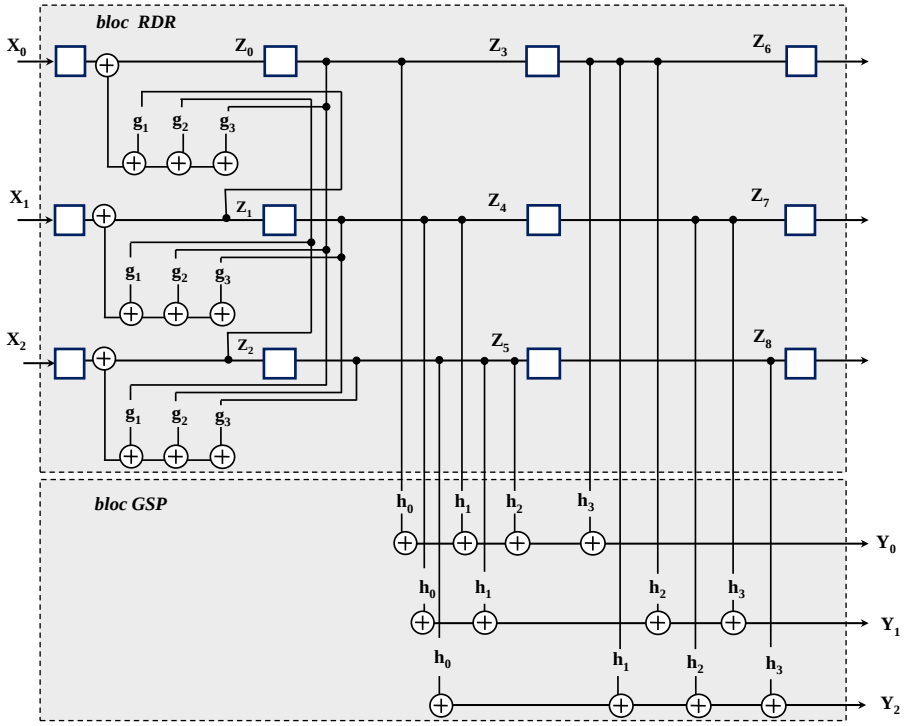


FIG. 3.8 – Codeur parallèle MTO pour $m = 3$ et $p = 3$

L'architecture globale du codeur parallèle CP_{mto} est représenté dans la figure 3.8 dans le cas où $m = 3$ et $p = 3$ [67].

Dans la figure 3.8, les entrées à l'instant $t + 1$ sont combinées à celles de l'instant t par l'intermédiaire des coefficients du polynôme de retour $G(X)$ (voir bloc RDR), conformément aux équations 3.8 et 3.9. Les sorties sont obtenues par une combinaison linéaire entre les bits du bloc RDR retardés et les coefficients du polynôme de direct $H(X)$ (voir équation 3.10). Par conséquent, le chemin critique est proportionnel à la taille mémoire du codeur m, et dépend plus précisément des coefficients non nuls de $G(X)$ et de $H(X)$.

3.4.2 Implantation du codeur haut débit de type OTM (CP_{otm})

Par analogie avec le codeur MTO, les équations qui relient le codeur OTM parallèle au codeur série correspondant sont définies ci-dessous :

$$((X_0)_t, \cdots, (X_{p-1})_t) = ((X)_{p\cdot t+(p-1)}, \cdots, (X)_{p\cdot t}) \tag{3.11}$$

$$((Y_0)_t, \cdots, (Y_{p-1})_t) = ((Y)_{p\cdot(t-1)+p}, \cdots, (Y)_{p\cdot(t-1)+1}) \tag{3.12}$$

Le fonctionnement du codeur OTM parallèle-pipeline est décrit par l'équation 3.13. Un seul bloc est nécessaire pour implanter le codeur (voir section 3.3.2). Cette équation relie les valeurs de sortie à l'instant $t+1$ aux valeurs d'entrées/sorties à l'instant t.

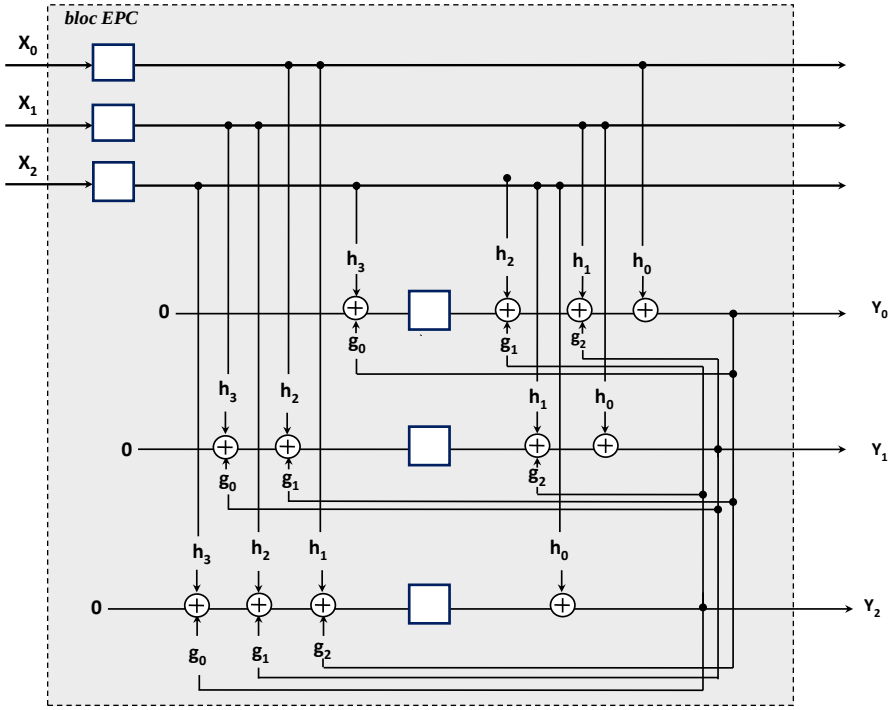


FIG. 3.9 – Codeur parallèle OTM pour $m = 3$ et $p = 3$

$$\forall i \in \{0, \cdots, p-1\},$$

$$\begin{aligned}(Y_i)_{t+1} &= \sum_{k=0}^{m} h_k \cdot X((i+k) \mod p)_{t-\lfloor (i+k)/p \rfloor} \\ &+ \sum_{k=0}^{m} g_k \cdot Y((i+p-k) \mod p)_{t+1-\lfloor (i+p-k)/p \rfloor}\end{aligned} \tag{3.13}$$

L'équation 3.13 représente la structure pipeline de calcul des sorties Y_i. Celles-ci sont obtenus par la contribution des entrées X_i multipliées par les coefficients g_i du polynôme $G(x)$, et les valeurs de sorties calculées au cycle d'horloge précédent multipliées par les coefficients du polynôme $H(x)$.

Comme pour le codeur MTO, les mêmes paramètres sont utilisés pour l'exemple de codeur OTM parallèle-pipeline, soit $m = 3$ et parallélisme $p = 3$. L'architecture globale correspondant à ce cas est représentée dans la figure 3.9.

Dans la figure 3.9, les sorties de l'instant $t + 1$ sont calculées à partir des entrées et sorties de l'instant t multipliées respectivement par les coefficients des polynômes $H(X)$ et $G(X)$, en respectant l'équation 3.13. Le chemin critique du codeur OTM parallèle-pipeline est proportionnel à $min(m, p)$, ou m est la taille mémoire du codeur et p le niveau de parallélisme.

3.5 Résultats expérimentaux

Les deux modèles du codeur parallèle-pipeline $CP2_{mto}$ et $CP2_{otm}$ ont été implantés sur un FPGA Stratix II. Les configurations choisies correspondent à des codeurs RSC de rendement $1/2$. Chaque modèle a été implanté pour les 3 degrés de parallélisme $p = 8$, 16 et 32. Les polynômes générateurs sont données dans le tableau 3.1. Leur choix est justifié par le fait qu'ils présentent les meilleures distances libres [163] pour une taille mémoire donnée. Les détails et les démonstrations relatifs à leurs choix sont donnés dans [163]. Il est à noter que les polynômes $G_1(x)$ et $G_2(x)$ sont utilisés respectivement les protocoles $X25 - CCITT$ pour le premier et $IBM - BISYNCH$ pour le deuxième.

Pour chaque modèle quatre configurations ont été testées avec les trois niveaux de parallélisation 8, 16 et 32. Les polynômes générateurs choisis sont indiqués dans le tableau 3.1.

TAB. 3.1 – Polynômes générateurs [50]

Notation	**Degré**	**Pôlynomes de la boucle récursif**
$G_1(x)$	16	$1 + x^2 + x^{15} + x^{16}$
$G_2(x)$	16	$1 + x^5 + x^{12} + x^{16}$
$G_3(x)$	32	$1 + x + x^4 + x^6 + x^8 + x^{10} + x^{12} + x^{13}$ $+x^{15} + x^{17} + x^{18} + x^{20} + x^{21} + x^{22} + x^{32}$
$G_4(x)$	32	$1 + x + x^2 + x^{22} + x^{32}$
Notation	**Degré**	**Pôlynomes de la branche directe**
$H_1(x)$	16	$1 + x + x^2 + x^4 + x^7 + x^9 + x^{10} + x^{14} + x^{16}$
$H_2(x)$	32	$1 + x + x^2 + x^3 + x^5 + x^{10} + x^{11} + x^{12} + x^{13} + x^{15} + x^{19} + x^{20}$ $+x^{21} + x^{25} + x^{27} + x^{31} + x^{32}$

TAB. 3.2 – Codeurs testés [50]

Codeur	**Fonction de transfert**
$(G_1(x),\ H_1(x))$	$H_1(x)/G_1(x)$
$(G_2(x),\ H_1(x))$	$H_1(x)/G_2(x)$
$(G_3(x),\ H_2(x))$	$H_2(x)/G_3(x)$
$(G_4(x),\ H_2(x))$	$H_2(x)/G_4(x)$

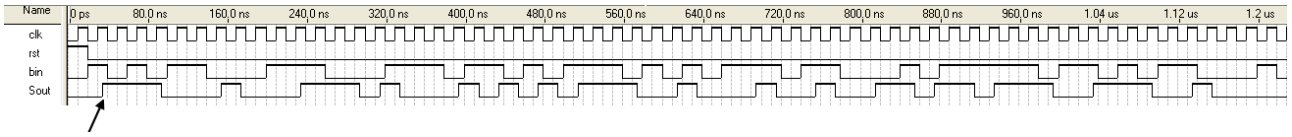


FIG. 3.10 – Simulation du codeur MTO série pour les polynômes G_1/H_1 de tableau 3.1

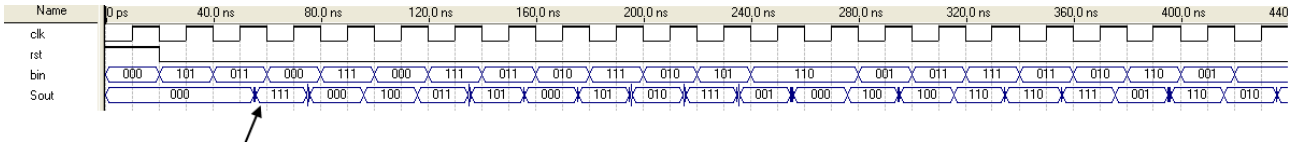


FIG. 3.11 – Simulation du codeur MTO parallèle-pipeline pour $m = 16$ et $p = 3$ et les polynômes G_1/H_1 de tableau 3.1

Des simulations ont été réalisées pour les architectures série et parallèle-pipeline des codeurs MTO et OTM, correspondant respectivement aux polynômes générateurs $G_1(X)/H_1(X)$ et $G_2(X)/H_1(X)$ du tableau 3.1. Les figures 3.10 et 3.11 représentent les courbes de simulation des codeurs MTO série et parallèle-pipeline avec $p = 3$ dans le cas d'une mémoire de taille $m = 16$, les mêmes séquences des bits étant utilisées en entrée.

La flèche sur les sorties sont des codeurs série (Fig 3.10 et parallèle 3.11), indique le point de début pour la comparaison entre les séquences produites par les deux codeurs. Il est clairement observable que la même séquence est obtenue, la seule différence étant dans la latence du codeur série (un cycle d'horloge) et le codeur parallèle (deux cycles d'horloge).

Le codeur OTM a lui aussi été testé dans les mêmes conditions, les résultats de simulation étant présentés dans les figures 3.12 et 3.13. Là encore, les mêmes séquences de bits sont obtenues en sortie $Sout$ pour la même séquence en entrée, avec une latence d'un cycle d'horloge dans les deux cas.

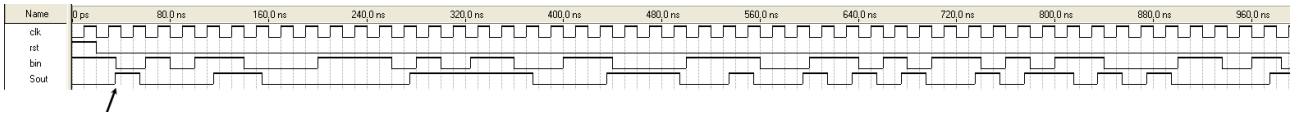


FIG. 3.12 – Simulation du codeur OTM série pour les polynômes G_2/H_1 de tableau 3.1

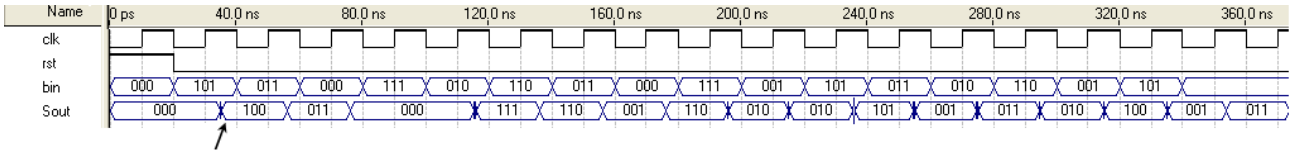


FIG. 3.13 – Simulation du codeur OTM parallèle-pipeline pour $m = 16$ et $p = 3$ et les polynômes G_2/H_1 de tableau 3.1

3.5.1 Performances du codeur CP_{mto}

L'architecture du codeur MTO parallèle-pipeline CP_{mto} a été testée pour les trois degrés de parallélisme, 8, 16 et 32 [67].

Sur les courbes présentés ci-dessous :

– la surface occupée ou taux d'occupation S_o dans le FPGA est donnée en nombre de cellules logiques CL utilisées ;

– le débit D (nombre de bits traités par seconde) est donné en $Gbits/s$. Il est calculé d'après l'équation 3.4.

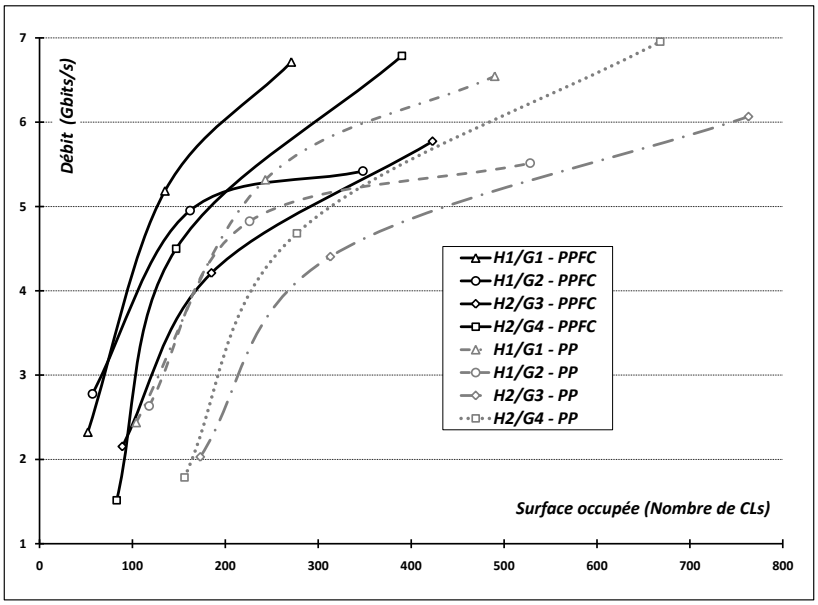


FIG. 3.14 – Comparatif compromis vitesse/surface pour les codeurs MTO-PPFC et MTO-PP

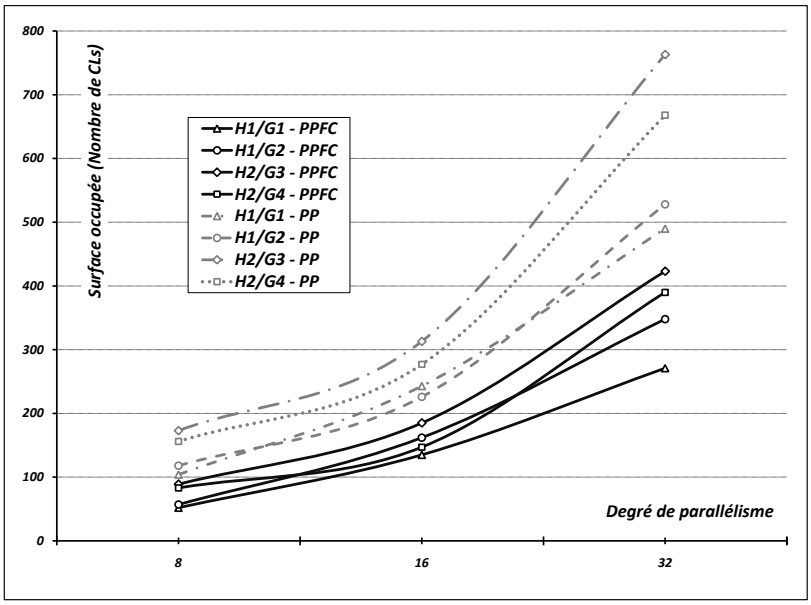


FIG. 3.15 – Comparatif compromis surface/parallélisme pour les codeurs MTO-PPFC et MTO-PP

De l'observation des courbes de la figure 3.14, on peut constater que, pour n'importe quel degré de parallélisme p, le compromis débit/surface du codeur MTO-PPFC proposé (correspondant aux lignes

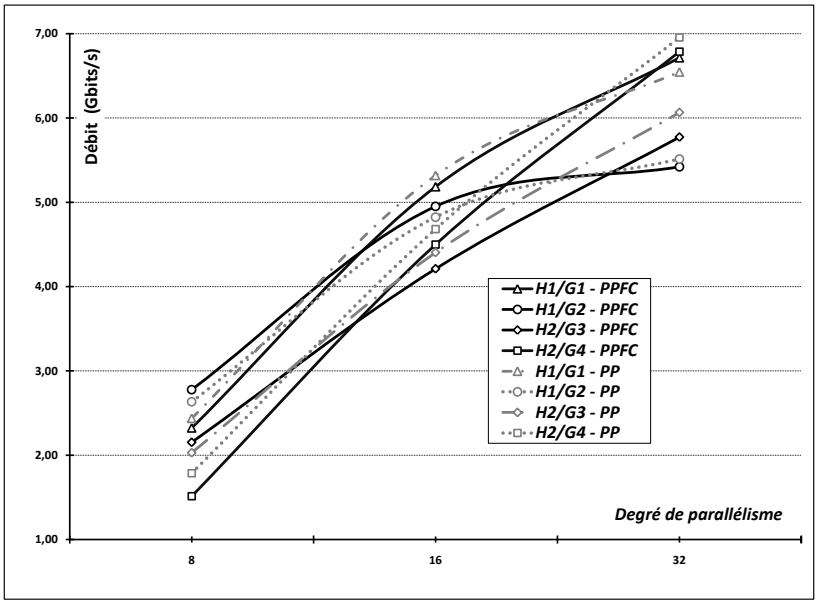


FIG. 3.16 – Comparatif compromis vitesse/parallélisme pour les codeurs MTO-PPFC et MTO-PP

continues dans la figure 3.14) est meilleur que celui du codeur MTO-PP précédent (correspondant aux lignes pointillés dans la figure 3.14).

De plus, la surface S_o requise pour le codeur PPFC est bien moins importante que pour le codeur PP (voir figure 3.15), réduction de surface consommé allant jusqu'à 50% dans le cas où $p = 32$. Ceci est une conséquence directe du fait que la nouvelle architecture parallèle-pipeline nécessite moins de registres dans sa structure à deux blocs (voir section 3.3.1).

La figure 3.16 montre les compromis vitesse/parallélisme pour les codeurs PPFC et PP. Il est claire que pour un degré de parallélisation identique, les débits atteints par les deux types de codeurs sont toujours comparables. Ces légères différences étant liées à la répartition des coefficients non nuls dans les polynômes générateurs $G(X)$ et $H(X)$ choisis.

De manière générale, la fréquence de fonctionnement décroît en fonction du degré de parallélisme. Pour évaluer la qualité des architectures étudiées pour les codeurs MTO, le tableau 3.3 met en évidence la dégradation de la performance en fonction du degré de parallélisme p.

TAB. 3.3 – Dégradation de $f_{mto\ (p)}$ en fonction du degré de parallélisme p

Codeur	$\Delta f_{mto\ (8)}\%$	$\Delta f_{mto\ (16)}\%$	$\Delta f_{mto\ (32)}\%$
$H_1(x)/G_1(x)$	46 %	47 %	63%
$H_1(x)/G_2(x)$	62%	43%	57%
$H_2(x)/G_3(x)$	41 %	35%	58%
$H_2(x)/G_4(x)$	30%	38%	66%
Moyenne	$\overline{(\Delta f)}_{mto\ (8)}\%$	$\overline{(\Delta f)}_{mto\ (16)}\%$	$\overline{(\Delta f)}_{mto\ (32)}\%$
	45%	41%	61%

$\Delta f_{x\ (p)}\%$ [1] représentant le taux de dégradation, en pourcentage de la fréquence $f_{x\ (p)}$ par rapport à la fréquence maximale de fonctionnement f_{max}, pour un degré de parallélisme p et une architecture x. Dans le cas présent $f_{max} = 500$ MHz, correspondant à la fréquence de fonctionnement maximale des FPGA de type Altera/Stratix II. Le taux de dégradation est calculé d'après l'équation (3.14).

$$\Delta f_{x\ (p)}\% = \frac{f_{max} - f_{x\ (p)}}{f_{max}} \times 100 \tag{3.14}$$

La dégradation moyenne de la fréquence de fonctionnement $\overline{(\Delta f)}_{x\ (p)}\%$, pour un degré de parallélisme p et une architecture x, est calculée par rapport à la valeur maximale f_{max} selon :

$$\overline{(\Delta f)}_{x\ (p)}\% = \frac{\sum_i^n \Delta f^i_{x\ (p)}\%}{n} \tag{3.15}$$

où $\Delta f^i_{x\ (p)}$ est le taux de dégradation $\Delta f_{x\ (\wp)}$ pour le codeur i et n représente le nombre de codeurs évalués.

On déduit des résultats du tableau 3.3 que la dégradation moyenne de la fréquence $\overline{(\Delta f)}_{mto\ (p)}\%$ dans le cas d'un codeur MTO est de :

- 45% pour une réduction par 8 du nombre de cycles d'horloge ;
- 41% pour une réduction par 16 du nombre de cycles d'horloge
- 61% pour une réduction par 32 du nombre de cycles d'horloge

On peut déterminé le bilan global pour chaque degré de parallélisme p, en calculant l'augmentation effective $A_{eff\ (p)}$ du nombre de bits traités par rapport au traitement série à $f_{max} = 500$ MHz, soit 500 Mbits/s.

L'augmentation effective $A_{eff\ (p)}$ est calculée selon l'équation (3.16) [2].

$$A_{eff\ (p)} = (1 - \overline{(\Delta f)}_{x\ (p)}) \times p \tag{3.16}$$

Dans le cas du codeur parallèle MTO, on obtient respectivement pour chaque degré de parallélisme les valeurs suivantes :

$$\begin{aligned} \widetilde{\Delta f}_{mto\ (8)} &\approx 0,45 \Rightarrow A_{eff\ (8)} = 0.55 \times 8 = 4,4 \\ \widetilde{\Delta f}_{mto\ (16)} &\approx 0,41 \Rightarrow A_{eff\ (16)} = 0.59 \times 16 = 9,44 \\ \widetilde{\Delta f}_{mto\ (32)} &\approx 0,61 \Rightarrow A_{eff\ (32)} = 0.39 \times 32 = 12,48 \end{aligned} \tag{3.17}$$

3.5.2 Performances du codeur CP_{otm}

L'architecture parallèle-pipeline du codeur OTM a elle aussi été étudiée en termes de performance et de surface, pour les différents polynômes générateurs de tableau 3.1. Les notations utilisées pour les courbes ci-dessous ont les mêmes que pour le cas MTO.

[1] La notation $\Delta f_{x\ (p)}$ signifie $\Delta f_{mto\ (p)}$ ou $\Delta f_{otm\ (p)}$.
[2] La notation $()_x$ signifie $()_{mto}$ ou $()_{otm}$

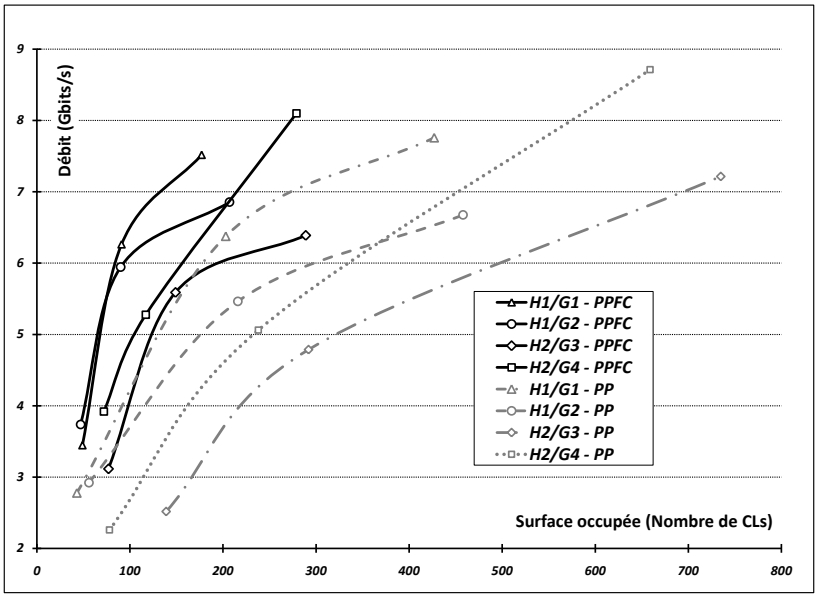


FIG. 3.17 – Comparatif compromis vitesse/surface du codeur OTM-PPFC et OTM-PP

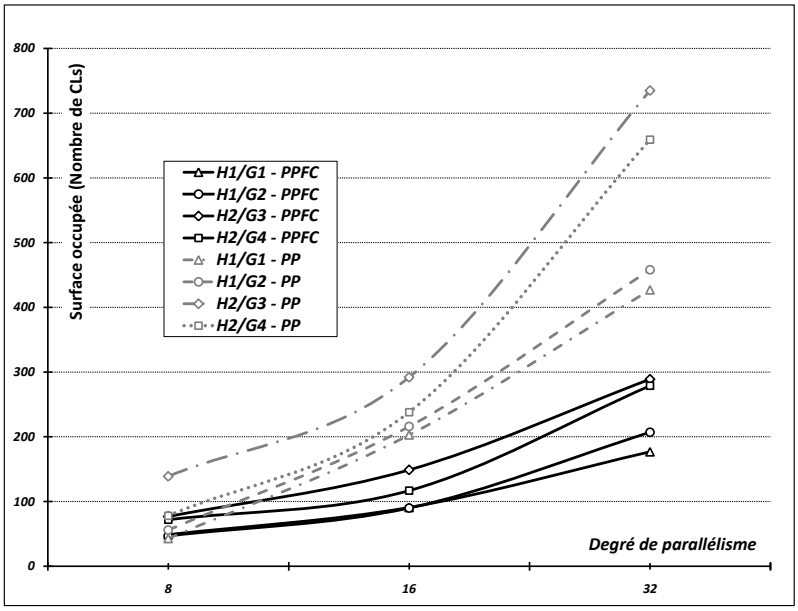


FIG. 3.18 – Comparatif compromis surface/parallélisme du codeur OTM-PPFC et OTM-PP

La figure 3.17 montre que, pour un niveau de parallélisme donné, le compromis vitesse/surface du codeur proposé OTM-PPFC (lignes continues) est le meilleur que celui du codeur précédent OTM-PP (lignes pointillés).

Le résultat le plus important est que la nouvelle architecture permet de fortes réductions de la surface consommée, jusqu'à 58% dans le cas d'un degré de parallélisme $p = 32$.

En outre, la figure 3.19 montre clairement que la nouvelle architecture offre des débits de données plus élevés. En effet, la plupart du temps, une amélioration significative de la vitesse est observée (jusqu'à $1,8$ fois plus rapide dans le meilleur des cas), en particulier lorsque $p < m$ (p étant le degré

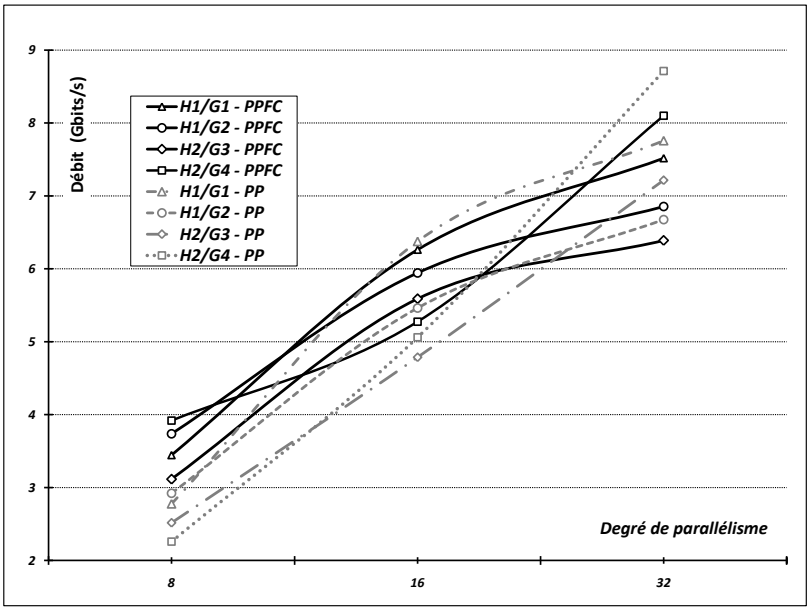


FIG. 3.19 – Comparatif compromis vitesse/parallélisme du codeur OTM-PPFC et OTM-PP

de parallélisme et m la taille mémoire du codeur).

TAB. 3.4 – Dégradation de $f_{otm\ (p)}$ en fonction du degré de parallélisme p

Codeur	$\Delta f_{otm\ (8)}\%$	$\Delta f_{otm\ (16)}\%$	$\Delta f_{otm\ (32)}\%$
$H_1(x)/G_1(x)$	22 %	30 %	60%
$H_1(x)/G_2(x)$	2%	34%	49%
$H_2(x)/G_3(x)$	13%	21%	53%
$H_2(x)/G_4(x)$	6%	25%	57%
Moyenne	$\overline{(\Delta f)}_{otm\ (8)}\%$	$\overline{(\Delta f)}_{otm\ (16)}\%$	$\overline{(\Delta f)}_{otm\ (32)}\%$
	11%	27%	54%

Le tableau 3.4 met en évidence la dégradation de la fréquence $\overline{(\Delta f)}_{mto\ (p)}\%$ du codeur OTM en fonction du degré de parallélisme p. En moyenne, on observe une perte de :

– 11% pour une réduction par 8 du nombre de cycles d'horloge ;

– 27% pour une réduction par 16 du nombre de cycles d'horloge ;

– 54% pour une réduction par 32 du nombre de cycles d'horloge ;

Le bilan global pour chaque degré de parallélisme p est évalué, comme pour le codeur MTO, en calculant l'augmentation effective du nombre de bits traités (équ. (3.18))

$$\begin{aligned} \widetilde{\Delta f}_{otm\ (8)} &\approx 0,11 \Rightarrow A_{eff\ (8)} = 0.89 \times 8 = 7,12 \\ \widetilde{\Delta f}_{otm\ (16)} &\approx 0,27 \Rightarrow A_{eff\ (16)} = 0.73 \times 16 = 11,68 \\ \widetilde{\Delta f}_{otm\ (32)} &\approx 0,57 \Rightarrow A_{eff\ (32)} = 0.43 \times 32 = 13,76 \end{aligned} \tag{3.18}$$

En comparant les valeurs (3.17) et (3.18) obtenues respectivement pour les versions CP_{mto} et

CP_{otm}, on voit bien que la version CP_{otm} est plus performante pour tous les degrés de parallélisme choisis $p = 8$, 16, et 32.

En prenant la surface et le débit comme critères de comparaison, les deux architectures parallèle-pipeline MTO et OTM proposées offrent un meilleur compromis vitesse/surface par rapport aux architectures précédentes. On peut remarquer également que l'architecture OTM occupe beaucoup moins de la surface que l'architecture MTO avec un débit légèrement supérieur, pour les deux architectures proposées.

3.6 Applications aux filtres récursifs RII

Le filtrage numérique est très utilisé dans de nombreux domaine d'applications de communication comme les communications mobiles. Il est utilisé pour diverses fonctions telles que la canalisation, l'égalisation de canal, le filtrage adaptatif et la génération d'impulsions [164, 165]. Un filtre est un circuit qui réalise une opération de traitement du signal. Il atténue certaines composantes fréquentielles d'un signal et en laisse passer d'autres (passe-bas, passe-haut, passe bande, réjecteur de bande, intégrateur, différentiateur, ...). Le traitement réalisé porte sur une séquence (ou suite) de nombres introduites à son entrée et fournit une nouvelle séquence numérique "filtrée" à sa sortie.

La demande de filtres numériques à haute vitesse et faible surface s'est largement renforcée avec l'émergence d'applications telles que la radio-logicielle dans les systèmes de communication mobile. Traditionnellement, dans ces applications les signaux de fréquences les plus élevées sont manipulés par des parties analogiques. Cependant, l'utilisation extensive de parties analogiques réduit la flexibilité et l'adaptabilité de ces applications, ce qui est un inconvénient majeur pour les systèmes multistandard actuels et futurs à vocation polyvalente. D'où, une demande croissante de systèmes numériques presque exclusive.

Les filtres numériques nécessaires dans les applications de communication à haute vitesse requièrent généralement des implantations matérielles soit de type RIF (*Réponse Impulsionnelle Finie*), soit RII (*Réponse Impulsionnelle Infinie*).

En effet, leur capacité à sélectionner ou à rejeter des bandes de fréquence avec une forte discrimination (plus de 40 dB) en utilisant des filtres d'ordres faible (inferieurs à 10 en général) le rends beaucoup plus adaptés à la parallélisation que leurs homologues RIF, tant sur le plan de la vitesse que celui de la consommation d'énergie. Cependant, les filtres RII sont plus sensibles aux erreurs de quantification que les filtres RIF. La récursivité permet en effet de générer des erreurs cumulatives. Pour les deux types de filtres (RII et RIF), deux structures possibles existent, la structure directe (SD) étant la plus simple avec un chemin critique linéairement proportionnel à l'ordre du filtre, et la structure transposé (ST) dont le chemin critique est fixe.

Différents travaux ont été menés pour concevoir des filtres rapides à structure parallèle au cours de la dernière décade [166–168]. Concernant les filtres RII, des propositions ont été faites, telles que le filtre numérique haut débit RII 3D de forme-directe pour les technos VLSI/FPGA [169], l'adaptation d'architectures de filtre RII pipeline en utilisant des transformations dispersées et détendue [170], où

la mise en oeuvre d'un pipeline pour la réduction du bruit dans les applications [171]. Concernant les filtres RIF, de nombreuses approches pour augmenter le débits et à réduire la complexité matérielle pouvant être trouvées dans [172–174].

Cependant, bien que ces architectures réussissent à atteindre des débits très élevés, de nombreux inconvénients les caractérisent dont les principaux sont :

– l'augmentation importante de la surface quand les degrés de parallélisme p élevés sont nécessaires, ce qui réduit fortement les bénéfices du parallélisme.
– difficulté de les configurer (changement en ligne des coefficients), car de nombreux pré-calculs sont nécessaires pour déterminer les coefficients réels à mettre en oeuvre (voir [175]).

En réalité, comme nous allons le voir dans la prochaine section, l'architecture d'implantation de filtre SD (ST) et le codeur convolutif récursif MTO (OTM) sont très semblables. Dans ce contexte, les deux structures SD et ST du filtre récursif RII sont vues comme celles de codeurs récursifs convolutif sde type MTO et OTM, respectivement.

Dans la dernière partie de ce chapitre, nous proposons une nouvelle architecture parallèle-pipeline pour les filtres RII [69]. Cette approche s'applique aux deux structures, directe SD et transposé ST du filtre RII. Laquelle SD ou ST est la meilleure structure de filtre en fonction du débit et de la surface consommées en fonction de degré de parallélisme p, c'est l'objet de l'étude qui suit.

3.6.1 Architectures séries d'un filtre RII

Les filtres récursifs RII sont caractérisés par leur équation de récurrence, définie à l'aide de deux jeux de coefficients a et b. Les entrées et sorties à l'instant t respectivement X_t et Y_t, sont maintenant des bus de données de largeur de bits d'entier. De manière générale, le comportement de ces filtres est décrit par une équation aux différences finie, linéaire et à coefficients constants, de la forme :

$$Y_t = \sum_{k=0}^{N} b_k \cdot X_{t-k} - \sum_{k=1}^{N} a_k \cdot Y_{t-k} \tag{3.19}$$

où b_k et a_k sont respectivement les coefficients directs et de retour, N étant l'ordre du filtre. L'équation suivante s'applique également, où D^k représente un retard de k cycles d'échantillonnage.

$$Y = \sum_{k=0}^{N} b_k \cdot X \cdot D^k - \sum_{k=1}^{N} a_k \cdot Y \cdot D^k \tag{3.20}$$

Les deux structures SD et ST sont possibles pour l'implantation de l'équation 3.20. À quelques exceptions rares, les filtres récursifs ont une réponse impulsion infinie. Il est en fait possible de réaliser certains filtres RIF par calcul récursif, mais ceci est généralement plutôt difficile.

L'intérêt des filtres récursifs est :

– leur faible calculatoire (du fait de leur ordre plus faible que leurs plus proches équivalent RIF) ;

– leur faible latence en général (ca dépend de l'architecture) dans beaucoup d'applications de communication.

Les inconvénients des filtres récursifs sont :

– leur phase non linéaire ;

– leur instabilité numérique.

Soit $h(n)$, la réponse impulsionnelle d'un filtre recursif. Pour être réalisable, un filtre récursif doit respecter :

$$h(n) = 0, \quad n < 0, \tag{3.21}$$

et

$$\sum_{n=0}^{\infty} |h(n)| < \infty. \tag{3.22}$$

Les équations 3.21 et 3.22 représentent respectivement la condition de causalité et la stabilité du filtre récursif.

À noter que dans l'implantation, le nombre de multiplications requis (MR) et d'additions (AR) est directement liée au nombre total de coefficients b_k et a_k. Dans les implantations séries, la fréquence d'horloge f_{clk} est égale à la fréquence d'échantillonnage f_s.

Structure directe (SD)

La structure directe (SD) est la structure la plus simple pour l'implantation du filtre numériques [176]. L'architecture série d'un filtre SD du 3ème-ordre est présentée dans la figure 3.20. Cette structure correspond à la transcription directe de l'équation aux différences finies 3.20. Dans cette structure, les "données" retardées par des registres sont multipliés par les coefficients correspondants du filtre et les résultats sommés entre eux pour générer le signal de la bande de retour et la sortie du filtre récursif.

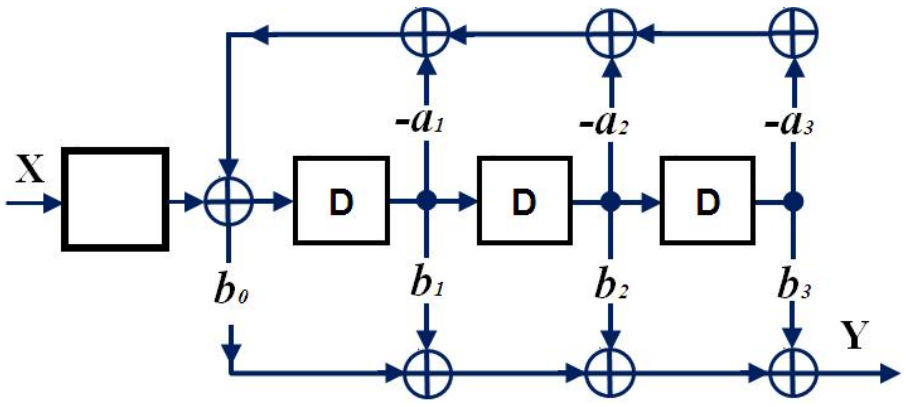


FIG. 3.20 – Structure directe SD du filtre récursif série du 3'ème-ordre

On peut remarquer la similitude entre l'architecture série du filtre SD (figure 3.20) et celle du codeur convolutif récursif MTO (voir section 1.3.3).

Le chemin critique de cette architecture situé dans la structure de calcul est proportionnel à $T_{mult} + (N-1) * T_{add}$ comme le montre la figure 3.20. Ce chemin critique peut être réduit à

$T_{mult} + \lceil log(N-1) \rceil * T_{add}$ en utilisant une structure e calcul en forme d'arbre.

Cette structure directe SD, avec une accumulation linéaire, est très simple à modéliser du fait de sa très grande régularité. Malheureusement, l'accroissement du chemin critique avec l'ordre de filtre fait de cette architecture un mauvais candidat quand une vitesse de fonctionnement élevé est souhaitée.

Structure transposé (ST)

L'application du théorème de transposition à l'équation aux différences 3.20 conduit à la structure de filtre récursif sous la forme transposé ST. Dans cette structure (voir figure 3.21), l'entrée alimente en parallèle tous les multiplieurs et les produits sont cumulés tout au long des périodes d'échantillonnage.

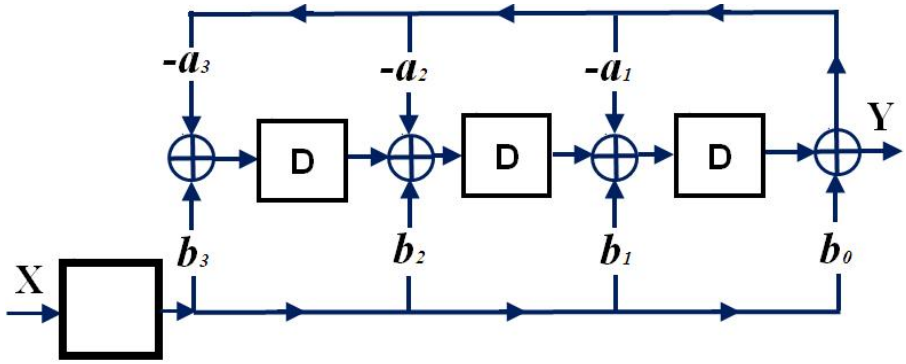


FIG. 3.21 – Structure transposée ST du filtre récursif série du 3ème-ordre

On remarque ici aussi une similitude entre l'architecture ST série (figure 3.20) et l'architecture du codeur convolutif récursif OTM (voir section 1.3.3).

Cette structure conserve la régularité d'accumulation linéaire de la structure directe SD mais avec un chemin critique plus court que celui de la structure SD, limité seulement à un multiplicateur et un additionneur, et donc indépendant de l'ordre du filtre (voir figure 3.21).

L'un des principaux inconvénients de cette structure est du au fait que les registres sont insérées dans le chemin d'accumulation, et par conséquence, un grande charge sur les données d'entrée puisque tous les multiplicateurs sont alimentés en parallèle.

3.6.2 Architectures parallèle-pipeline d'un filtre RII

Par analogie avec les codeurs convolutif parallèle-pipeline MTO et OTM, deux nouvelles architectures parallèle-pipeline adaptés aux filtres SD et ST sont présentés. Aucun précalcul de coefficients n'est nécessaire. Comparativement aux versions existantes, ces architectures offrent de meilleures performances (vitesse) pour une surface consommée largement réduite.

Architecture SD parallèle-pipeline

L'architecture SD parallèle-pipeline du filtre est représenté dans la figure 3.22. Elle est constituée de deux blocs.

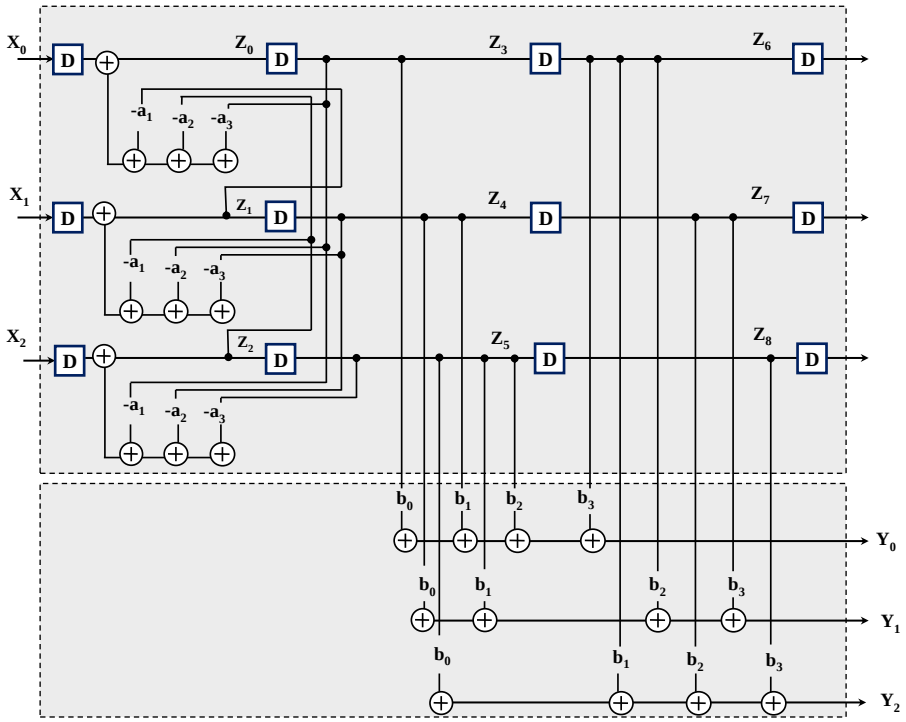


FIG. 3.22 – Filtre SD parallèle-pipeline

La correspondance entre les états des entrées/sorties du filtre SD série et du parallèle-pipeline est donnée respectivement par les équations 3.28 et 3.29.

$$((X_0)_t, \cdots, (X_{p-1})_t) = ((X)_{p\cdot t+(p-1)}, \cdots, (X)_{p\cdot t}) \tag{3.23}$$

$$((Y_0)_t, \cdots, (Y_{p-1})_t) = ((Y)_{p\cdot(t-2)+p}, \cdots, (Y)_{p\cdot(t-2)+1}) \tag{3.24}$$

Le fonctionnement du filtre SD parallèle-pipeline peut être facilement décrit en utilisant la notation Z_i pour les lignes intérieures de l'architecture, conformément à la figure 3.22. Les équations liant les valeurs aux instants $t+1$ et t peuvent être décrites comme suit :

$$\forall i \in \{0, \cdots, p-1\}, \quad (Z_i)_{t+1} = (X_i)_t - \sum_{k=1}^{m} a_k \cdot (Z_{i+k})_{t+1} \tag{3.25}$$

$$\forall i \in \{p, \cdots, p+m-1\}, \quad (Z_i)_{t+1} = (Z_{i-p})_t \tag{3.26}$$

$$\forall i \in \{0, \cdots, p-1\}, \quad (Y_i)_{t+1} = \sum_{k=0}^{m} b_k \cdot (Z_{i+k})_t \tag{3.27}$$

Architecture ST parallèle-pipeline

L'architecture ST parallèle-pipeline, dont le schéma général est représenté dans la figure 3.23, est une architecture parallèle-pipeline rapide et nécessitant une surface réduite.

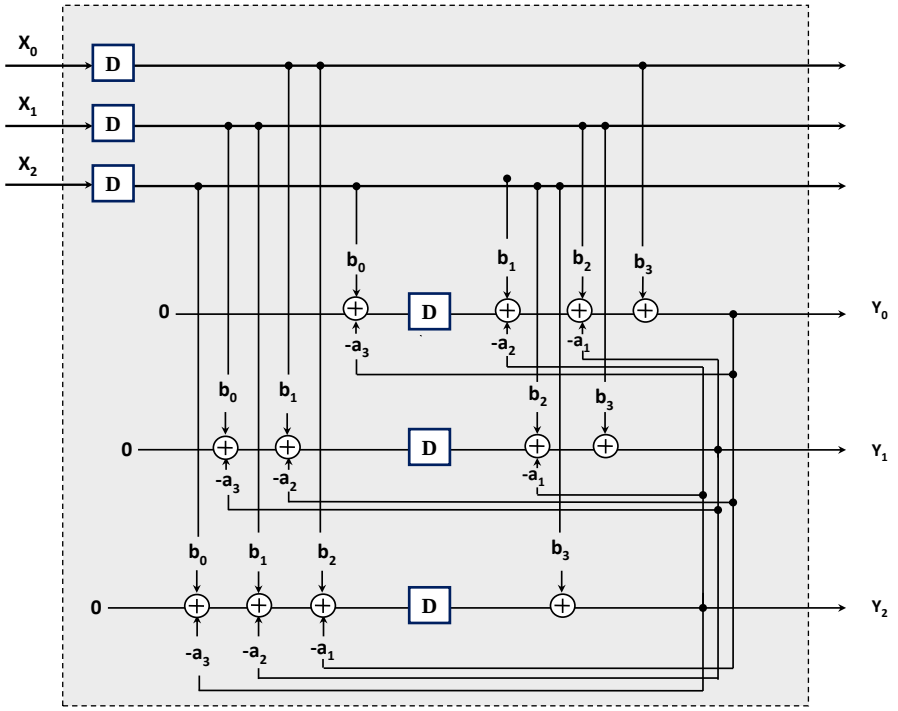


FIG. 3.23 – Filtre ST parallèle-pipeline

Comme le montre la figure 3.23, l'architecture est constituée d'un seul bloc. La structure de ce bloc génère les p sorties en parallèle en combinant les différentes valeurs retardés (dans le pipeline) des entrées X_i et des sorties Y_i et utilisant les coefficients a_k et b_k, respectivement.

Les entrées/sorties des architectures ST série et parallèle-pipeline sont données dans les équations 3.28 et 3.29.

$$((X_0)_t, \cdots, (X_{p-1})_t) = ((X)_{p\cdot t+(p-1)}, \cdots, (X)_{p\cdot t}) \tag{3.28}$$

$$((Y_0)_t, \cdots, (Y_{p-1})_t) = ((Y)_{p\cdot(t-1)+p}, \cdots, (Y)_{p\cdot(t-1)+1}) \tag{3.29}$$

Le fonctionnement du filtre ST parallèle-pipeline peut être facilement décrit par l'équation suivante reliant les valeurs d'entrée X_i et les valeurs de sortie Y_i :

$$\forall i \in \{0, \cdots, p-1\},$$

$$\begin{aligned}(Y_i)_{t+1} &= \sum_{k=0}^{m} b_k \cdot (X((i+k) \mod p))_{t-\lfloor (i+k)/p \rfloor} \\ &- \sum_{k=0}^{m} a_k \cdot (Y((i+p-k) \mod p))_{t+1-\lfloor (i+p-k)/p \rfloor}\end{aligned} \tag{3.30}$$

Comparaison entre les deux architectures

Le choix entre les versions parallèles-pipeline SD ou ST dépend du compromis vitesse/surface. Pour les deux versions, le nombre total d'additionneurs et de multiplieurs nécessaires dépend du nombre de coefficients a_k et b_k non nuls. Lorsque les filtres sont paramétrables en ligne (pour le filtrage adaptatif, par exemple), les coefficients doivent être considérées comme différent de zéro car, n'étant pas connus à la synthèse, ils doivent être stockés dans des registres.

Afin d'évaluer la qualité des différentes filtres SD et ST parallèle-pipeline, il est utile d'estimer leur chemin critique et leur surface en fonction des paramètres d'implantation (ordre du filtre N, parallélisme p, etc...). Le tableau 3.5 résume les caractéristiques théoriques des filtres concernant le chemin critique et la surface (uniquement concernant les additionneurs et les multiplieurs). Il est à noter que, en général, T_{mult} et A_{mult} ont tendance à être plus grands que leurs homologues T_{add} et A_{add}, $A_{add}(A_{mult)}$ et $T_{add}(T_{mult)}$ étant respectivement les surfaces et le temps de propagation d'un additionneur et d'un multiplicateur.

TAB. 3.5 – Comparaisons entre les architectures SD et ST parallèle-pipeline

Filter RII	Chemin critique	Surface active
Structure Directe (SD)	$T_{mult} + N \cdot T_{add}$	$2 \cdot (N+1) \cdot p \cdot A_{add} + (2N+1) \cdot p \cdot A_{mult}$
Structure Transposée (ST)	$T_{mult} + \min(N,p) \cdot T_{add}$	$(2N+1) \cdot p \cdot (A_{add} + A_{mult})$

On peut remarquer que, le chemin critique du filtre SD ne dépend que de l'ordre N du filtre RII et non du degré du parallélisme p. Dans le cas du filtre ST, le chemin critique est lié au minimum de p et N. Par conséquent, le meilleur choix entre les filtres SD et ST concernant le chemin critique dépend de la valeur relative de p et N. Selon les critères de débit, ST est le meilleur choix lorsque $p < N$.

Concernant la consommation en surface, le filtre basé SD utilise plus d'additionneurs (p en fait) que le filtre basé ST. Bien entendu, la surface additionnelle requis pour les registres et le câblage dans les implémentations finales et peut modifier légèrement ces estimations.

3.6.3 Résultats expérimentaux

Dans cette section, nous étudions la surface consommée et la performance de filtres SD et ST parallèle-pipeline implantés sur des FPGA de type Stratix II d'Altera en utilisant l'outil Quartus II (version 7.1). La fréquence maximale de fonctionnement f_{max} de cette famille de FPGA est limité à 500 MHz. Plusieurs configurations SD et ST parallèle-pipeline ont été générées correspondant à différents degrés de parallélisme p et de largeur de mot w.

Chaque circuit a été analysé concernant la surface occupée et la fréquence d'échantillonnage maximale f_s, avec $f_s = p \cdot f_{clk}$. À noter, le débit de bit correspondant est égal à $w \cdot f_s$.

Les résultats sont rassemblés dans le tableau 3.6, où la fréquence d'échantillonnage f_s est exprimée en GHz et la consommation en surface est mesurée en nombre de cellules logiques CLBs (FPGA) utilisées. Par rapport à l'architecture SD, l'architecture ST permet une grande économie de matériel.

TAB. 3.6 – Comparaison des résultats expérimentaux pour les filtres SD et ST

w	p	Consommation en surface (LB)		$\frac{ST-SD}{SD}$	Fréquence d'échantillonnage f_s (GHz)		$\frac{ST-SD}{SD}$
		filtre SD	filtre ST		filtre SD	filtre ST	
4	4	203	179	$-2,95\%$	$0,31$	$0,51$	$+64,5\%$
	8	400	276	$-31,0\%$	0.59	0.87	$+47,5\%$
	12	611	410	$-32,9\%$	$0,86$	$1,17$	$+36,0\%$
	16	811	540	$-33,4\%$	$1,11$	$1,68$	$+51,4\%$
	20	1011	674	$-33,3\%$	$1,29$	$2,01$	$+55,8\%$
	24	1213	803	$-33,8\%$	$1,69$	$2,49$	$+47,3\%$
8	4	287	225	$-21,6\%$	$0,31$	$0,46$	$+48,4\%$
	8	493	349	$-29,2\%$	$0,51$	$0,82$	$+60,8\%$
	12	705	481	$-31,8\%$	0.56	$1,25$	$+123,2\%$
	16	914	613	$-32,9\%$	$0,61$	$1,58$	$+159,0\%$
	20	1134	745	$-34,3\%$	$0,79$	$2,00$	$+153,2\%$
	24	1320	877	$-33,6\%$	$0,95$	$2,13$	$+124,2\%$

Il est claire aussi que la quantité du matériel consommé dépend directement de degré de parallélisme p. D'après le tableau 3.6, nous pouvons voir que le gain en surface du filtre ST varie entre 11,8% et 33,8% pour une largeur de mot $w = 4$, et entre 21,6% et 34,3% pour $w = 8$, la moyenne étant d'environ 30%. Il est claire que la surface consommée du filtre ST est faible par rapport au filtre SD.

En termes de fréquence d'échantillonnage maximale f_s, l'architecture ST offre des meilleures performances que celles du filtre SD, l'avantage allant jusqu'à 159%un $w = 8$. À noter que la fréquence f_s dépend aussi du parallélisme p.

D'après ces résultats, il est donc clair que le compromis débit/surface est largement favorable à l'architecture ST. Il est à noter que les résultats sont en bonne harmonie avec ceux prévus dans le tableau 3.5.

3.7 Conclusion

Dans ce chapitre, nous avons présenté une architecture rapide pour les codeurs convolutifs récursifs et son adaptation directe aux filtre numériques récursifs RII.

Concernant les codeurs convolutifs, deux architectures parallèles-pipeline correspondant aux structures MTO et OTM ont été testées pour différents degrés de parallélisme et différents polynômes générateurs. Tous les modèles RTL ont été décrits en VHDL. Ils sont indépendants de la technologie cible et configurables avant synthèse. Le débit atteint avec les implantations sur FPGA de type Stratix II est $6,69$ Gbits/s ($8,10$ Gbits/s), dans le cas du codeur CP_{mto} (respectivement CP_{otm}) pour un degré

de parallélisme $p = 32$. La version CP_{mto} est moins performantes que la CP_{otm}. Du point de vue de la surface consommée, l'architecture CP_{otm} est moins gourmande que la CP_{mto}. La version CP_{otm} parallèle-pipeline présente donc un meilleur compromis débit/surface.

Ces architectures de codeurs ont été généralisées aux filtres RII. Plusieurs configurations parallèles-pipeline de filtre récursif direct SD et transposé ST ont été testées pour différents degrés de parallélisme p et de largeur de mot w. Les résultats obtenus après implantation sur FPGA montrent que la version ST parallèle-pipeline du filtre offre le meilleur compromis débit/surface. L'avantage sur la version ST allant jusqu'à 159% de vitesse en plus et 32,9% de surface en moins pour un $w = 8$.

Chapitre 4

Méthodologie de conception de codeurs en bloc SdF

4.1 Introduction

Le développement des technologies microélectroniques entraîne une utilisation de plus en plus importante des circuits intégrés dans les systèmes numériques. La SdF est devenu un aspect très intéressant dans la conception des circuits intégrés. Elle est de plus en plus nécessaire dans différents domaines d'application, notamment pour les systèmes de transmission numérique dont les codeurs et décodeurs sont considérés comme des éléments essentiels.

Dans ce chapitre, nous introduisons une méthodologie basée principalement sur le développement de codeurs sûrs en présence de fautes FS (*Fault-Secure*). La surface consommée par les codeurs FS est évaluée en comparaison à leur homologues simples (S) (utilisés selon l'approche classique de duplication et comparaison (voir section 2.7.1)). Pour évaluer les surfaces consommées et les débits des codeurs S et FS, des modèles génériques décrits en VHDL au niveau d'abstraction RTL ont été synthétisés sur des FPGA detype Stratix II d'Altera . Différentes campagnes d'injection de fautes simples (SEU) et multiples (MBU) ont été mises en place pour valider et montrer l'intérêt du concept.

Nous détaillons ensuite pour la partie décodeur, l'étude de la technique de protection en vue de la FS. Une architecture FS de décodeur cyclique parallèle-pipeline est présentée. Finalement, une comparaison suivie d'une discussion sur les résultats (débit, surface) obtenus par implantations sur FPGA de la famille de Stratix II est proposée pour demontrer l'intérêt de ces décodeurs FS.

4.2 Introduction de la méthodologie

Notre objectif est de proposer et de valider une méthodologie de conception d'architectures de codeurs sûrs en présence de fautes pour les codes en blocs. Afin d'y parvenir, la méthode que nous avons adoptée consiste à étudier le comportement fonctionnel du codeur, indépendamment de la structure du bloc principal du codeur. La complexité et la performance du codeur FS sont évaluées par implan-

tation sur FPGA. L'évaluation de la sûreté est obtenue par différentes campagnes d'injection de fautes au niveau RTL. L'étude fonctionnelle, qui précède l'étude architecturale, est réalisée mathématiquement.

Avec la réduction continue d'échelle des technologies microélectroniques qui conduit à des fonctionnalités utilisant des blocs de taille réduite, des tensions d'alimentation basses et des débits de fonctionnement élevés, la sensibilité des circuits intégrés aux perturbations externes (impact de particules, perte de l'intégrité du signal, etc.) augmente. Cela implique des dysfonctionnements plus fréquents, et par conséquence, la SdF pour ces circuits n'est pas garantie. À l'heure actuelle, le taux d'erreurs dans les circuits logiques est en augmentation rapide [2,5,178,179]. La tolérance aux fautes devient donc un critère important pour améliorer la fiabilité et la qualité de service des circuits intégrés. Aujourd'hui, les principales sources d'erreurs dans les systèmes de mémoire RAM sont les erreurs temporaires, appelées SEU et causées par le rayonnement cosmique [5]. Différents codes EDC et ECC ont été utilisés pendant des années pour accroître la fiabilité des systèmes de mémoire RAM [180–189] et des systèmes de transmission [6, 190–192]. Évidemment, la classe de codes utilisée dépend de l'organisation de données, des exigences de performance et de la nature des erreurs potentielles.

Le besoin de mémoires RAM très rapides pour le traitement et le stockage de données est de plus en plus important. Ce type de RAM est conçu pour opérer avec des blocs tels que des microprocesseurs, des codeurs, ou des décodeurs. Ces blocs doivent eux mêmes présenter des niveaux de performance et de fiabilité élevés. Dans notre cas, on cherche à concevoir une version FS d'un codeur placé entre une RAM et une chaîne de transmission, ce qui nécessite de pouvoir effectuer la lecture/écriture parallèle des mots de mémoire RAM. Ces mots sont protégés par des bits de contrôle propres. Les perturbations affectant un canal de transmission sont souvent la cause de plusieurs bits d'erreurs (*burst*), de sorte que les modèles d'erreurs supposés affecter le canal de transmission d'une part et la mémoire RAM d'autre part sont différents. Autrement dit, les caractéristiques de détection et/ou de correction requises pour les codes utilisés dans les transmissions de données sont généralement différentes de celles requises pour les codes employés à la protection d'une mémoire RAM.

Comme nous l'avons rappelé dans l'état de l'art de ce mémoire, les codes SEC/DED peuvent être utilisés pour rendre les systèmes de mémoire RAM tolérants aux fautes [180–184] : citons par exemple les codes de Hamming [180] et les codes de Hamming modifiés par Hsiao [181]. D'autres approches peuvent être trouvées dans [182–189]. Il est important aussi de rappeler qu'un système de mémoire RAM tolérante aux fautes est typiquement composé de deux parties (voir Fig 4.1) : (i) les cellules mémoire stockant les données protégées par l'EDC ou ECC et (ii) le circuit de détection et de correction EDAC (*Error Detection And Correction*) qui vérifie l'exactitude des données lues à partir de la RAM et corrige les erreurs détectés, si nécessaire. Ensuite, les bits de contrôle de mémoire sont tout simplement ignorés et seule la partie «donnée» proprement dite est envoyée vers l'entrée du codeur. Concernant le système de transmission, les codes cycliques CRC, BCH, et RS sont largement utilisés, car ils ont de bonnes propriétés. Un certain nombre d'entre eux ont été standardisés pour les normes de transmission (ils seront détaillés plus tard).

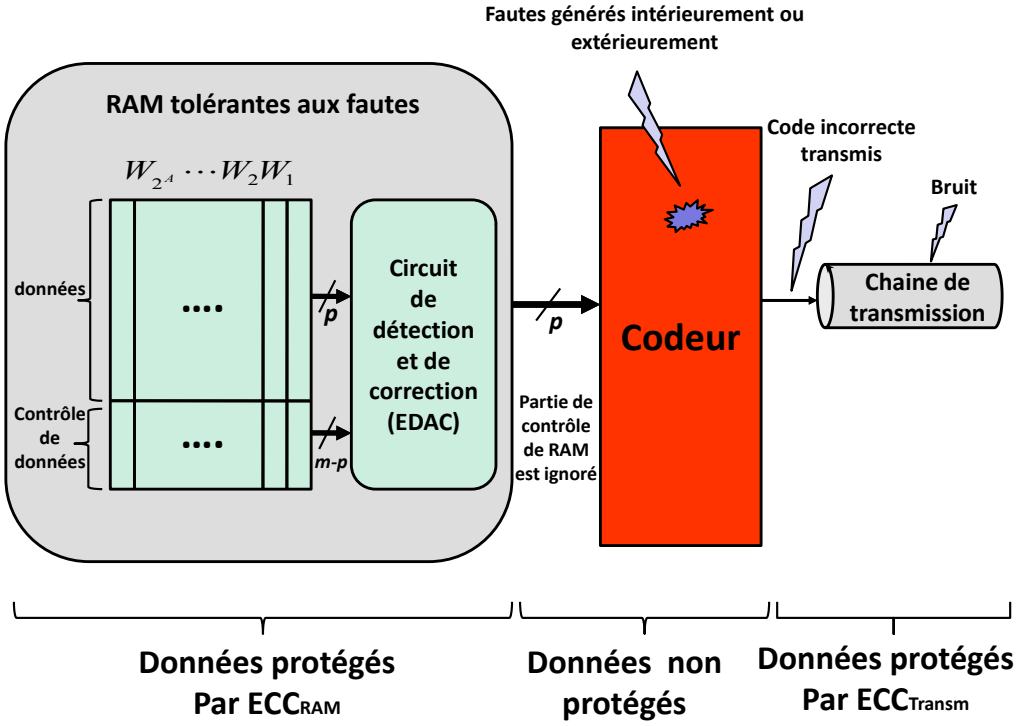


FIG. 4.1 – Codeur non protégé entre une RAM tolérante aux fautes et un chaine de transmission

Les codeurs et décodeurs sont des circuits critiques dans une chaîne de transmission. En fait, certaines erreurs pouvant être introduites par des (dé)codeurs défectueux peuvent passer inaperçues, sauf si ces circuits sont fournis avec des moyens simultanée de détection des erreurs (CED – *Concurrent Error Detection*). En effet, un codeur défectueux peut introduire des erreurs dans les données à corriger provenant de la RAM, de telle sorte que des données erronées avec des bits de contrôle correspondants corrects puissent être transmises dans le canal de transmission, sans possibilité ultérieure de détection ou correction d'erreurs. Des erreurs peuvent également se produire sur le bus non protégé situé entre le circuit EDAC de la RAM et le codeur. Ces erreurs ne peuvent évidemment pas être détectées si le codeur ne dispose pas de moyens de contrôle des données. Pour être correctement protégé contre toutes ces erreurs, le codeur doit être implanté comme FS (c'est-à-dire, que la première occurence d'une erreur doit être détectée) [115]. De même, les fautes internes dans un décodeur non protégé peuvent entraîner des données erronées au niveau du récepteur, malgré l'absence d'erreurs dans la transmission.

À ce jour, plusieurs auteurs ont abordé des aspects d'auto-contrôlabilité : des codeurs auto-contrôlables ont été proposés pour les codes CRC dans [193], ainsi que des codeurs et décodeurs auto-contrôlables pour les codes BCH et RS dans [194–197]. Récemment, des codeurs et décodeurs FS ont été proposés dans [198] pour protéger une RAM tolérante aux fautes par des codes de parité à faible densité. Aucun cependant n'a pris en considération la protection de codeurs placés entre la mémoire tolérante aux fautes et le canal de transmission. Le principe proposé dans cette thèse consiste à prendre en considération les informations de redondance disponibles dans la RAM (voir figure 4.3)

pour améliorer la tolérance du codeur.

Plusieurs raisons motivent la conception de codeurs et décodeurs FS. La miniaturisation des technologies des circuits intégrés VLSI (*Very Large Scale Integration*) augmente la sensibilité des architectures aux fautes transitoires causées par les radiations cosmiques, non seulement au niveau de la mémoire, mais aussi de la logique combinatoire. Par conséquent, ce type d'erreurs se produit dans les systèmes électroniques non seulement dans les applications spatiales et avioniques, mais également au niveau du sol. Hors, un nombre croissant d'applications de transmission à haute performance nécessite une fiabilité élevée pour pouvoir gérer des situations critiques dans des environnements défavorables succeptibles d'introduire des fautes transitoires. Les codeurs et décodeurs classiques non protégés contre ce type de fautes ne sont donc pas appropriés.

Quelques systèmes représentatifs peuvent être trouvés dans [192]. Enfin, il est à noter que les systèmes électroniques implantés avec des nanotechnologies devraient présenter des taux de fautes transitoires encore plus élevés [198]. Par conséquent, chaque partie d'une mémoire et d'un système de transmission devient critique dans la conception d'un système fiable tolérant aux fautes.

Le but de ce chapitre est de concevoir un codeur FS situé entre une mémoire RAM protégée et un canal de transmission. La conception d'un tel codeur repose principalement sur les aspects suivants :

- la prise en compte de la disponibilité des bits de contrôle des codes cycliques EDC ou ECC protégeant la RAM, pour s'assurer de la validité des données provenant de la RAM ;
- la prise en compte des bits de contrôle de la transmission, calculés par le codeur principal, pour générer une référence de comparaison avec les bits de contrôle de la RAM. Le but est de détecter les erreurs induites dans le codeur pendant son fonctionnement, et par conséquent, de le rendre tolérant aux fautes.

L'approche proposée pour concevoir un codeur tolérant aux fautes remplace avantageusement la méthode classique de duplication avec comparaison qui requiet plus de 100% de surcoût matériel (voir section 2.7.1). Dans la plupart des cas, le surcoût matériel du codeur FS proposé ici est inferieur à 50%. Avec cette nouvelle approche, la complexité est réduite, et par conséquent la consommation d'énergie et les performances temporelles restent acceptables. Dans notre étude, on souhaite travailler au niveau RTL, comme cela est le cas pour différentes travaux de conception et synthèse de circuits auto-contrôlables [108, 200–204]. Les concepts utilisés ne peuvent donc être basés que sur des propriétés vérifiables au niveau RTL.

La suite de ce chapitre est organisé en 5 sections. La section 4.3 introduit une méthodologie générale pour concevoir un codeur FS. La section 4.4 présente le contexte théorique conduisant à des implantations parallèles de codeurs FS et les possibilités de les simplifier. Dans la section 4.5, la complexité et la performance (débit) de l'architecture FS sont évaluées, l'efficacité de détection du codeur FS étant étudiée au moyen de la technique d'injection de fautes. Ensuite, dans la section 4.6, le codeurs FS parallèle-pipeline est implanté pour différentes configurations. À la fin du chapitre, une architecture FS de décodeur cyclique en bloc est proposée. Afin d'évaluer la complexité et la performance de cette architecture, différents décodeurs FS sont implantés correspondant à différents codes, et une discussion des résultats obtenus est présentée.

4.3 Élaboration de la méthodologie

L'objectif de ce travail est d'assurer la fiabilité de fonctionnement du codeur en présence de fautes temporaires, telles que les fautes SEU causés par le rayonnement cosmique [2, 5, 178, 179]. L'architecture du contrôleur doit détecter immédiatement les erreurs produites par les fautes SEU. Une fois l'erreur détectée, le processus de codage des données D doit être interrompu puis répété. Si une faute ne produit pas d'erreur, le processus de codage peut être poursuivi, comme si la faute n'avait pas eu lieu. Rappelons qu'une classe de circuits dits auto-contrôlables et ayant ce comportement a été formellement définie comme suit (voir section 2.8.1) :

soit F désignant l'ensemble de fautes susceptibles d'affecter un circuit H. Nous supposons ici que F contient tous les fautes affectant une seule ligne, à l'exclusion toutefois des lignes d'horloge. Un circuit H est dite sûr en présence de fautes (FS) pour un ensemble de fautes F, si pour tout $f \in F$, il ne produit jamais un mot code incorrect en sortie en présence d'un mot code à l'entrée, c'est-à-dire :

- soit la sortie est correcte (faute masqué),
- soit la sortie est un mot non code (erreur détectable).

Cette propriété garantit que, pour l'ensemble de fautes F, le circuit H ne génère pas de sortie erronée appartenant au code, et donc non détectable par le contrôleur.

Rappelons que l'algorithme de codage systématique utilisé pour les codes cycliques en blocs est un codage par division (section 1.3.2) consistant à coder le message $D(x)$ à l'aide du polynôme générateur $G(x)$ de la manière suivante (équation 1.12) :

$$\begin{aligned} C(x) &= x^{n-k}D(x) + R(x) \\ R(x) &= [x^{n-k}D(x)] \mod G(x) \end{aligned} \tag{4.1}$$

Le mot code $C(x)$ est dit sous forme systématique et les composantes de $R(x)$ constituent les symboles de redondance (ou de parité).

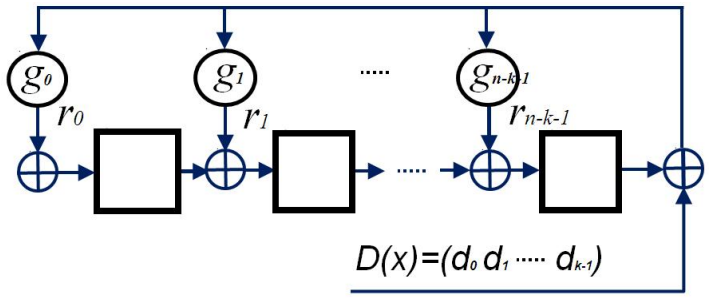


FIG. 4.2 – Schéma général de l'implantions série de la division du polynôme par $G(x)$

La division polynômiale série peut être réalisée par un LFSR (*Linear Feedback Shift Register*), dont l'architecture est schématisée dans la figure 4.2, les données $D(x)$ étant pré-multipliées par x^{n-k} (section 1.3.2) avant d'entrer dans le LFSR (conséquence du choix du point d'injection dans le LFSR).

4.3.1 Schéma général d'un codeur FS

Le circuit proposé, dont le schéma général est indiqué dans la figure 4.3, est un codeur FS pour codes cycliques linéaires EDC ou ECC utilisés en transmission de données (tels que les codes de parité, CRC, BCH et RS). Nous allons tout d'abord décrire ses principes de fonctionnement, puis nous allons montrer qu'il possède le comportement FS revendiqué. Il est à noter que ce schéma ne s'applique pas seulement à un ECC, mais aussi à un EDC (tel que le code de parité simple). Nous utilisons cependant dans la suite la notation ECC ($\mathrm{ECC_{RAM}}$) pour les deux cas, la notation EDC étant réservée spécifiquement pour les codes utilisés dans la construction du codeur FS et dont le rôle sera expliqué par la suite.

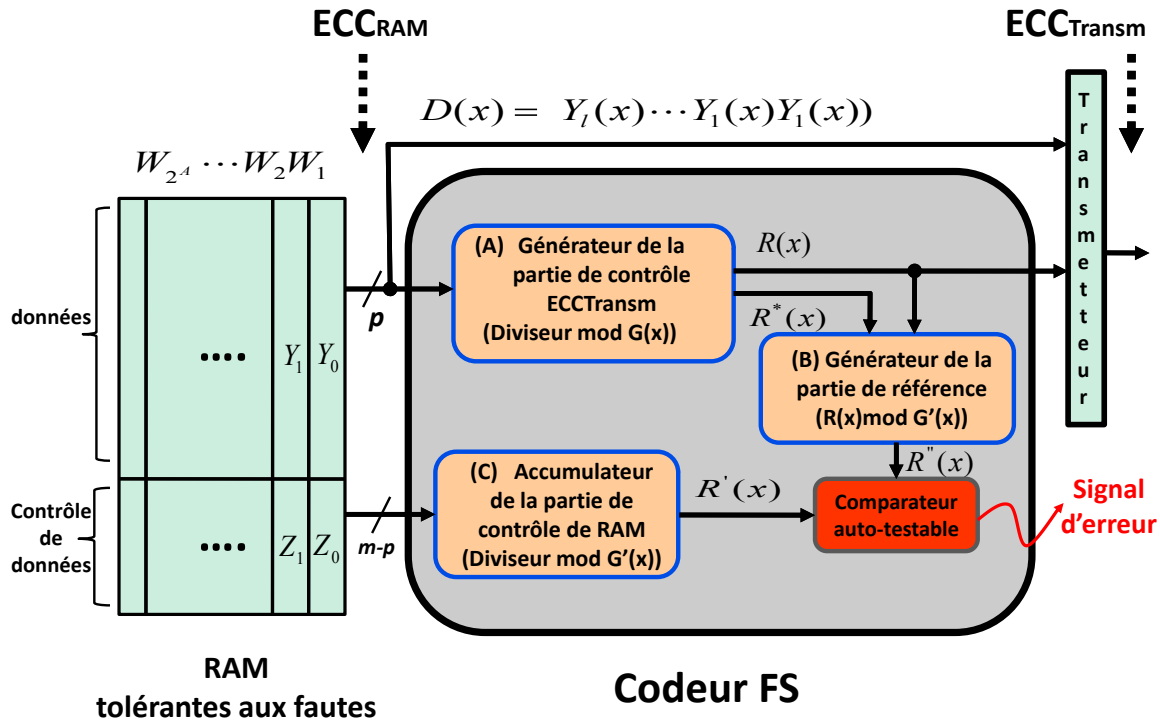


FIG. 4.3 – Schéma d'un codeur FS (*Fault-Secure*) proposé

Dans un système de RAM tolérant aux fautes utilisant un code systématique $\mathrm{ECC_{RAM}}$, un mot code W_i de m-bit est la concaténation de $p$bits de données Y_i et de $(m-p)$bits de contrôle Z_i, soit $W_i = (Y_i Z_i)$. Les codes $\mathrm{ECC_{RAM}}$ employés peuvent être des codes EDC tels que le code de parité (qui permet de détecter toutes les erreurs de multiplicité impaire) ou des codes Hamming SEC ou SEC/DED. Tous ces codes sont des codes linéaires pouvant être mis sous forme cyclique [6]. Puisque nous avons trouvé que la propriété cyclique du code est essentielle pour implanter un codeur FS parallèle, nous avons donc supposé que les codes $\mathrm{ECC_{RAM}}$ sont des codes cycliques générés par des polynômes générateurs $G'(x)$ de degré $(m-p)$. Par conséquence, les équations suivantes

s'appliquent :

$$\begin{aligned} W_i(x) &= Y_i(x) \cdot X^{m-p} + Z_i(x), \\ Z_i(x) &= (Y_i(x) \cdot X^{m-p}) \bmod G'(x). \end{aligned} \tag{4.2}$$

Les données $D(x)$ à transmettre sont lues depuis la RAM tolérante aux fautes afin de former le bloc de $l \cdot p$bits de mots code $Y_i(x)$ $(1 \leq i \leq l)$, la longueur du bloc de données à transmettre étant $k = l \cdot p$. Dans un codeur non protégé, les bits de contrôle $Z_i(x)$ sont éliminés avant le codage et la transmission. Dans la notation polynômiale, $D(x)$ peut être écrit sous la forme :

$$D(x) = \sum_{i=1}^{l} Y_i(x) \cdot x^{(l-i)p}, \quad k = l \cdot p, \tag{4.3}$$

où

$$Y_i(x) = d_{k-ip-p} + d_{k-ip-p+1}x + \cdots + d_{k-ip-1}x^{p-1}, \quad d_i = 0, \quad i \notin \{1, \cdots, l\}. \tag{4.4}$$

Les données à transmettre sont protégées contre les erreurs de transmission en utilisant des codes EDC ou ECC efficaces capables de détecter et/ou corriger plusieurs erreurs. Ils seront ici notées $\mathrm{ECC}_{\mathrm{Transm}}$ et supposés être des codes cycliques linéaires. Le mot code $C(x)$ transmis (voir figure 4.4) résulte de la concaténation de k bits de données $D(x)$ et de $(n-k)$ bits de contrôle $R(x)$, c'est-à-dire, $C(x) = D(x) \cdot x^{n-k} + R(x)$. Les bits de contrôle $R(x)$ sont calculés par le circuit générateur de la partie de contrôle de $\mathrm{ECC}_{\mathrm{Transm}}$ (qui n'est rien d'autre que le diviseur "mod $G(x)$"). Il s'agit en fait du seul bloc commun entre un codeur FS et un codeur non protégé (le codeur FS comporte des blocs supplémentaires calculant $R'(x)$ et $R''(x)$ et nécessaires au fonctionnement FS).

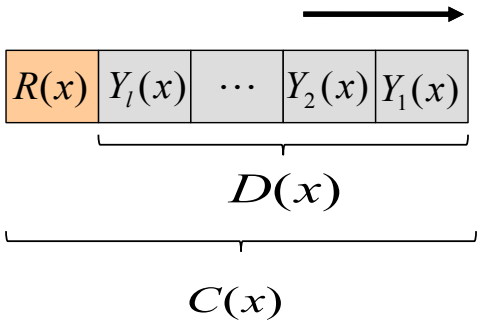


FIG. 4.4 – Structure d'un bloc de transmission de données codées

La version FS du codeur comporte trois blocs supplémentaires : (i) l'accumulateur de bits de contrôle de la RAM, qui génère les bits de contrôle $R'(x)$ calculés à partir de l mots consécutifs Z_i, et n'est rien d'autre que le diviseur "mod $G'(x)$" ; (ii) le générateur de bits de contrôle de référence $R''(x)$ qui n'est rien d'autre que le diviseur $[R(x) + R^*(x)] \bmod G'(x)$, et (iii) le comparateur auto-testable [184] capable de détecter toute divergence (et donc erreur) entre $R'(x)$ et $R''(x)$ à chaque cycle d'horloge.

La propriétée FS de l'ensemble du circuit peut être facilement prouvée en se basant sur les deux chemins différents indépendants, allant de la RAM tolérante aux fautes au comparateur auto-testable. Supposons qu'il y ait une et une seule ligne fautive dans l'un des quatre blocs du codeur FS provoquant une erreur à un certain cycle d'horloge. Si la faute (ou n'importe qu'elle erreur causée par celle-ci) se produit dans le chemin de la partie de données (provenant de la partie données de la RAM, via le générateur étendu et le générateur de bits de contrôle des références $R''(x)$), cela conduit à $R''^e(x) \neq R''(x)$. Par conséquent, le comparateur signale un désaccord entre $R''^e(x)$ et le signal correct produit par la partie de contrôle $R'(x)$ (provenant de la partie de contrôle de RAM via l'accumulation de bits de contrôle de RAM). Le même raisonnement s'applique si une faute produit dans $R'(x)$: cela conduit de nouveau à $R'^e(x) \neq R'(x)$ et à un désaccord $R'^e(x) \neq R''(x)$ signalé par le comparateur. Bien évidemment, toutes les fautes simples dans le comparateur sont détectées, car celui-ci est auto-testable.

Comme nous l'avons expliqué, notre but est de concevoir un codeur FS. Ce codeur est connecté d'une part à la RAM où les données à traiter sont stockées, et d'autre part au codeur où la redondance est calculée pour être transmise avec les données. À chaque cycle d'horloge $p\text{bits}$ correspand à la largeur du mot de données Y_i dans la RAM sont lus parallèle par le codeur, ce qui implique que le codeur FS doit être capable de traiter ces $p\text{bits}$ en parallèle. Dans la suite nous décrivons le fonctionnement de ce codeur FS parallèle.

4.4 Implantations de codeurs FS parallèles

Dans cette section nous allons examiner les différentes équations régissant le fonctionnement parallèle du codeur FS. Ce codeur FS parallèle permet non seulement d'obtenir un débit élevé, mais également une protection des données en son sein.

En le cas de manifestation d'une erreur, un comparateur auto-testable est utilisé pour détecter le disfonctionnement du codeur. Rappelons que pour un code systématique, le message original $D(x)$ est directement inclut dans le mot code $C(x)$ (section 1.3.2)

$$\begin{aligned} C(x) &= c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \\ &= D(x) \cdot x^{n-k} + R(x), \end{aligned} \tag{4.5}$$

le polynôme $R(x)$ étant défini comme

$$\begin{aligned} R(x) &= r_0 + r_1 x + \cdots + r_{n-k-1} x^{n-k-1} \\ &= (D(x) \cdot x^{n-k}) \bmod G(x). \end{aligned} \tag{4.6}$$

L'opération de codage ci-dessus requiert le calcul de la division polynômiale par $G(x)$. Puisque le nombre k de bits de données à coder est relativement important, le codage direct via un circuit purement combinatoire aurait un coût très élevé. Une implantation série à l'aide d'un LFSR est géné-

ralement utilisée à la place.

Concernant $R(x)$, plusieurs cycles d'horloge sont nécessaires pour le générer avant de pouvoir le concaténer à $D(x)$ et former ainsi $C(x)$. En général, une faute survenant au cours du codage peut causer des erreurs simples (SEU) ou multiples (MBU) qui pourraient ne pas être détectées.

Dans cette section, nous allons proposer une procédure de codage permettant non seulement une implémentation parallèle, mais rendant aussi possible la conception d'un circuit FS capable de contrôler son propre fonctionnement à chaque cycle d'horloge.

4.4.1 Algorithme de codage parallèle

Nous allons maintenant introduire les différentes formules mathématiques sous-jacente à la conception d'un codeur FS parallèle, selon le principe de base de la figure 4.3 (voir [70]). Ces équations génèrent les différents polynômes $R(x)$, $R'(x)$ et $R^*(x)$ nécessaires, de degrés respectifs p, $(m-p)$ et $(m-p)$ [1].

L'équation pour calculer $R(x)$ est définie à partir de :

$$R_i(x) = \left(\sum_{u=0}^{i-1} Y_u(x) \cdot X^{(i-1-u)p+n-k} \right) \bmod G(x), \tag{4.7}$$

d'où l'équation de récurrence :

$$\begin{aligned} R_0(x) &= 0 \\ R_{i+1}(x) &= (R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}) \bmod G(x). \end{aligned} \tag{4.8}$$

Rappelons que $r_{i,j}$ est le coefficient de x^j dans $R_i(x)$. Par exemple, pour $p = 1$, l'algorithme série correspondant peut être formellement décrit comme suit :

$$\begin{aligned} R_0(x) &= 0 \\ R_{i+1}(x) &= (R_i(x) \cdot x + d_{k-i-1} \cdot x^{n-k}) \bmod G(x) \\ &= R_i(x) \cdot x + r_{i,n-k-1} G(x) + d_{k-i-1}(x^{n-k} + G(x)). \end{aligned} \tag{4.9}$$

Notons que le polynôme $R(x)$ n'est rien d'autre que $R_l(x)$, où l est le nombre de mots de RAM à transmettre.

Par un raisonnement analogue à 4.6, 4.7 et 4.8, nous pouvons définir la relation de récurrence suivante pour calculer le polynôme $R'(x)$. La partie contrôle $R'(x)$ a générer par le polynôme $G'(x)$ correspond à :

$$R'(x) = (D(x) \cdot x^{m-p}) \bmod G'(x). \tag{4.10}$$

[1] Les notations introduites ci-dessous seront utilisées a travers ce chapitre

Par conséquent :

$$R'_i(x) = \left(\sum_{u=0}^{i-1} Y_u(x) \cdot x^{(i-1-u)p+m-p} \right) \bmod G'(x), \tag{4.11}$$

d'où la récurrence :

$$\begin{aligned} R'_0(x) &= 0 \\ R'_{i+1}(x) &= [R'_i(x) \cdot x^p + Y_i(x) \cdot x^{m-p}] \bmod G'(x) \\ &= [R'_i(x) \cdot x^p] \bmod G'(x) + [Y_i(x) \cdot x^{m-p}] \bmod G'(x) \\ &= [R'_i(x) \cdot x^p] \bmod G'(x) + Z_i(x). \end{aligned} \tag{4.12}$$

Rappelons que notre objectif est de définir une équation d'identité $R'(x) = R''(x)$ qui permettre de

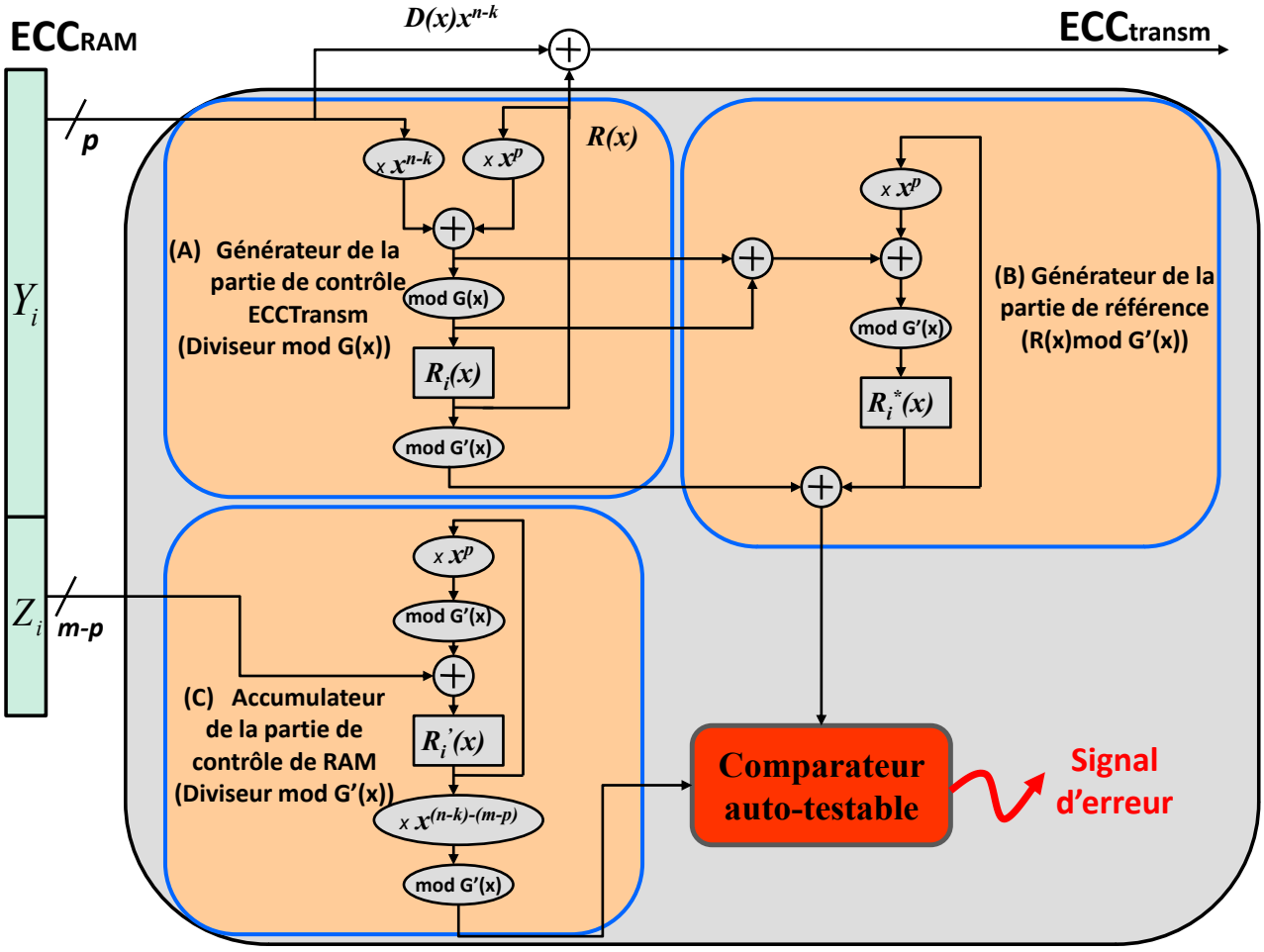


FIG. 4.5 – Architecture détaillée d'un codeur FS (*Fault-Secure*) parallèle

contrôler le codeur à chaque cycle d'horloge. Cette égalité sera surveillée par le comparateur auto-testable (voir figure 4.3). Pour cela, une nouvelle suite de polynômes $R^*_i(x)$ de degré $(m-p)$ doit être determinée à additionner à $R_i(x)$ afin de générer les bits de contrôle de $R''(x)$ (qui n'est autre que le

diviseur "$[R(x) + R^*(x)] \bmod G'(x)$" [71] de degré $(m-p)$) :

$$R_i^*(x) + R_i(x) \bmod G'(x) = [R_i'(x) \cdot x^{n-k-(m-p)}] \bmod G'(x), \;\; 1 \leq i \leq l. \tag{4.13}$$

Les équations ci-dessous montrent une telle relation de récurrence pour $R_i^*(x)$ indépendamment de $R_i'(x)$:

$$\begin{aligned}
R_0^*(x) &= 0 \\
R_{i+1}^*(x) &= [R_{i+1}(x) + R_{i+1}'(x) \cdot x^{n-k-(m-p)}] \bmod G'(x) \\
&= [(R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}) \bmod G(x) \\
&\quad + (R_i'(x) \cdot x^p + Y_i(x) \cdot x^{m-p}) \cdot x^{n-k-(m-p)}] \bmod G'(x) \\
&= [(R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}) \bmod G(x) \\
&\quad + R_i'(x) \cdot x^{m-p} \cdot x^p + Y_i(x) \cdot x^{n-k}] \bmod G'(x) \\
&= [(R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}) \bmod G(x) \\
&\quad + (R_i^*(x) + R_i(x)) \cdot x^p + Y_i(x) \cdot x^{n-k}] \bmod G'(x) \\
&= [R_i^*(x) \cdot x^p) \bmod G'(x) + ((R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}) \bmod G(x) \\
&\quad + R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}] \bmod G'(x).
\end{aligned} \tag{4.14}$$

Pour générer correctement les bits de contrôle de référence $R_i''(x)$, il est très important de signaler que $R_i^*(x)$ doit être indépendant de l'autre branche du comparateur, celle générant $R_i'(x)$. En effet, il doit y avoir deux chemins différents et indépendants provenant respectivement de $Y(x)$ et $Z(x)$ (de la RAM jusqu'au le comparateur auto-testable du codeur FS) pour contrôler sûrement $R''(x) = [R'(x) \cdot x^{n-k-(m-p)}] \bmod G'(x)$.

La figure 4.5 présente l'architecture générale implantant l'équation (4.13) et (4.14). Elle est composée de trois blocs [70, 71] : (A) le générateur de la partie contrôle du code $\mathrm{ECC}_{\mathrm{Transm}}$, (B) le générateur de la partie de référence $R''(x)$ et (C) l'accumulateur des bits de contrôle de RAM $R'(x)$, à comparer à $R''(x)$ à chaque cycle d'horloge. L'opération de multiplication x^p (ou x^{n-k}) consiste dans la pratique à ajouter p (ou $n-k$) zéros à la fin du message $D(x)$. Cette opération n'induit aucun coût de surface additionnel au circuit, et correspond à un simple changement des indices des données en entrée.

Comme nous le montrerons dans la section d'évaluation de la tolérance par la méthode d'injection d'erreurs, la majorité des fautes temporaires simples et multiples sont détectées par le comparateur auto-testable (voir figure 4.5). Par une approche formelle, nous avons pu d'identifier une classe d'erreurs pour laquelle la détection est toujours assurée : toutes les fautes détectables par le polynôme $G'(x)$ le seront au niveau du comparateur auto-testable. La démonstration formelle de cette approche est présentée dans la prochaine section.

4.4.2 Autre propriété de détection

L'objectif de cette section est de montrer que l'approche proposée permet non seulement de détecter les erreurs induites dans le codeur, mais aussi de détecter ou vérifier si les données lues à partir de la RAM sont correctes ou non. Ainsi tout mot non code $W'(x)$ est détecté par le codeur FS. Par la suite, nous allons montrer que toutes les erreurs à être détectées par le polynôme $G'(x)$, utilisés pour protéger la RAM, sont également détectés par le codeur FS parallèle. Rappelons que dans la RAM tolérante aux fautes, le mot code $W_i(x)$ est défini par l'équation (4.2) [71] :

$$W_i(x) = Y_i(x) \cdot x^{m-p} + Z_i(x), \tag{4.15}$$

où $Y_i(x)$ et $Z_i(x)$ sont respectivement les $p$bits de donnée et les $(m-p)$bits de contrôle.

Supposons qu'une erreur détectable par $G'(x)$ affecte les données entre la RAM et le codeur FS : le mot ainsi modifié présent à l'entrée du codeur FS est noté $W'_i(x)$. Nous pouvons remarquer que dans ce cas

$$W'_i(x) \bmod G'(x) \neq 0. \tag{4.16}$$

Nous pouvons récrire $W'_i(x)$ comme

$$W'_i(x) = \widehat{W_i(x)} + \widehat{E_i(x)}, \tag{4.17}$$

où

$$\begin{aligned} \widehat{W_i(x)} &= W'_i(x) + W'_i(x) \bmod G'(x) \\ &\text{et} \\ \widehat{E_i(x)} &= W'_i(x) \bmod G'(x) \end{aligned} \tag{4.18}$$

cela peut être vu comme une décomposition de $W'_i(x)$ en : mot code correct $\widehat{W_i(x)}$ et contribution de l'erreur $\widehat{E_i(x)}$. On peut décomposer $\widehat{W_i(x)}$ en :

$$\widehat{W_i(x)} = \widehat{Y_i(x)} \cdot x^{m-p} + \widehat{Z_i(x)}, \tag{4.19}$$

où

$$\widehat{Z_i(x)} = (\widehat{Y_i(x)} \cdot x^{m-p}) \bmod G'(x). \tag{4.20}$$

Par conséquent,

$$W'_i(x) = (\widehat{Y_i(x)} \cdot x^{m-p}) + (\widehat{Z_i(x)} + \widehat{E_i(x)}). \tag{4.21}$$

On peut remarquer que les degrés de $\widehat{Z_i(X)}$ et $\widehat{E_i(X)}$ sont inférieurs à $(m-p)$ (degré de $G'(X)$). Par conséquent, $(\widehat{Z_i(X)} + \widehat{E_i(X)})$ devrait correspondre à la partie de contrôle $\widehat{Y_i(X)}$, mais ce n'est pas le cas, car :

$$(\widehat{Y_i(x)} \cdot x^{m-p}) \bmod G'(X) \neq (\widehat{Z_i(x)} + \widehat{E_i(x)}). \tag{4.22}$$

L'identité devrait être contrôlée à chaque cycle d'horloge par le comparateur (Eqn(4.13)) est :

$$\widehat{R^*_{i+1}(x)} + \widehat{R_{i+1}(x)} \bmod G'(x) = [\widehat{R'_{i+1}(x)} \cdot x^{n-k-(m-p)}] \bmod G'(x). \tag{4.23}$$

Dans l'équations (4.8), (4.12), et (4.14) nous pouvons remplacer respectivement $R_{i+1}(x)$, $R'_{i+1}(x)$, et $R^*_{i+1}(x)$. Du côté gauche de l'équation (4.23), on obtient :

$$\begin{aligned} R''_{i+1}(x) =& (R^*_i(x) \cdot x^p) \bmod G'(x) + [(R_i(x) \cdot x^p + \widehat{Y_i(x)} \cdot x^{n-k}) \bmod G(x)] \bmod G'(x) \\ &+ (R_i(x) \cdot x^p + \widehat{Y_i(x)} \cdot x^{n-k}) \bmod G'(x) \\ &+ [(R_i(x) \cdot x^p + \widehat{Y_i(x)} \cdot x^{n-k}) \bmod G(x)] \bmod G'(x) \\ =& [(R^*_i(x) + R_i(x)) \cdot x^p + \widehat{Y_i(x)} \cdot x^{n-k})] \bmod G'(x) \\ =& [R'_i(x) \cdot x^{n-k-(m-p)} \cdot x^p + \widehat{Y_i(x)} \cdot x^{n-k})] \bmod G'(x) \end{aligned} \tag{4.24}$$

et du côté droit de l'équation (4.23) :

$$\begin{aligned} \widehat{R'_{i+1}(x)} =& (R'_{i+1}(x) \cdot x^{n-k-(m-p)}) \bmod G'(x) \\ =& [R'_i(x) \cdot X^p) \bmod G'(x) + (\widehat{Z_i(x)} + \widehat{E_i(X)})] \cdot x^{n-k-(m-p)}] \bmod G'(x) \\ =& [R'_i(x) \cdot x^{n-k-(m-p)} \cdot x^p \\ &+ ((\widehat{Y_i(x)} \cdot X^{m-p}) \bmod G'(x) + \widehat{E_i(x)}) \cdot x^{n-k-(m-p)}] \bmod G'(x) \\ =& [R'_i(x) \cdot x^{n-k-(m-p)} \cdot x^p + \widehat{Y_i(x)} \cdot x^{n-k} + \widehat{E_i(x)} \cdot x^{n-k-(m-p)}] \bmod G'(x). \end{aligned} \tag{4.25}$$

Nous pouvons facilement prouver que $R''_{i+1}(x) \neq \widehat{R'_{i+1}(x)}$, car $E'(x) \neq 0$. Ce désaccord est signalé par le comparateur auto-testable. Par conséquent, les erreurs détectées par $G'(x)$ sont aussi détectables par le codeur FS.

4.4.3 Possibilités de simplification

En éliminant le polynôme $R^*(x)$ dans l'équation (4.13), il est possible de réduire le surcoût matériel nécessaire pour implanter le codeur FS (et de réduire ainsi la consommation d'énergie). Dans ce cas, le calcul des bits de références $R''_i(x)$ ne dépend que du polynôme $R(x)$, ce qui permet de réduire la complexité du codeur FS.

Maintenant, nous allons déterminer la relation entre $G(x)$ et $G'(x)$ permettant d'aboutir à l'élimination de $R^*(x)$.

Notons que lorsque $G(x)$ peut être décomposé en $G(x) = Q(x) \cdot G'(x)$ (la valeur $Q(x)$ étant sans importance ici), alors [71] :

$$\begin{aligned} R^*_0(x) &= 0 \\ R^*_{i+1}(x) &= (R^*_i(x) \cdot x^p) \bmod G'(x) = 0, \end{aligned} \tag{4.26}$$

Cela signifie que

$$\forall\ i, \quad R_i^*(x) = 0. \tag{4.27}$$

Lorsque le polynôme $R^*(x) = 0$, seul les deux blocs (A) et (C) sont nécessaire spour réaliser le codeur FS (figure 4.5), ce qui réduit par conséquent la complexité du codeur comme cela sera montré dans la prochaine section.

Soit $P(x) = R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}$. On peut montrer que :

$$(P(x) \bmod G(x) + P(x)) \bmod G'(x) = 0. \tag{4.28}$$

Dans la suite nous allons prouver les équations (4.26) et (4.28). Deux polynômes $U(x)$ et $V(x)$ peuvent être définis comme suit :

$$\begin{aligned} U(x) &= P(x) \bmod G(x) \\ V(x) &= P(x) \bmod G'(x), \end{aligned} \tag{4.29}$$

ce qui est équivalant à :

$$\begin{aligned} P(x) &= Q'(x) \cdot G(x) + U(x) \\ &= Q''(x) \cdot G'(x) + V(x). \end{aligned} \tag{4.30}$$

Soit $G''(x)$ le reste de la division du polynôme $G(x)$ par $G'(x)$, c'est-à-dire $G(x) = Q(x) \cdot G'(x) + G''(x)$. Dans le cas général, nous avons :

$$\begin{aligned} &[P(x) \bmod G(x) + P(x)] \bmod G'(x) \\ &\quad = [U(x) + P(x)] \bmod G'(x) \\ &\quad = [P(x) + Q'(x)G(x) + P(x)] \bmod G'(x) \\ &\quad = [Q'(x)G(x)] \bmod G'(x) \\ &\quad = [Q'(x)[Q(x)G'(x) + G''(x)]] \bmod G'(x) \\ &\quad = [Q'(x)G''(x)] \bmod G'(x). \end{aligned} \tag{4.31}$$

En particulier, si $G''(x) = G(x) \bmod G'(x) = 0$, alors :

$$\begin{aligned} &[P(x) \bmod G(x) + P(x)] \bmod G'(x) \\ &\quad = [Q'(x)G''(x)] \bmod G'(x) \\ &\quad = 0. \end{aligned} \tag{4.32}$$

Il est très intéressant de noter que lorsque $G(x)$ est multiple de $G'(x)$, alors $R^*(x) = 0$ dans l'équation (4.26). Dans ce cas, le codeur FS de la figure 4.5 est réduit aux deux blocs (A) et (C). Ains, la complexité du codeur FS peut être réduite.

Ici, nous allons démontrer que : 1) le schéma général du codeur FS de la figure 4.3 peut être

simplifié lorsque $G'(x)$, le polynôme générateur du code Hamming utilisées pour protéger la RAM, est un facteur du polynôme générateur $G(x)$ utilisé pour protéger le canal de transmission ; 2) c'est le cas pour différents codes cycliques en blocs tels que les codes CRC, BCH, et RS. Supposons que $G(x)$ soit le produit de s polynômes primitifs $G_j(x)$

$$G(x) = \prod_{l=1}^{s} G_j(x), \quad s > 1. \tag{4.33}$$

Si l'un de ces polynômes primitifs $G_j(x)$ en particulier est le polynôme générateur $G'(x)$ de $\mathrm{ECC_{RAM}}$, et donc, $G(x)$ est un multiple de $G'(x)$, alors l'équation (4.13) peut être simplifiée en :

$$R_i(x) \bmod G'(x) = [R'_i(x) \cdot x^{n-k-(m-p)}] \bmod G'(x). \tag{4.34}$$

Par conséquent, nous pouvons prendre avantage de $R^*_i(x) = 0$ pour tout i, $0 \leq i \leq j-1$ (ce qui sera démontré plus tard) pour contrôler directement l'équation précédente à la place de (4.13).

Pour démontrer la faisabilité et l'utilité de ce qui précède, considérons quelques codes CRC, BCH, et RS générés par des polynômes $G(x)$, et quelques codes de Hamming cycliques générés par des polynômes $G'(x)$.

Pour le premier exemple, considérons le code CRC-32 bits utilisé pour la transmission de données, dont le polynôme générateur $G(x)$ est :

$1+x+x^2+x^4+x^6+x^7+x^{10}+x^{11}+x^{15}+x^{16}+x^{17}+x^{19}+x^{20}+x^{26}+x^{28}+x^{29}+x^{30}+x^{32}$, pouvant être décomposé en :

$G(x) = (1+x)(1+x^2+x^3)(1+x^6+x^8+x^9+x^{12}+x^{14}+x^{16}+x^{19}+x^{20}+x^{22}+x^{28})$.
Si la mémoire RAM est protégée par le code de parité dont le polynôme générateur $G'(x) = (x+1)$, alors le polynôme $G'(x)$ utilisé pour protéger la RAM est un facteur de polynôme $G(x)$. Malheureusement, aucun des codes Hamming cycliques SEC ou SEC/DED utilisés pour proteger la RAM ne peut être générés par l'un des trois facteurs du polynôme $G(x)$.

Prenons maintenant le code BCH (63,39,4) dont le polynôme générateur $G(x)$ est :

$1+x+x^2+x^4+x^5+x^6+x^8+x^9+x^{10}+x^{13}+x^{16}+x^{17}+x^{19}+x^{20}+x^{22}+x^{23}+x^{24}$, pouvant être décomposé en :

$G(x) = (1+x+x^6)(1+x+x^2+x^4+x^6)(1+x+x^2+x^5+x^6)(1+x^3+x^6)$.
Pour protéger la RAM par un code SEC ou SEC/DED, on peut choisir le polynôme $G'_1(x) = (1+x+x^6)$ ou $G'_2(x) = (1+x+x^2+x^5+x^6)$ générant des codes cycliques de Hamming. Dans les deux cas, le polynôme sélectionné $G'(x)$ est un facteur du polynôme $G(x)$. Cependant, les autres facteurs de $G(x)$ ne correspondent a aucun polynôme $G'(x)$ générant des code de Hamming.

Considérons enfin le code RS (15,9) capable de corriger trois erreurs avec des symboles définis dans le corps de Galois $GF(2^4)$. Le polynôme générateur de ce code est :

$$G(x) = (\alpha+x)(\alpha^2+x)(\alpha^3+x)(\alpha^4+x)(\alpha^5+x)(\alpha^6+x),$$

où les coefficients α^i sont définies dans $GF(2^4)$.

Supposons maintenant que le polynôme primitif utilisé pour construire le corps de Galois $GF(2^4)$ est $G'(x) = 1 + x + x^4$. Si le même polynôme est utilisé pour générer le code de Hamming $\mathrm{ECC}_{\mathrm{RAM}}$, alors le polynôme $G'(x)$ sera un facteur de $G(x)$, et le circuit simplifié correspondre à l'équation 4.34 au lieu de 4.13.

Pour les exemples ci-dessus, le polynôme $G(x)$ est multiple de polynôme $G'(x)$ et donc $R_i^*(x) = 0$. Le calcul de la partie de référence $R''(x)$ est réduit à :

$$R_i''(x) = R_i(x) \bmod G'(x). \tag{4.35}$$

Ci-dessus, nous avons bien montré que c'est pour différents codes BCH et RS, le polynôme $G'(x)$ choisi pour générer $\mathrm{ECC}_{\mathrm{RAM}}$ peut être égal à l'un des polynômes $G_l(x)$, et par conséquence $R^*(x) = 0$. Malheureusement, il est impossible d'annuler le facteur $R^*(x)$ pour la plupart des codes CRC connus. En effet, nous avons observé que relativement peu de polynômes générant des codes de Hamming cyclique correspondent à des polynômes primitifs $G_l(x)$. Ainsi dans le cas général, le choix des codes CRC conduisent à $R^*(x) \neq 0$ (la seule exception lorsque la RAM est protégée par le code de parité).

Par contre, tous les polynômes $G(x)$ générant des codes BCH et RS sont définis dans le corps de Galois $GF(2^n)$, où n est le degré du polynôme primitive $G_i(x)$ (utilisé pour construire le corps de Galois), et donc le polynôme $G(x)$ défini dans $GF(2^{n-k})$ est toujours multiple de $G_i(x)$. En choisissant $G'(x)$ identique à $G_i(x)$, $G(x)$ est un multiple de $G'(x)$, et donc $R^*(x) = 0$. Nous montrerons dans la section suivante que c'est l'une des raisons, et donc pour laquelle le surcoût matériel pour implanter un codeur FS avec un code CRC pour $\mathrm{ECC}_{\mathrm{Transm}}$ est relativement élevée par rapport au codes BCH et RS.

4.5 Aspects fonctionnels et architecturaux

L'objectif est de montrer l'efficacité de l'approche proposée ici pour concevoir un codeur FS par rapport à la méthode de tolérance classique (duplication avec comparaison). Dans un premier temps, nous évaluons la tolérance de notre approche à l'aide de plusieurs campagnes d'injections de fautes divers simples et multiples. Ainsi le taux de détection d'erreurs est calculé pour chaque campagne d'injections. Ensuite, des comparaisons entre la complexité (surface consommé) et la performance (débit) des codeurs FS parallèles sont réalisées par rapport à leur homologues simples, pour différentes combinaisons de codes linéaires cycliques ($G(x)$ et $G'(x)$).

Les deux aspects fonctionnels et architecturaux sont évalués. Du point de vue fonctionnel, la capacité de détection d'erreurs des codeurs FS est évaluée par la méthode d'injection de fautes. En ce qui concerne l'aspect architectural, le surcoût matériel et la fréquence maximale de fonctionnement (et par conséquent, les débits de données) sont étudiés et comparés à ceux obtenus avec versions simple (S).

TAB. 4.1 – Quelques polynômes génèrent codes Hamming

$G'_1(x)$	$1+x$ ‡	$G'_{10}(x)$	$1+x+x^4+x^5+x^6$ †
$G'_2(x)$	$1+x+x^4$	$G'_{11}(x)$	$1+x+x^2+x^5+x^6$ †
$G'_3(x)$	$1+x^3+x^4$	$G'_{12}(x)$	$1+x+x^3+x^4+x^6$ †
$G'_4(x)$	$1+x+x^2+x^3+x^5$ †	$G'_{13}(x)$	$1+x+x^7$
$G'_5(x)$	$1+x+x^2+x^4+x^5$ †	$G'_{14}(x)$	$1+x^2+x^4+x^6+x^7$ †
$G'_6(x)$	$1+x+x^3+x^4+x^5$ †	$G'_{15}(x)$	$1+x+x^2+x^3+x^7$ †
$G'_7(x)$	$1+x^2+x^3+x^4+x^5$ †	$G'_{16}(x)$	$1+x^2+x^3+x^4+x^7$ †
$G'_8(x)$	$1+x+x^6$	$G'_{17}(x)$	$1+x^2+x^3+x^4+x^8$ †
$G'_9(x)$	$1+x^5+x^6$		

† – code SEC/DED, ‡ – code de parité, sinon : code Hamming cyclique SEC

Différentes configurations de codeurs FS sont analysées, correspondant à trois niveaux de parallélisme $p = 8$, 16 et 32 (largeur de mot de RAM) et à plusieurs combinaisons de polynômes $G(x)$ et $G'(x)$. Les polynômes $G(x)$ sont présentés dans les tableaux 4.1, 4.2 et 4.3. Ce sont des polynômes standards et couramment employés dans divers systèmes de transmission pour coder les données. Ils sont regroupés en quatre classes de codes : codes de Hamming, CRC, BCH et RS. Les polynômes $G'(x)$ sont présentés dans le tableau 4.1. Ces polynômes correspondent au code de parité simple ($G'_1(x)$) ou à des codes de Hamming cycliques SEC et SEC/DED ($G'_2(x)$ au $G'_{17}(x)$).

TAB. 4.2 – Quelques polynômes génèrent codes CRC standards

$G(x)$	Polynômes	Nom utilisé
$G_1(x)$	$1+x^2+x^{15}+x^{16}$	CRC-16-CCITT
$G_2(x)$	$1+x+x^3+x^4+x^5+x^6+x^7+x^{10}$ $+x^{11}+x^{14}+x^{17}+x^{18}+x^{23}+x^{24}$	CRC-24-MIL-STD
$G_3(x)$	$1+x+x^2+x^4+x^6+x^7+x^{10}+x^{11}+x^{15}+x^{16}$ $+x^{17}+x^{19}+x^{20}+x^{26}+x^{28}+x^{29}+x^{30}+x^{32}$	CRC-32-Ethernet MTU
$G_4(x)$	$1+x+x^2+x^4+x^5+x^7+x^8+x^{10}+x^{11}$ $+x^{12}+x^{16}+x^{22}+x^{23}+x^{26}+x^{32}$	CRC-32-802.3 Ethernet
$G_5(x)$	$1+x+x^4+x^7+x^9+x^{10}+x^{12}+x^{13}+x^{17}+x^{19}$ $+x^{21}+x^{22}+x^{23}+x^{24}+x^{27}+x^{29}+x^{31}+x^{32}$ $+x^{33}+x^{35}+x^{37}+x^{38}+x^{39}+x^{40}+x^{45}+x^{46}$ $+x^{47}+x^{52}+x^{53}+x^{54}+x^{55}+x^{57}+x^{62}+x^{64}$	CRC-64-ECMA-182

4.5.1 Analyse fonctionnelle et validation par injection d'erreurs

L'objectif de la tolérance aux fautes est de maintenir le fonctionnement correct du matériel, malgré la présence de fautes, qu'il s'agisse de dégradations physiques du matériel, de défauts logiciels, etc. L'évaluation de la tolérance peut être réalisée par la méthode d'injection de fautes dans l'architecture. La simulation basée sur l'injection de fautes est un cas particulier de l'injection logicielle de fautes qui "peut soutenir une variété de niveaux d'abstraction du système : architecturale, fonctionnelle, logique et électrique" [128], et pour cet raison elle a été largement utilisée. La technique de base utilisée consiste à instrumenter de manière bien contrôlée le modèle de la conception pour intégrer l'injection

TAB. 4.3 – Quelques polynômes génèrent codes BCH et codes RS standards

$G(x)$	Polynômes	Nom
$G_6(x)$	$1+x+x^2+x^4+x^5+x^6+x^8+x^9+x^{10}+x^{13}$ $+x^{16}+x^{17}+x^{19}+x^{20}+x^{22}+x^{23}+x^{24}$	BCH(63,39,4)
$G_7(x)$	$1+x^2+x^3+x^6+x^{12}+x^{13}+x^{15}+x^{19}$ $+x^{20}+x^{22}+x^{23}+x^{24}+x^{28}+x^{39}+x^{40}$ $+x^{43}+x^{46}+x^{47}+x^{49}$	BCH(127,78,7)
$G_8(x)$	$1+x^2+x^5+x^{15}+x^{18}+x^{19}+x^{21}+x^{22}+x^{23}$ $+x^{24}+x^{25}+x^{26}+x^{30}+x^{31}+x^{32}+x^{33}+x^{35}$ $+x^{36}+x^{38}+x^{40}+x^{47}+x^{48}+x^{49}+x^{51}$ $+x^{53}+x^{55}+x^{56}+x^{61}+x^{63}$	BCH(127,64,10)
$G_9(x)$	$\alpha^{105}+\alpha^{128}x+\alpha^{128}x^2+\alpha^{163}x^3+\alpha^{20}x^4+\alpha^{119}x^5$ $+\alpha^{194}x^6+\alpha^{115}x^7+\alpha^{29}x^8+\alpha^{190}x^9+\alpha^{89}x^{10}+\alpha^{8}x^{11}$ $+\alpha^{230}x^{12}+\alpha^{200}x^{13}+x^{14}$	RS(255,241,6)

Les coefficients α^i du polynôme $G_9(x)$ sont définis dans $GF(2^8)$.

de fautes, et à vérifier ensuite le comportement du circuit fautif lors de son fonctionnement [205]. Les fautes considérées ici sont les SEU (un seul bit change dans un seul registre suite à l'impact d'une particule) et les MBU (de multiples bits changent à la fois dans un seul registre). Ces deux modèles d'erreurs (SEU et MBU) sont couramment utilisés avec les modèles RTL [98, 99]. Dans la suite, le terme *faute* signifie SEU ou MBU.

Plusieurs codeurs FS ont été implantés pour différentes configurations de polynômes $G(x)$ et $G'(x)$. Les configurations sont celles des tableaux 4.6 et 4.7. Comme indiqué précédemment, pour toutes les configurations où $G(x)$ est multiple de $G'(x)$, le bloc $R^*(x)$ n'existe pas, et donc aucune d'injection de fautes n'est nécessaire dans ce bloc. Pour chaque circuit, deux campagnes (soit d'expériences d'injection d'erreurs en séries) ont été organisés : une campagne d'injection d'erreurs simples (SEU), doubles et triples (MBU), et une autre campagne pour l'injection d'erreurs aléatoires de poids supérieur à 6. Les résultats sont présentés dans les tableaux 4.4 et 4.5, respectivement. À noter que pour chaque configuration, le nombre spécifié d'erreurs injectées correspond au nombre total d'erreurs injectées dans les registres correspondant à $R(x)$, $R^*(x)$, et $R'(x)$. Après une erreur injectée, le comparateur auto-testable indique si l'erreur est détectée ou non dans un délai maximum de 3 cycles (latence maximale du codeur). Chaque simulation est ainsi arrêtée 3 cycles après l'injection, que l'erreur soit détectée ou non. Puis une nouvelle simulation démarre avec une nouvelle faute injectée. Le "nombre d'injections d'erreurs" est donc le nombre total d'erreurs injectées sur l'ensemble des simulations, et le "taux de détection d'erreur" correspondant au nombre (en %) de fautes détectées par le comparateur auto-testable. Une erreur se manifeste sur un registre $R(x)$, $R^*(x)$ ou $R'(x)$, si sa valeur dans le modèle fautif diffère de sa valeur dans le modèle sans erreur.

La table 4.4 résume les taux de détection d'erreurs pour la figure 4.5, obtenus en utilisant une injection de fautes simple, double ou triple. On peut facilement remarquer que le taux de détection d'erreurs

TAB. 4.4 – Taux de détection d'erreurs pour une injection d'erreur simple, double et triple

Numéro de config.	Nombre d'injections d'erreur	Taux détection d'erreurs [%]								
		Injection simple			Injection double			Injection triple		
		$R(X)$	$R^*(X)$	$R'(X)$	$R(X)$	$R^*(X)$	$R'(X)$	$R(X)$	$R^*(X)$	$R'(X)$
1	512	98,05	–	99,41	97,07	–	95,51	75,59	–	77,54
2	22454	98,76	–	99,69	94,96	–	95,99	75,15	–	79,65
3	63513	99,75	–	98,45	98,02	–	98,17	85,52	–	86,02
4	618	82,85	–	81,72	67,80	–	67,31	59,55	–	60,52
5	25226	89,85	–	91,10	84,31	–	84,59	70,08	–	70,83
6	64301	98,26	–	98,33	93,71	–	94,02	81,42	–	81,90
7	1551	89,75	–	86,65	80,34	–	81,69	69,70	–	69,37
8	25321	91,82	–	90,38	91,39	–	88,16	92,20	–	88,32
9	76557	93,09	–	94,78	91,62	–	92,29	91,57	–	91,65
10	2621	87,94	–	89,81	90,58	–	89,43	80,35	–	83,25
11	29435	87,85	–	87,21	86,82	–	86,50	69,34	–	70,58
12	69783	93,65	–	92,67	95,41	–	85,80	82,77	–	82,57
13	3651	91,43	–	90,33	93,13	–	84,74	91,40	–	83,46
14	25762	89,16	–	87,16	91,23	–	79,63	76,35	–	76,81
15	77259	94,55	–	94,81	93,73	–	93,54	93,44	–	91,85
16	1241	90,81	–	91,62	93,80	–	85,74	90,49	–	83,16
17	21516	97,48	–	95,49	92,38	–	92,48	94,65	–	93,73
18	72323	99,10	–	97,29	97,68	–	97,08	92,01	–	93,19
19	2159	95,65	96,11	96,43	92,96	93,19	93,14	95,65	95,18	95,37
20	23426	96,38	95,83	95,84	91,92	92,33	90,95	83,05	83,86	83,90
21	37828	98,81	98,94	98,97	92,20	92,32	91,53	83,50	84,22	84,27
22	2436	99,51	96,22	96,63	85,34	86,12	86,12	72,91	72,99	72,41
23	23588	97,40	96,13	96,46	93,15	92,28	91,95	74,76	74,84	75,03
24	46236	99,83	99,81	99,85	94,47	94,57	94,63	80,86	80,95	72,99
25	3531	97,17	97,45	98,13	92,35	91,02	91,31	79,24	77,99	78,31
26	24866	99,28	97,55	97,59	95,21	95,98	95,52	72,33	70,96	70,96
27	55286	98,46	99,01	98,84	97,77	97,77	97,78	97,66	96,81	96,79

pour une injection de faute simple est élevé : plus de 81% pour les configurations 1 à 18 (où $G(x)$ est multiple de $G'(x)$) et plus de 89% pour toutes les autres configurations (où $G(x)$ n'est pas multiple de $G'(x)$).

La couverture d'erreur reste raisonnable (avec une diminution linéaire légère) pour une injection de fautes doubles ou triples, avec des taux supérieurs à 67% et 59%, respectivement. On peut aussi remarquer que les cas les moins favorables se presentent lorsque les erreurs sont injectées dans $R(x)$. Cela est dû probablement aux effets d'*aliasing* qui se produisent dans le cycle d'horloge suivant l'injection de faute. En effet, la sortie de $R(x)$ est transmise vers $R''(x)$ via deux chemins différents (voir figure 4.5). Ces chemins se combinent dans l'additionneur pour générer le polynôme $R''(x)$, quelques masquages partiels des erreurs pouvant se produire.

La table 4.5 rassemble les résultats obtenus par injection de fautes aléatoires de vecteurs de poids supérieur à 6. Le but était ici d'évaluer le comportement du codeur FS dans des conditions très défavorables de fonctionnement. En dépit de ces conditions, le taux de détection d'erreurs reste acceptable (environ 50%) dans toutes les configurations.

TAB. 4.5 – Taux de détection d'erreur pour injection d'erreurs aléatoires

Numéro de config.	Nombre d'injections d'erreur	Taux de détection[%] Injection aléatoires			Nombre de Config.	Numéro d'injections d'erreur	Taux de détection[%] Injection aléatoires		
		$R(X)$	$R^*(X)$	$R'(X)$			$R(X)$	$R^*(X)$	$R'(X)$
1	512	$50,20$	–	$51,76$	16	1241	$50,68$	–	$48,99$
2	22454	$49,59$	–	$50,94$	17	21516	$49,96$	–	$49,83$
3	63513	$50,19$	–	$49,75$	18	72323	$49,61$	–	$50,05$
4	518	$50,58$	–	$51,26$	19	2159	$51,51$	$50,86$	$50,02$
5	23226	$50,55$	–	$50,43$	20	23426	$49,10$	$50,31$	$49,93$
6	64301	$49,20$	–	$50,01$	21	37828	$49,72$	$50,17$	$50,20$
7	1451	$48,04$	–	$49,69$	22	2436	$48,07$	$47,54$	$50,37$
8	24321	$50,18$	–	$49,28$	23	23588	$49,69$	$49,92$	$49,91$
9	74557	$49,93$	–	$50,03$	24	46236	$49,42$	$50,02$	$50,29$
10	2421	$49,44$	–	$50,69$	25	3531	$51,43$	$50,07$	$50,78$
11	26435	$50,43$	–	$50,25$	26	24866	$50,39$	$50,26$	$50,28$
12	66783	$50,15$	–	$51,23$	27	55286	$50,01$	$5,11$	$50,05$
13	3451	$51,20$	–	$49,38$					
14	23762	$49,92$	–	$49,61$					
15	74259	$49,98$	–	$49,89$					

Ceci nous permet de conclure que cette approche de protection matérielle est très avantageuse. Pour des injections d'erreurs simples, 90% environ des erreurs sont détectées dans les pluparts de configurations. De même, pour des injections de fautes doubles ou triples, la capacité de détection est importante, respectivement de 67% et 59%. Dans les conditions défavorables d'injection des fautes de poids > 6, la correction reste possible avec un taux environ 50%. Par contre, si l'application est située dans un environnement plus perturbé avec un taux d'erreur supérieur à 9, ce qui est très rare, le recour à d'autres techniques de protection matérielles devient nécessaire.

En résumé, les expériences présentées dans cette section montrent sans le moindre doute que l'architecture proposée offre des avantages nets en termes de capacité de détection d'erreur. Par conséquent, l'approche FS proposée peut être efficacement utilisée dans des applications exigeant un niveau de fiabilité important.

4.5.2 Aspect architectural

Les codeurs FS ci-dessus proposés et leurs homologues simple, qui serviront de référence dans le cadre d'une duplication, ont tous été synthétisés sur FPGA de la famille StratixII à l'aide du logiciel Altera/Quartus II. Il est à noter que l'architecture proposée dans ce chapitre n'aborde pas le problème de fautes/erreurs pouvant apparaître dans la mémoire de configuration du FPGA. Le choix d'utiliser des FPGA pour implanter les codeurs s'est imposé par leur facilité d'emploi, le but étant seulement de mesurer le surcoût matériel et la dégradation de vitesse relatifs a des versions FS par rapport aux simples (S) (duplication), dans le but de les comparer.

Les mesures de surface consommée et de performance ont été regroupées en deux tableaux : les tables 4.6 et 4.7 présentent respectivement les cas lorsque où $G(x)$ *est* ou *n'est pas* multiple

de $G'(x)$. La consommation en surface est mesurée par le nombre de blocs logiques configurables CLBs nécessaires pour implanter le codeur. La performance (débit) du codeur est mesurée par la fréquence f_{max} maximale de fonctionnement multipliée par le degré de parallélisme p. Les polynômes $W_i(x)$ apparaissant dans les tableaux correspondent aux restes de la division de $G(x)$ par $G'(x)$. Leurs valeurs respectives se trouvent dans le tableau 4.8. Il est évident que, $W_0(x) = 0$.

Dans les pluparts des cas, la complexité du codeur croît avec $(n - k)$, le degré de $G(x)$. On peut observer que le surcoût matériel relatif aux blocs supplémentaires (correspondant au $R'(x)$ et $R''(x)$) diminue avec le niveau de parallélisme p.

Avec une RAM protégée par un code de parité (de polynôme générateur $G'_1(x) = 1 + x$), les données du tableau 4.6 montrent que, pour les configurations 1–6 (correspondant aux deux premiers polynômes $G_1(x)$ et $G_3(x)$ générant des codes CRC), le surcoût matériel du codeur FS est de 4 à 35%. À noter que le surcoût matériel relatif au polynôme $G_3(x)$ est inférieur à celui du polynôme $G_1(x)$, puisque le degré du polynôme $G_3(x)$ est deux fois celui du polynôme $G_1(x)$ 32 et 16, respectivement.

TAB. 4.6 – Comparatif (surface et vitesse) du codeur FS et S lorsque $G(x)$ est multiple de $G'(x)$

Numéro de config.	Polynômes			p	No. CLBs		Surcoût matériel [%]	$f_{\max}$ [MHz]		Réduction de fréq.[%]
	$G(X)$	$G'(X)$	$W(X)$		A_S	A_{FS}		f_S	f_{FS}	
1	$G_1(x)$	$G'_1(x)$ ‡	$W_0(x)$	8	32	43	34,37	500	450,25	9,95
2				16	36	48	33,33	413,05	287,44	30,41
3				32	45	59	31,11	381,53	329,6	13,61
4	$G_3(x)$	$G'_1(x)$ ‡	$W_0(x)$	8	68	81	19,11	500	475,06	4,90
5				16	87	101	16,00	500	438,79	12,24
6				32	132	137	3,78	407,33	354,48	12,97
7	$G_6(x)$	$G'_8(x)$	$W_0(x)$	8	54	86	**59,25**	500	329,6	34,08
8		$G'_{11}(x)$ †		16	66	101	**53,03**	408,16	337,61	17,28
9		$G'_{10}(x)$ †		32	96	125	30,20	282,81	250,38	11,60
10	$G_7(x)$	$G'_{13}(x)$	$W_0(x)$	8	103	151	46,60	500	259,81	48,20
11		$G'_{14}(x)$ †		16	129	179	38,75	370,92	237,08	36,08
12		$G'_{15}(x)$ †		32	184	238	29,34	265,67	217,16	18,25
13	$G_8(x)$	$G'_{13}(x)$	$W_0(x)$	8	133	186	39,84	423,91	216,83	48,84
14		$G'_{16}(x)$ †		16	163	221	35,58	333,44	219,3	34,23
15		$G'_{14}(x)$ †		32	246	303	23,17	238,83	190,84	20,09
16	$G_9(x)$	$G'_{17}(x)$ †	$W_0(x)$	8	1504	1727	14,80	228,68	223,71	2,17
17				16	1933	2141	10,70	217,11	209,47	3,51
18				32	2553	2832	10,90	143,82	140,04	2,69

† – code SEC/DED, ‡ – code de parité, sinon : code Hamming cyclique SEC

Pour une RAM protégée par un code SEC (correspondant aux configurations 7, 10 et 13), le surcoût matériel de la version FS est relativement élevé, compris entre 36 et 59% pour un parallélisme de 8bits et des degrés de polynômes générateurs ECC assez importants (voir table 4.3). Le surcoût matériel relatif diminue clairement avec le niveau de parallélisme p, lorsque la RAM est protégée par un code SEC/DED. Pour $p = 16$, la complexité diminue jusqu'à 35% et 10% respectivement

pour les configurations 14 et 17. Pour $p = 32$, la complexité est d'environ 23% et 35% pour les configurations 15 et 18, elle se compare favorablement avec la duplication pour laquelle la complexité dépasse évidemment 100% de surcoût matériel.

Pour la plupart des configurations énumérées dans le tableau 4.6, la dégradation de la fréquence est inférieure à 37% avec des exceptions correspondant aux configurations 10 et 13 pour laquelles la dégradation est d'environ 48%.

TAB. 4.7 – Comparatif (surface et vitesse) du codeur FS et S lorsque $G(x)$ n'est pas multiple de $G'(x)$

Numéro de config.	Polynômes			p	No. CLBs		Surcoût matériel [%]	$f_{\max}$ [MHz]		Réduction de fréq.[%]
	$G(x)$	$G'(x)$	$W(x)$		A_S	A_{FS}		f_S	f_{FS}	
19	$G_1(x)$	$G'_3(x)$	$W_3(x)$	8	32	63	96,87	500	281,06	43,78
20		$G'_6(x)$ †	$W_6(x)$	16	36	82	127,77	423,73	244,14	42,38
21		$G'_{11}(x)$ †	$W_4(x)$	32	45	108	140,00	372,02	257,73	30,72
22	$G_2(x)$	$G'_2(x)$	$W_2(x)$	8	52	88	69,23	500	353,61	29,27
23		$G'_4(x)$ †	$W_5(x)$	16	65	116	78,46	489	244,68	49,96
24		$G'_{11}(x)$ †	$W_7(x)$	32	96	167	73,95	282,81	250	11,60
25	$G_4(x)$	$G'_5(x)$ †	$W_1(x)$	8	69	117	69,56	500	290,11	41,97
26		$G'_5(x)$ †		16	85	138	62,35	477,78	273,37	42,78
27		$G'_{12}(x)$ †		32	128	206	60,93	292,14	243,37	16,69
28	$G_5(x)$	$G'_2(x)$	$W_2(x)$	8	134	185	**38,00**	494,56	248,14	49,82
29		$G'_7(x)$ †	$W_3(x)$	16	169	236	**39,64**	277,32	219,73	20,76
30		$G'_9(x)$	$W_8(x)$	32	249	349	**40,16**	229,15	184,6	19,40

† – code SEC/DED, ‡ – code de parité, sinon : code Hamming cyclique SEC

Dans le tableau 4.7, on compare les versions simples et FS de différents codeurs pour lequels le polynôme $G(x)$ n'est pas multiple de $G'(x)$. Rappelons que nous considérons ici les polynômes CRC standards $G_1(x)$, $G_2(x)$, $G_4(x)$ et $G_5(x)$, répertoriés dans le tableau 4.2. Les polynômes $W(x)$ sont les restes de la division des polynômes $G(x)$ par les polynôme $G'(x)$. Ils sont listés dans le tableau 4.8. On peut remarquer que le surcoût matériel relatif au codeur FS diminue lorsque le poids minimal de Hamming $W(x)$ diminue. En particulier, pour deux polynômes $W_i(x)$ et $W_j(x)$ de même poids de Hamming le codeur tend à consommer moins de surface quand les bits sont uniformément répartis. Dans la pratique, nous avons sélectionné pour chaque degré de parallélisation p uniquement les polynômes $G'_i(x)$ conduisant à un poids de Hamming minimal pour $W(x)_{min}$. Rappelons que les polynômes $G_i(x)$ figurant dans le tableau 4.7 sont classés par ordre croissant des degrés respectivement 16, 24, 32 et 64.

Quand le polynôme $G(x)$ n'est pas multiple de $G'(x)$, nous pouvons encore identifier des cas que l'on peut considerer comme "favorables" et "acceptables" et correspondant à des surcoûts matériels d'environ 40% et de 60 à 70%, notés respectivement dans le tableau 4.7 en gras et en souligné. Le surcoût relatif diminue pour la version FS, lorsque le degré de $G_i(x)$ augmente. Pour $G_1(x)$, dont le degré est 16, le surcoût est très élevé (près de 100%, voir plus), non compétitif par rapport à la

TAB. 4.8 – L'ensemble des polynômes restes $W(x)$ résultant de la division de $G(x)$ par $G'(x)$

$W_0(x)$	0	$W_5(x)$	$1+x+x^2+x^3$
$W_1(x)$	x^2	$W_6(x)$	$1+x^4$
$W_2(x)$	$1+x^2$	$W_7(x)$	$1+x^5$
$W_3(x)$	$x+x^2$	$W_8(x)$	x^3+x^5
$W_4(x)$	$1+x^3$		

duplication. Pour les polynômes $G_2(x)$ et $G_4(x)$, de degrés 24 et 32, le surcoût est de 60 à 70%. Le cas le plus favorable est celui du polynôme $G_5(x)$, le surcoût étant seulement de 40%. On peut observer que le degré de ce polynôme est élevé (voir 64), et le poids de Hamming du polynôme $W(x)$ correspondant est faible ($= 2$). On peut également remarquer que pour un même polynôme $G(x)$, la consommation en surface (en nombre de cellules logiques CLB) est presque proportionnel au niveau du parallélisme p de l'architecture.

Le surcoût matériel est relativement faible (moins de 50%) pour toutes les configurations du tableau 4.6 (à l'exception des deux cas indiqués en gras). Par contre, il est relativement élevé (plus de 50%) pour la plupart des configurations du tableau 4.7. Ceci est une confirmation quantitative de notre conjecture (voir section 1.3.1) qu'un surcoût materiel minimal peut être obtenu lorsque le polynôme $G(x)$ est multiple du polynôme $G'(x)$.

Pour les configurations du tableau 4.7, la dégradation de la fréquence est inférieure à 30% pour $p = 32$, est comprise entre 20 et 49% lorsque $p = 8$ ou 16. Pour toutes combinaisons des paramètres de conception (k, m, n, p), le temps de traitement est toujours égal à $l + 1$ cycles d'horloge, où l représente le nombre de mots de RAM du bloc à transmettre. Autrement dit, les résultats obtenus montrent qu'un niveau de parallélisme p important peut être réalisé avec une diminution raisonnable de la fréquence maximale d'horloge.

La fréquence maximale de fonctionnement des codeurs FS dépend du chemin critique situé quelque part entre la RAM et le comparateur du codeur (voir figure 4.5). Les chemins principaux sont ceux traversant les blocs (A) et (B) d'une part, et le bloc (C) d'autres part. La longueur de ces chemins est en rapport avec les degrés $(n - k)$ et $(m - p)$ des polynômes $G(x)$ et $G'(x)$, respectivement. C'est pourquoi nous allons nous concentrer uniquement sur les chemins traversant les blocs (A) et (B). Ces chemins déterminent le chemin critique du codeur FS.

Pour les configurations des tableaux 4.6 et 4.7 touchées par une dégradation de fréquence importante, une technique de pipeline est proposée dans la suite. Elle a pour but de minimiser le chemin critique de façon à maintenir une fréquence de fonctionnement du codeur dans sa version FS.

Notez d'abord qu'à l'intérieur du bloc (A), deux chemins de longueurs différentes peuvent être identifiés impliquant les deux fonctions $F_1(R_i(x)) = (R_i(x) \cdot x^p) \bmod G(x)$ et $F_2(Y_i(x)) = (Y_i(x) \cdot x^{n-k}) \bmod G(x)$. Si $n-k > p$ (respectivement $p > n-k$), alors le chemin critique se trouve à priori dans le bloc F_1 qui a $(n-k)$ bits d'entrée (respectivement F_2 qui a p bits d'entrée). Cela dépend donc du degré $(n - k)$, du nombre de termes non nuls et de leurs positions dans le polynôme choisi $G(x)$, ainsi que du niveau de parallélisation p (dépendance non linéaire). Le chemin le plus long de retard du bloc (B) étant le même dans les deux cas.

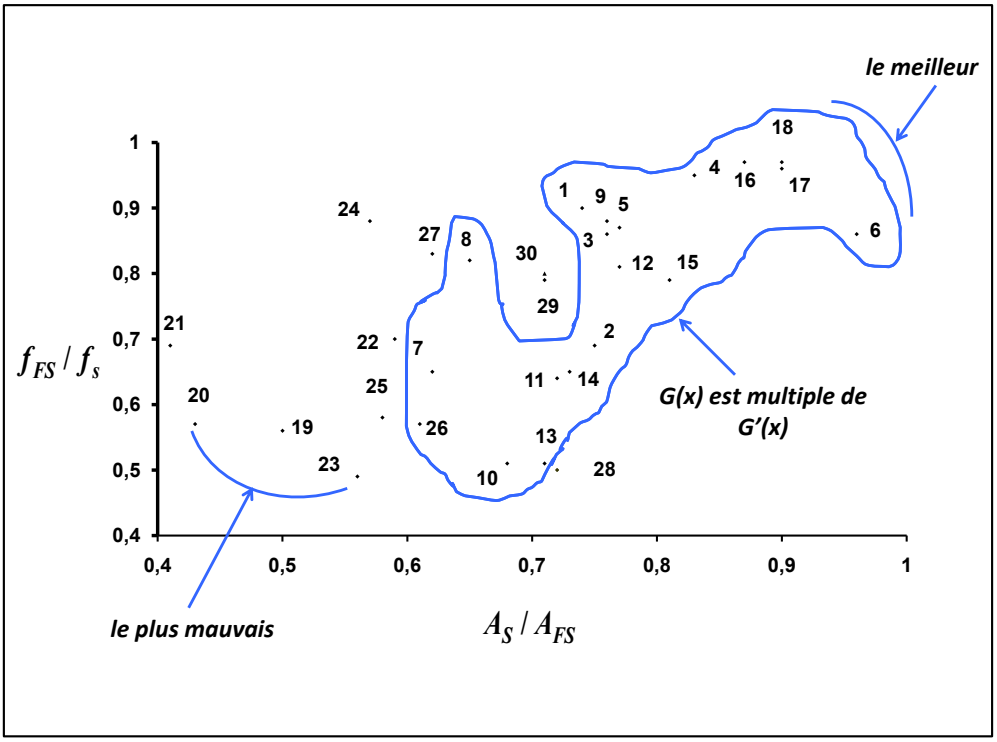


FIG. 4.6 – Compromis surface/fréquence de codeurs S et FS pour les configurations des tableaux 4.6 et 4.7

La figure 4.6 propose pour toutes les codeurs FS une vue globale des performances exprimées sous forme de compromis surface/fréquence. Le rapport A_S/A_{FS} représente le surcoût matériel du codeur FS par rapport à sa version simple S, alors que le rapport f_{FS}/f_S mesure la dégradation de la fréquence d'horloge entre les codeur FS et S. Il est clair que $A_S/A_{FS} < 1$, le surcoût minimal de matériel correspondant au A_S/A_{FS} le plus proche possible de 1. Parmi toutes les configurations, la configuration 6 est la meilleure ($A_S/A_{FS} = 0,96$). La raison en est le parallélisme $p = 32$ élevé, le fait que le polynôme $G'(x)$ correspondant soit multiple du polynôme $G(x)$ (et par conséquent le bloc générant le polynôme $R^*(x)$ n'existe plus) et que le degré de $G'(x)$ (1) soit très inferieur à celui degré de $G(x)$ (32). On peut observer la même chose pour les configurations 16, 17 et 18, le surcoût devenant de plus en plus important avec la diminution du degré de $G(x)$ d'une part et du parallélisme p d'autre part (voir les configurations $7-15$). Les configurations encerclées dans la partie droite de la figure 4.6 sont celles du tableau 4.6 pour lequelles le polynôme $G(x)$ est multiple du polynôme $G'(x)$. Les autres sont des configurations listées dans le tableau 4.7, pour lequel les $G(x)$ n'est pas multiple de $G'(x)$. La plupart des configurations dans la zone entourée présentent de meilleurs compromis (vitesse, surface) en étant globalement celles qui induisent les plus faibles surcoûts matériels. Pour les autres configurations, la plus faible complexité du codeur FS peut être obtenue en sélectionnant proprement les polynômes $G'(x)$ et $G(x)$, de façon a minimiser le poids du polynôme $W(x)$ résultant. Les cas les plus favorables correspondent aux configurations 28, 29, et 30.

Concernant la fréquence d'horloge, aucun cas $f_{FS} > f_S$ n'ayant été observé, on peut supposer que

$f_{FS}/f_S = 1$ une la valeur maximale correspondant à une implantation FS n'induisant pas de dégradation de la fréquence du fonctionnement. La fréquence maximale du codeur dépend du chemin critique de blocs générant les polynômes $R(x)$, $R'(x)$, $R^*(x)$ (voir ci-dessous). Les configurations 16, 17, et 18 (correspondant aux codes RS) du tableau 4.6 présentent les meilleurs compromis vitesse/surface, la dégradation de fréquence étant comprise entre 2 et 3,5%.

En prenant la surface comme critère de comparaison, nous pouvons constater que l'architecture FS proposée occupe beaucoup moins de surface que la méthode de la duplication. Elle offre de plus un débit comparable au codeur simple dans certaines configurations, et une dégradation acceptable de la fréquence du fonctionnement du codeur FS parallèle. Cette dégradation devient cependant importante pour certaines configurations. Afin d'améliorer la performance des codeurs FS dans ces cas défavorables, une approche parallèle-pipeline peut permettre d'augmenter le nombre de bits traités à chaque cycle d'horloge sans trop augmenter le chemin critique, c'est-à-dire en maintenant la fréquence de fonctionnement la plus élevés possible. Une présentation de cette approche sera détaillée dans la prochaine section suivie d'une discussion sur les résultats obtenus lors de l'implantation .

4.6 Implantation de codeurs FS parallèles-pipeline

Comme on l'a vu dans la section précédente, le surcoût matériel du codeur FS proposé est très inférieur à celui obtenu par la méthode de duplication. Malgré ce succès de minimisation de la surface, la fréquence du codeur FS se dégrade rapidement avec le degré de parallélisme p pour certains configurations, limitant ainsi les bénéfices du parallélisme. L'amélioration de la performance est effectuée ici à l'aide des techniques de pipeline, permettant de conserver une fréquence du fonctionnement la plus élevée possible. Cette approche est détaillée dans cette section.

Pour pouvoir appliquer des techniques de pipeline à notre codeur, plusieurs étapes préparatoires sont nécessaires. Dans la première étape on va rendre la formulation des équations calculant $R(x)$,$R'(x)$, et $R''(x)$ plus adaptée au pipeline en adoptant la forme matricielle. Ensuite, nous appliquerons la technique de pipeline sur ces matrices par "insertion" de registres entre les lignes de ces matrices.

4.6.1 Principe de réalisation d'un codeur FS parallèle-pipeline

Les équations polynômiales qui régissent les différentes parties du codeur $R(x)$, $R'(x)$, et $R''(x)$ sont définies comme suit (voir section 4.4.1). Pour $R(x)$:

$$\begin{aligned} R_{i+1}(x) &= (R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}) \bmod G(x) \\ &= (R_i(x) \cdot x^p) \bmod G(x) + (Y_i(x) \cdot x^{n-k}) \bmod G(x) \end{aligned} \tag{4.36}$$

L'équation de $R'(x)$ est :

$$R'_{i+1}(x) = [R'_i(x) \cdot x^p] \bmod G'(x) + Z_i(x). \tag{4.37}$$

Pour $R^*(x)$ on a :

$$\begin{aligned} R^*_{i+1}(x) =& [(R^*_i(x) \cdot x^p) \bmod G'(x) + (R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}) \bmod G(x) \\ & + R_i(x) \cdot x^p + Y_i(x) \cdot x^{n-k}] \bmod G'(x) \\ =& (R^*_i(x) \cdot x^p) \bmod G'(x) \\ & + [(R_i(x) \cdot x^p) \bmod G(x) + (Y_i(x) \cdot x^{n-k}) \bmod G(x)] \bmod G'(x) \\ & + (R_i(x) \cdot x^p) \bmod G'(x) + (Y_i(x) \cdot x^{n-k}) \bmod G'(x) \end{aligned} \tag{4.38}$$

De manière générale, la fonction *modulo* correspond à un diviseur. Prenons un exemple : $C(x) = A(x) \bmod B(x)$, le reste de la division de $A(x)$ par $B(x)$. Cette équation peut se transformer en une équation matricielle comme suit [53, 57] :

$$[C] = [A] \cdot [D] \tag{4.39}$$

où $[A]$ et $[C]$ sont des vecteurs contenant les valeurs des coefficients de A et C, respectivement a_i et c_i. Par contre, $[D]$ est une matrice (m lignes, n colonnes) dont m et n dépendent respectivement des largeurs des vecteurs $[C]$ et $[A]$. Les coefficients de ce matrice sont indépendantes des coefficients des vecteurs $[A]$ et $[C]$. Ils dépendent uniquement de la fonction $\bmod B(x)$, et donc des coefficients du vecteur $[B]$.

En appliquant ce changement aux équations précédentes, les nouvelles équations matricielles deviennent :

$$\begin{aligned} R_{i+1}(x) &= R_i(x) \cdot T^p + Y_i(x) \cdot M \\ R'_{i+1}(x) &= R'_i(x) \cdot T'^p + Z_i(x) \\ R^*_{i+1}(x) &= R^*_i(x) \cdot T^p + R_i(x) \cdot H + Y_i(x) \cdot M' \end{aligned} \tag{4.40}$$

Notons que l'équation de comparaison entre $R'(x)$ et $R''(x)$ à chaque cycle d'horloge :

$$R^*_i(x) + R_i(x) \bmod G'(x) = [R'_i(x) \cdot x^{n-k-(m-p)}] \bmod G'(x), \;\; 1 \leq i \leq l \tag{4.41}$$

devient

$$R^*_i(x) + R_i(x) \cdot W = R'_i(x) \cdot L \tag{4.42}$$

où les dimensions respectives de ces matrices sont définies dans le tableau 4.9 :

La figure 4.7 montre le nouveau schéma FS parallèle-pipeline obtenu à partir de la forme matricielle. Le temps totale de traitement du codeur FS dans cette figure est égale à $p + (n - k) + 3$, où p est le degré de parallélisme et $n - k$ le degré du polynôme générateur $G(x)$. Chaque matrice signalée dans la figure est implantée selon une structure pipeline dont la correspondance est expliqué à l'aide

TAB. 4.9 – Dimensions des différentes matrices

Matrice	Nb. de lignes	Nb. de colonnes
T^p	$n-k$	$n-k$
T'^p	$m-p$	$m-p$
M	p	$n-k$
M'	p	$m-p$
H	$n-k$	$m-p$
W	$n-k$	$m-p$
L	$n-k$	$m-p$

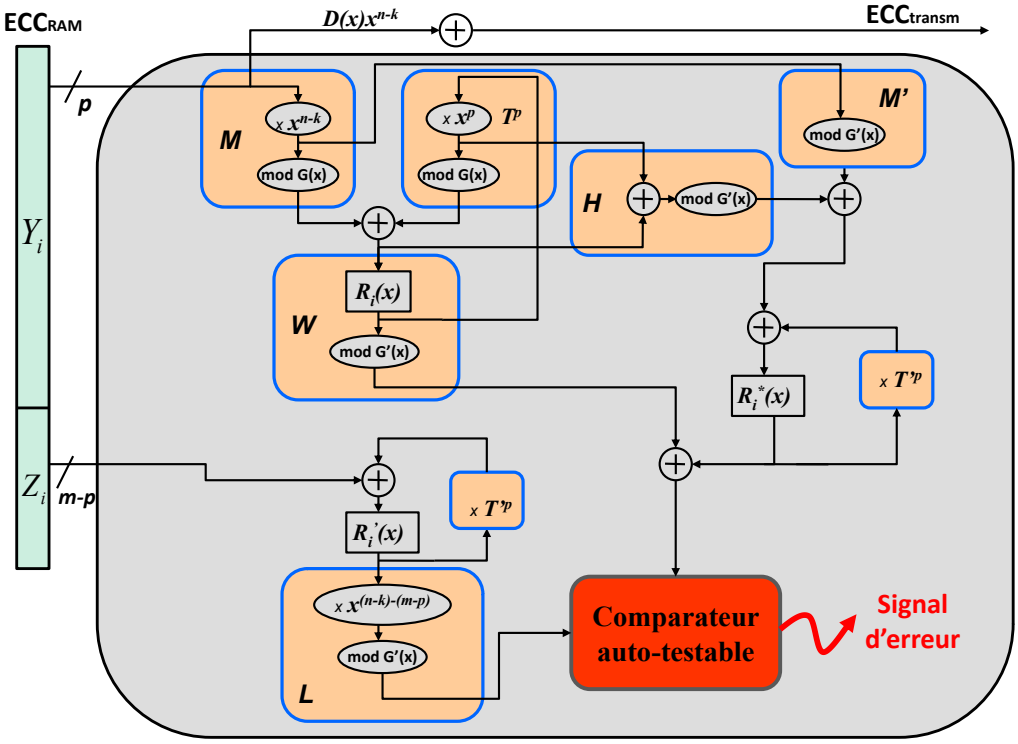


FIG. 4.7 – Schéma bloc d'un codeur FS (*Fault-Secure*) parallèle-pipeline

de l'exemple suivant. Soit $S(x)$ le polynôme résultant de l'équation :

$$S_i(x) = (Y_i(x) \cdot x^{n-k}) \bmod G(x) \tag{4.43}$$

S et Y sont de degré p et $n-k$ (degré de $G(x)$) et p respectivement. La transformation matricielle correspondante donc :

$$[s_0, \cdots, s_{n-k-2}, s_{n-k-1}] = [y_0, \cdots, y_{p-2}, y_{p-1}] \cdot \begin{pmatrix} m_{00} & m_{01} & \cdot & m_{0(n-k-1)} \\ m_{10} & m_{11} & \cdot & m_{1(n-k-1)} \\ \cdot & \cdot & \cdot & \cdot \\ m_{(p-1)0} & m_{(p-1)1} & \cdot & m_{(p-1)(n-k-1)} \end{pmatrix} \tag{4.44}$$

où s_i, y_i, et m_{ij} sont les coefficients de S, Y et M (p lignes, $(n-k)$ colonnes) respectivement. La matrice M est pipelinée verticalement par l'insertion de registres entre ses lignes.

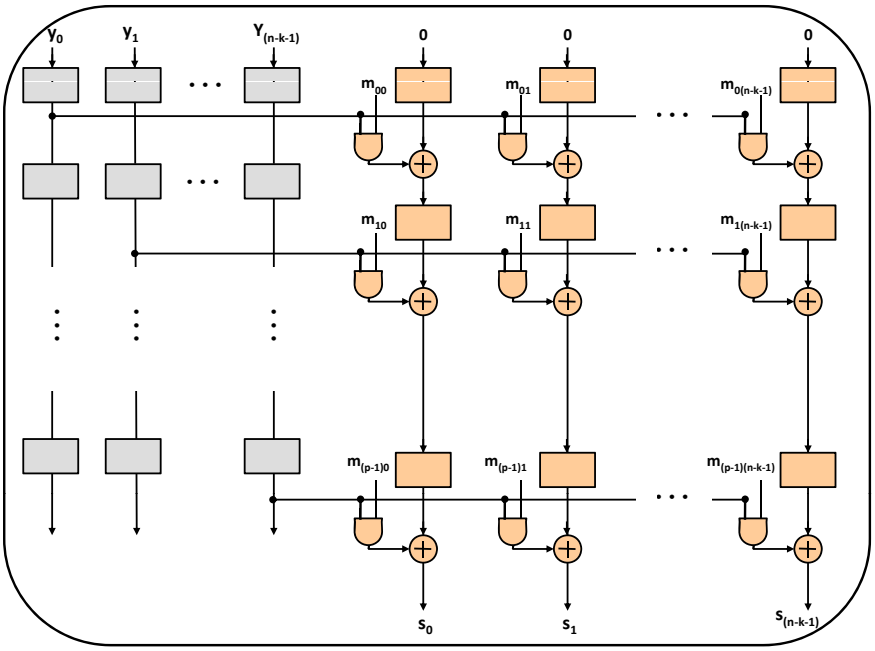


FIG. 4.8 – Structure pipeline pour la matrice M

La figure 4.8 montre la structure du pipeline correspondante, où les coefficients y_i sont les bits de données provenant de la mémoire RAM, les m_{ij} sont les coefficients de la matrice M, et les s_i sont les coefficients de S. La latence totale de ce bloc est égale à p, où p correspond au nombre de lignes de la matrice M. À chaque cycle d'horloge $p$bits de données sont traités. Toutes les autres matrices M', H, W, et L de la figure 4.7 sont pipelinés de la même façon. Les matrices T^p et T'^p étant à l'intérieur d'une boucle, elles ne peuvent être pipelinés. La latence de la détection d'erreurs du codeur FS parallèle-pipeline dépend également du nombre d'étage de pipeline pour chaque matrice, le temps totale du traitement du circuit étant égale au nombre total d'étages de pipeline dans le codeur (entre l'entré des données lues à partir de la RAM jusqu'au comparateur auto-testable).

4.6.2 Résultats expérimentaux

Les codeurs FS parallèles-pipeline précédents ont été synthétisés sur FPGA de la famille StratixII d'Altera pour les mêmes configurations des tableaux 4.6 et 4.7 en utilisant le logiciel Altera/Quartus II. Les résultats concernant la performance et la complexité ont été regroupés dans les figures 4.9 et 4.10, respectivement.

La figure 4.9 compare les débits de fonctionnement entre les versions FS et FS pipeline du codeurs. La figure 4.10 compare la complexité exprimée en termes de CLBs du FPGA, entre les versions simple-pipeline-dupliqué (voir figure 4.10 Sp (duplication)) et FS-pipeline (FSp). Le but de cette

figure est de comparer, pour une telle configuration, la consommation en surface du codeur FSp par rapport à celle résultant de la duplication d'un codeur simple dupliqué Sp, notre objectif étant de concevoir des codeurs FS rapides dont le surcoût matériel reste inferieur à celui du codeur dupliqué.

Rappelons que $G_1(X)$ à $G_5(X)$ sont des polynômes CRC, $G_6(X)$ à $G_8(X)$ des polynômes BCH et $G_9(X)$ un polynôme RS, le niveau de parallélisme p étant égal à la largeur des données dans la mémoire RAM.

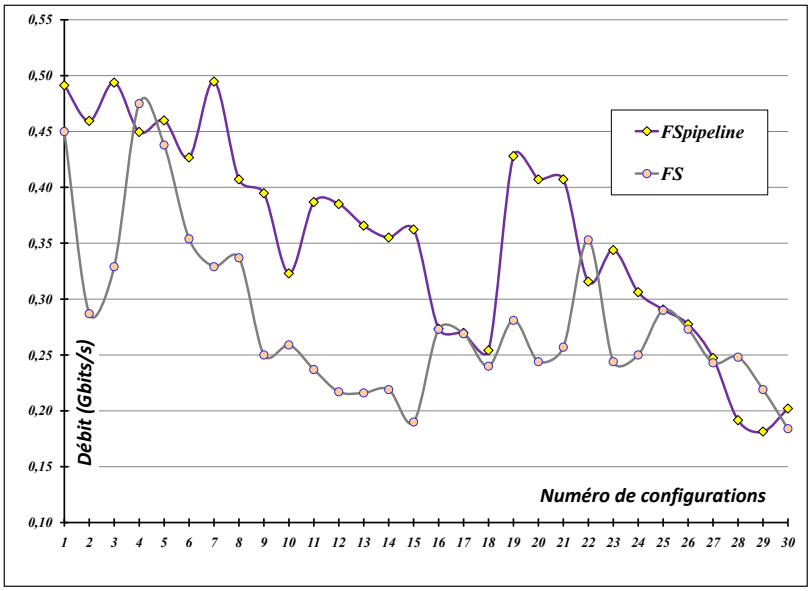


FIG. 4.9 – Comparatif des performances (débit) entre les codeurs FS et FS pipeline pour les configurations de tableaux 4.6 et 4.7

D'après les résultats de la figure 4.9, on remarque que, dans certains cas la performance du codeur est améliorée d'une manière significative (voir les configurations 6–15). Dans d'autre cas, comme par exemple pour le code RS, la performance avec le pipeline reste identique (voir configurations 16–18).

Bien que la performance du codeur FS soit améliorée dans différentes configurations, la surface occupée correspondante tend à exploser linéairement avec le degré du polynôme $G(x)$ (voire plus dans certains cas). En pratique dans la plupart des cas l'amélioration de la performance ne compense pas l'important surcoût matériel. Les résultats de la figure 4.10 montrent en effet que dans plusieurs cas, le surcoût matériel pour le codeur FS pipeline est beaucoup plus important que celui du codeur pipeline obtenu par duplication (simple pipeline dupliqué Sp (duplication)). Le pipeline a donc dans ces cas un effet négatif sur le codeur FS. Les cas les plus défavorables sont ceux des configurations 16, 17 et 18, correspondant au code RS, dont le degré du polynôme générateur est très élevé (voir 112).

On remarque que le pipeline est favorable quand uniquement le degré p est supérieur à $(n-k)$, (voir les configurations 3, 6, 9, 21, et 24). L'explication de cet effet tient au fait que dans ce cas, le chemin critique du codeur FS se situe uniquement dans la matrice M, de degré p. En pipelinant uniquement cette matrice, une amélioration significative de la performance est obtenue avec un surcoût

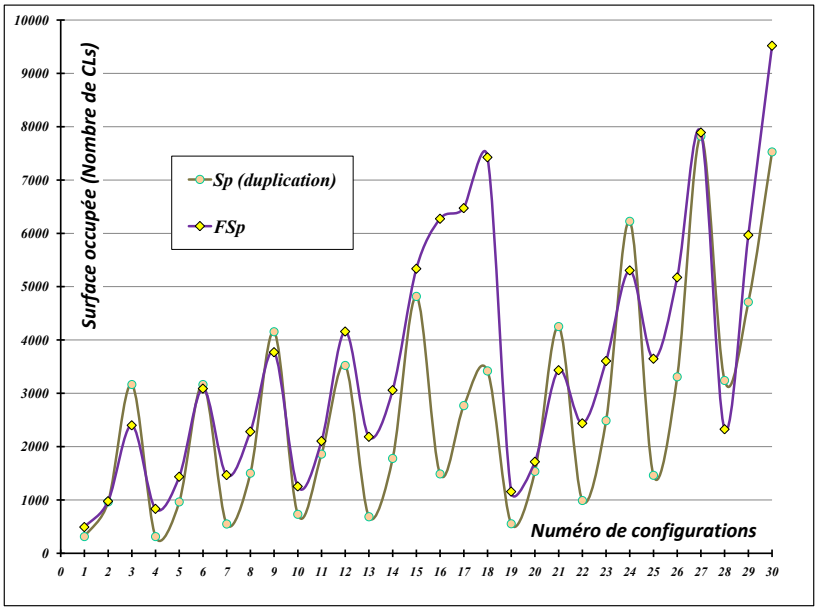


FIG. 4.10 – Comparatif du surcoût matériel entre les codeurs FS pipeline (FSp) et simple-pipeline dupliqué (Sp (duplication)) pour les mêmes configurations

matériel acceptable. Pour toutes les autres matrices (M', H, W, et L), le gain en performance ne justifie pas la quantité de matériel nécessaire pour le pipeline. Par conséquence, le pipeline n'est pas favorable lorsque le degré $n - k$ est supérieur à p.

TAB. 4.10 – Effet du pipeline sur le codeur FS

	Polynômes générateurs G(x)							
	CRC				**BCH**			**RS**
Parallélisme	16	24	32	64	24	49	16	
8	−	−	−	−	−	−	−	−
16	+	−	−	−	−	−	−	−
32	+	+	+	−	+	−	−	−

Le tableau 4.10 résume l'effet du pipeline sur le codeur FS. Il est positif (+) lorsque le degré de parallélisme p est inférieur au degré du polynôme générateur $G(x)$. Les quelques cas favorables (+) correspondant à ces conditions correspondent aux codes CRC-16 pour $p = 16$ et 32 ; ainsi qu'aux codes CRC-24, CRC-32 et BCH-24 pour $p = 32$. L'effet du pipeline est défavorable (−) dans touts les autres cas.

4.7 Décodeurs FS parallèle-pipeline

Dans cette section, nous considérons le problème de la conception de décodeurs cycliques sûrs en présences de fautes FS. Comme cela a été défini dans la section 1.4.2, un décodeur cyclique est constitué d'un codeur, d'un registre de syndrome à partir du quel une fonction de décision EPD (*Error*

Pattern Detection) ou MLV (*Majority Logic Voting*) décide de la validité du bit testé et de quelques additionneurs modulo-2. Le principal avantage de ce type de décodeur réside dans la mise en œuvre simple et dans la rapidité de décodage. Ceci le rend bien adapté non seulement aux applications haut débit mais aussi à la SdF.

Nous proposons ici une architecture FS du décodeur parallèle-pipeline. La SdF du décodeur repose sur une légère modification de la structure parallèle-pipeline de façon à pouvoir contrôler le fonctionnement du codeur parallèle-pipeline à chaque cycle d'horloge avec un surcoût matériel qui reste très inférieur à celui de la duplication.

4.7.1 Architecture d'un décodeur FS parallèle-pipeline

Dans la figure 4.11, nous proposons une version FS du décodeur parallèle-pipeline présenté dans la section 1.4.2 ([51]). Nous allons tout d'abord décrire son principe de fonctionnement, montrer ensuite qu'il présente bien le comportement FS. Le principe général nous permet de contrôler correctement son fonctionnement avec une légère modification de l'architecture parallèle-pipeline du décodeur cyclique en bloc proposé dans [51].

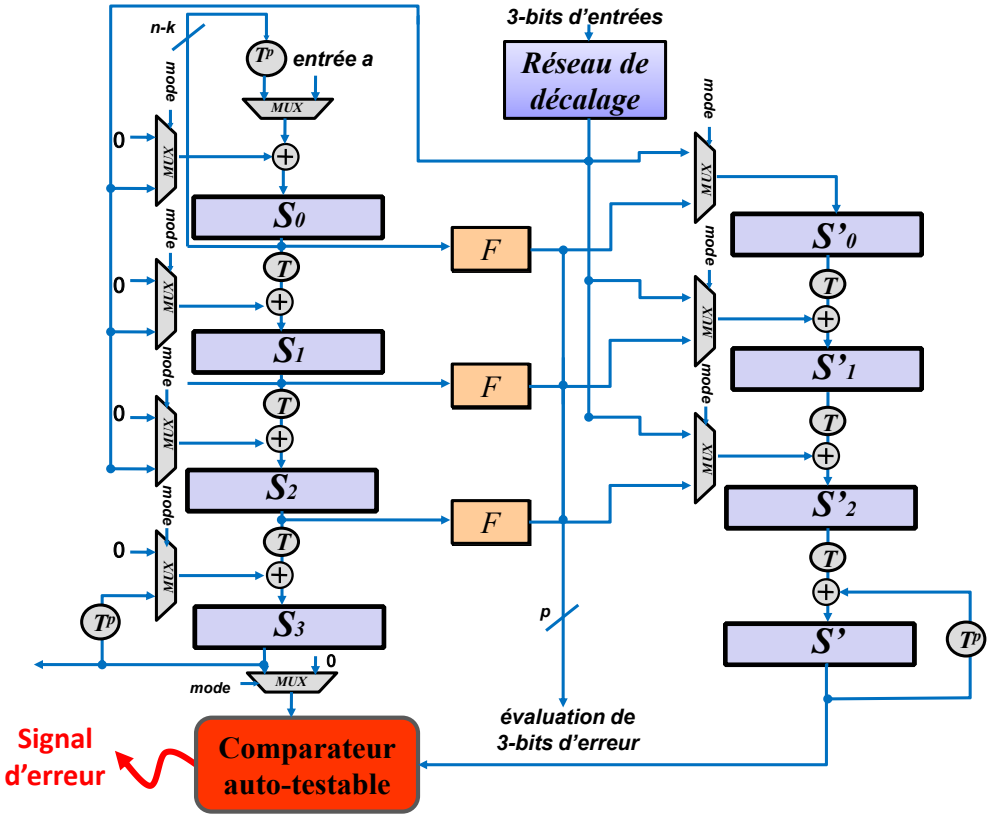


FIG. 4.11 – Architecture FS parallèle-pipeline du décodeur cyclique pour $p = 3$

Le fonctionnement du décodeur FS parallèle-pipeline peut être décrit comme suit :

– le syndrome du mot code reçu est calculé simultanément dans les parties droite et gauche du décodeur FS parallèle-pipeline ;

– a chaque cycle d'horloge pendant calcul du syndrome, les registres S_i et S'_i doivent être égaux. Les équations régissant l'évolution des registres S'_i (et de manière analogue S_i) sont :

$$\begin{aligned}(S'_i)_{t+1} &= (S'_{i-1})_t \cdot T + d_i, \quad S'_0 = d_0 \\ (S')_{t+1} &= (S')_t \cdot T^p + (S'_{p-1})_t;\end{aligned} \tag{4.45}$$

– dès que le syndrome a été calculé, la valeur en S' est transférée dans S_0 grâce à la connexion *entrée a* ;

– a chaque cycle d'horloge, une p-permutation cyclique est appliquée au registre S_0. L'erreur estimée est injectée dans les étages du pipeline S'_i. Par conséquent, $p$bits d'entrée peuvent être corrigés par les résultats de p F. Les équations correspondantes sont les suivantes :

$$\begin{aligned}(S_0)_{t+1} &= (S_0)_t \cdot T^p \\ (S_i)_{t+1} &= (S_{i-1})_t \cdot T, \quad i \in 1, \cdots, p-1 \\ (S)_{t+1} &= (S_{p-1})_t\end{aligned} \tag{4.46}$$

Alors pour S' on a :

$$(S'_i)_{t+1} = (S'_{i-1})_t \cdot T + F[(S_i)_t] \cdot (1, 0, 0) \tag{4.47}$$

– F est une fonction de décision utilisée pour l'estimation de l'erreur, cela peut être une fonction de détection de motif d'erreur EPD (*Error Patten Detection*) où une fonction à logique majoritaire MLV (*Majority Logic Voting*) [6] ;

– un multiplexeur est utilisé dans la partie droite du décodeur pour sélectionner la bonne entrée selon le mode de fonctionnement (calcule de syndrome ou décodage).

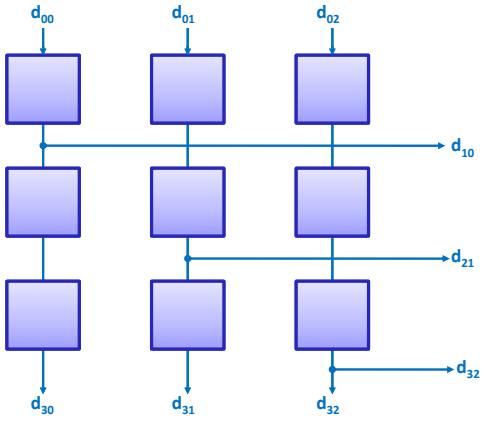


FIG. 4.12 – Réseau de décalage pour 3-bits d'entées

La figure 4.12 montre une structure pipeline de données pour $p = 3$. Il est à noter que ce réseau de décalage est protégé par un code de parité, sinon, toute erreur se produisant dans ce réseau sera vue comme une erreur dans la chaine de transmission.

L'architecture présentée dans la figure 4.11 est une version sécurisée d'un décodeur cyclique

parallèle-pipeline. Elle permet de détecter les fautes temporaires survenant dans la la phase de calcul du syndrome et celle de décodage. Une analyse fonctionnelle de la technique de détection d'erreur de la figure 4.11 peut être décrite comme suit :

un comparateur auto-testable est utilisé pour comparer les valeurs des registres S et S'. A la fin de la phase de calcul du syndrome, ces valeurs doivent être égales puisque le mot code est injecté dans les parties droites et gauche du décodeur. Par conséquent, toute faute temporaire dans S ou S' est détectée par le comparateur. À la fin de la phase de décodage, la valeur de S' doit être égale à zéro puisque l'erreur est injectée deux fois : une fois dans le mode de calcul de syndrome et une deuxième fois dans le mode du décodage à travers de la fonction F (voir le multiplexeur connecté au comparateur dans la figure 4.11). Donc toute erreur induite pendant la phase de décodage est aussi détectée, car elle est contrôlée par le comparateur auto-testable.

Considérons le cas pour exemple d'une erreur temporaire simple affectant le décodeur FS. Supposons qu'une faute simple se produit quelque part dans le registre de syndrome S_i. Cela produit, après quelques cycles d'horloge, $S^e \neq S'$ dans le mode calcul de syndrome, ou $S' \neq 0$ à la fin du mode du décodage. Le comparateur va alors signaler le désaccord entre S^e et le signal correct produit par la partie de contrôle S'. Le même raisonnement s'applique aussi pour une faute se produisant dans la partie de contrôle S', qui conduire à $S'^e \neq S'$. Par conséquent, le désaccord entre $S \neq S'^e$ sera signalé par le comparateur. Toutes les fautes simples sont donc détectées par le comparateur, celui-ci étant auto-testable.

Concernant la latence de detection d'erreurs de cette architecture, elle est au plus égale à p (lorsque le registre S_0 est affecté) dans le mode calcul de syndrome, et égale au plus à n/p dans le mode décodage.

Concernant l'aspect architectural, le surcoût matériel et la fréquence maximale de fonctionnement du décodeur FS seront évalués dans la prochaine section.

4.7.2 Résultats expérimentaux

Nous présentons ici les paramètres des deux versions du décodeurs parallèle-pipeline : simple (S) et sûr en présences de fautes FS. Ces deux versions conformes aux figures 4.11 et 1.22 respectivement, ont été synthétisées en utilisant le logiciel Altera/Quartus ciblant des FPGA de la famille StratixII d'Altera. Différents codes cycliques DSCC (*Different Set Cyclic Codes*), BCH, CC (*Cyclic Codes*), et MLC (*Maximum Length Codes*) ont été considérés. Les résultats de performance (débit) et de complexité (en termes de CLBs) ont été regroupés dans le tableau 4.12. Notons que la fréquence maximale de fonctionnement (f_{max}) de cette famille de FPGA est limitée à 500 MHz.

Les résultats du tableau 4.12 montrent que le surcoût matériel du décodeur FS parallèle-pipeline est relativement faible : moins de 32%. Une faible quantité de materiel est donc rajoutée au circuit pour la rendre FS, puisque l'architecture proposé reposant sur une légère modification de celle existante (voir figure 1.22). Cette approche est donc nettement meilleure que la duplication.

TAB. 4.11 – Codes simulés

code	(n, k)	Polynôme
DSCC	(21,11)	$1 + x^2 + x^4 + x^6 + x^7 + x^{10}$
BCH	(15,7)	$1 + x^4 + x^6 + x^7 + x^8$
MLC	(15,4)	$1 + x + x^2 + x^3 + x^5 + x^8$
CC	(7,4)	$1 + x + x^3$
Hamming	(31,26)	$1 + x^2 + x^5$

Le débit de fonctionnement est comparable pour les deux versions S et FS, celles-ci très semblables, puisque le chemin critique ne change pas radicalement étant toujours lié à la matrice T^p (situé dans la partie gauche du décodeur). Un débit de 7.03 Gbits/s est atteint pour la version FS avec un parallélisme $p = 31$. La latence de calcul du syndrome est proportionnelle au degré de parallélisme p, tandis que la latence de correction d'erreur est égale à $\lfloor n - k/p \rfloor$.

TAB. 4.12 – Complexité de décodeurs parallèle-pipeline simple et FS for les polynômes du tableau 4.11

code		Nu. CLBs		Surcoût	débit [Gbits/s]		Latence	
	p	Simple	FS	matériel [%]	Simple	FS	calcul de syndrome	correction d'erreur
DSC	3	74	93	25,68	0,70	0,70	3	7
	7	179	201	12,29	1,75	1,49	7	3
	11	338	356	5,33	2,39	2,48	11	2
	21	864	882	2,08	4,40	3,98	21	1
BCH	3	60	79	31,67	0,97	0,83	3	5
	5	106	122	15,09	1,40	1,22	5	3
	15	474	488	2,9	3,80	3,87	15	1
MLC	3	61	79	29,51	0,78	0,73	3	5
	5	104	122	17,31	1,09	1,15	5	3
	15	474	493	4,01	3,56	3,46	15	1
CC	4	48	56	16,67	1,42	1,30	4	2
	7	103	110	6,80	2,26	2,14	7	1
Hamm	31	1309	1315	0,46	7,67	7,03	31	1

La figure 4.13 montre un comparatif global entre les performances des décodeurs simple (S) et FS pour les configurations du tableau 4.12. En prenant la surface et le débit comme critères de comparaison entre les deux décodeurs FS et simple (S), nous pouvons constater que l'architecture FS proposée occupe beaucoup moins de surface (surcoût materiel < 32%) que la méthode de la duplication. Elle offre un débit presque identique à celui du décodeur simple pour toutes les configurations (voir tableau 4.12).

Les figures 4.14 et 4.15 montrent respectivement les signaux de simulation sans et avec injection d'erreur, pour le code CC du tableau 4.11 et un parallélisme $p = 4$. $C = (96)_h$ est le mot code à transmettre, et on suppose que le mot code reçu est $C' = (6D)_h$. Le comparateur signale après quelques cycles d'horloge le désaccord entre S et S'. Le signal du registre d'erreur $Reg - E$ qui doit

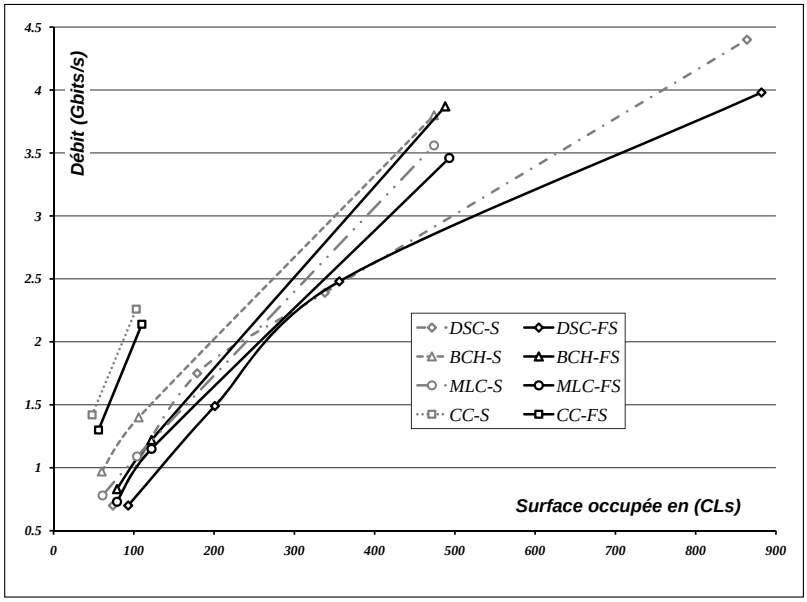


FIG. 4.13 – Comparatif compromis vitesse/surface du décodeur Simple et FS

TAB. 4.13 – Notation de simulation

Notation	Fonction
$d\text{-}in$	bits d'entrées
$d\text{-}out$	bits de sorties
$SEL\text{-}MOD$	mode de calcule de syndrome/décodage
$Reg\text{-}RS$	registre de syndrome S
$Reg\text{-}E$	registre d'erreur S'
RT	registre de test
$F\text{-}err$	évaluation d'erreur

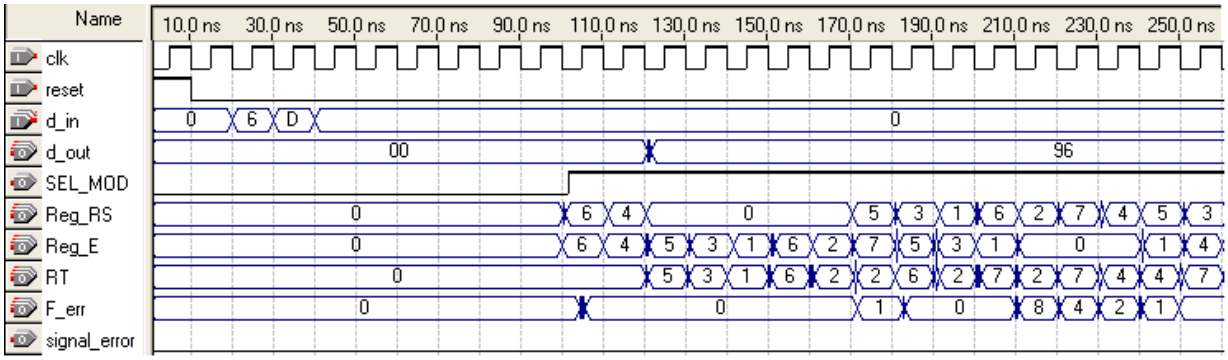


FIG. 4.14 – Décodeur cyclique parallèle-pipeline FS sans erreur

être égal à zéro à la fin du décodage (en absence d'erreur), est différent de zéro lorsque l'erreur est injectée (figure 4.15), et par conséquence, le comparateur auto-testable détecte cette divergence.

Vu les résultats obtenus précédemment, on peut conclure que l'architecture proposée offre bien un meilleur compromis surface/débit que la duplication. Seule une faible quantité de matériel étant

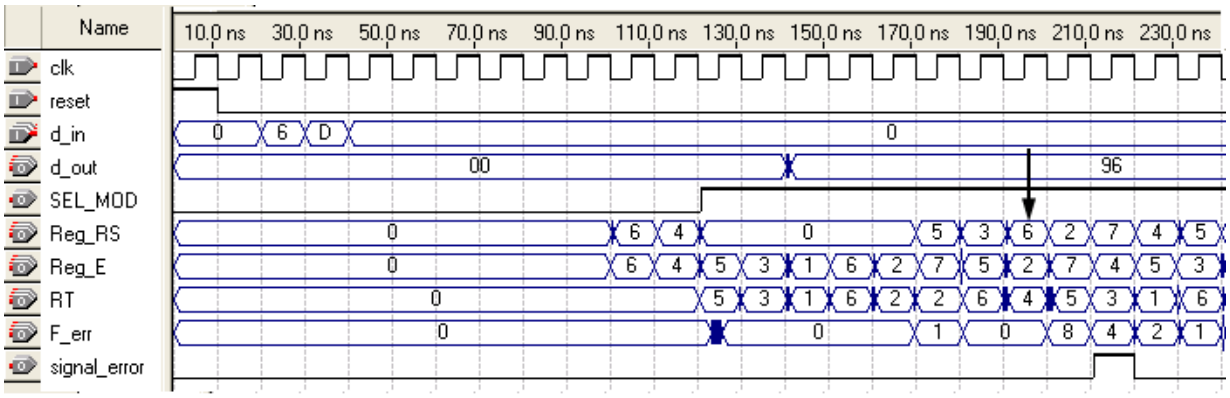


FIG. 4.15 – Décodeur cyclique parallèle-pipeline FS avec une erreur injectée

ajoutée à l'architecture pour la rendre sécurisée. Par conséquence, c'est une méthode efficace pour concevoir des décodeurs FS parallèle-pipeline. Concernant le débit du décodeur FS, il est toujours comparable à celui du codeur simple, une dégradation négligeable de la fréquence étant observée dans chacune des configurations du décodeur FS parallèle-pipeline.

4.8 Conclusion

Dans ce chapitre, nous avons présenté une méthodologie de conception d'un codeur FS parallèle placé entre une RAM tolérante aux fautes et un canal de transmission de données.

Contrairement aux précédentes, la nouvelle architecture possède deux chemins d'entrées indépendants pour tirer avantage de la disponibilité des bits de contrôle de la RAM et améliorer ainsi le niveau de tolérance aux fautes. Nous avons montré que la propriété cyclique de codes est déterminante pour la mise en œuvre des circuits parallèles. Par conséquent, seul les codes cycliques (codes de parité, codes de Hamming cycliques, codes CRC, codes BCH, et les codes RS) ont été considérées.

Différentes campagne d'injection de fautes ont été appliquées au codeur FS afin d'évaluer sa tolérance aux fautes. Les résultats obtenus s'avèrent très bons avec des taux de détection d'erreurs très élevés pour une injection de fautes simples à triples. La capacité de détection reste acceptable même pour des erreurs injectées de multiplicité élevée (de 6 à 9). Les performances et la complexité des codeurs FS et de leurs homologues Simples (S) ont été comparées après implantation sur FPGA. Les résultats obtenus montrent que le surcoût matériel requis pour les codeurs FS par rapport aux codeurs simples (S) est faible comparativement à celui induit par la duplication. Une approche pipeline à été proposée pour réduire la dégradation de la fréquence. Dans la pratique, seuls quelques cas permettent d'améliorer le compromis (vitesse, surface) avec le pipeline .

L'étude d'architectures tolérantes aux fautes a ensuite été étendue aux décodeurs parallèles-pipeline pour les codes cycliques en blocs. L'approche choisie repose sur une légère modification des architectures parallèles-pipeline déjà développées au laboratoire [51]. Les implantations sur FPGA démontrent l'efficacité de méthode proposée, le surcoût matériel requis pour la sécurisation étant faible et les performances temporelles étant globalement identiques.

CONCLUSION GÉNÉRALE

Conclusion générale

Les systèmes de transmissions numériques de données ont subit de nombreuses et rapides mutations ces dernières années : augmentation des débits, renforcement des contraintes de qualité de service, intégration sous forme de système embarqué, évolutivité et flexibilité, etc. Parallèlement, les technologies CMOS ont elles aussi beaucoup évolué, notamment avec la réduction des dimensions, l'augmentation des fréquences de fonctionnement et la réduction de niveaux d'alimentation. Il est aujourd'hui possible d'intégrer sur une seule puce des ensembles fonctionnels complexes, voire un système complet, présentant des caractéristiques (vitesse de traitement, surface et puissance consommées, faible coût) très intéressantes pour la réalisation de systèmes de communication embarqués à haut débit. Dans le même temps, ces mêmes paramètres technologiques conduisent à une augmentation importante de la sensibilité des circuits aux différentes sources de perturbation, notamment aux collisions de particules alpha provenant des radiations cosmiques, et donc à un accroissement des erreurs affectant les circuits. Ceci n'est pas acceptable alors même que les applications de transmission de données exigent des niveaux de qualité de service toujours plus élevés. Il devient donc nécessaire d'ajouter, dans le flot de conception, des contraintes de sûreté de fonctionnement (SdF) aux contraintes plus classiques de performance et de coût (débit de données, surface occupée). Il est donc nécessaire de mettre en œuvre de nouvelles approches architecturales. Dans le travail présenté dans ce mémoire, nous avons considéré les fautes temporaires comme cause de perturbation et avons choisi des techniques de tolérance aux fautes pour la mise en œuvre de la sûreté de fonctionnement (SdF).

Le travail présenté dans ce mémoire est consacré à l'étude architecturale au niveau RTL (*Register Transfer Level*) de (dé)codeurs SdF pour le (dé)codage canal dans des systèmes de communication embarqués. Les codes considérés sont les codes cycliques en blocs et les codes convolutifs récursifs. Les objectifs visés sont :

- rapidité et faible coût : les architectures doivent pouvoir traiter des volumes de données importants rapidement (contraites de débits de transmission élevés) en utilisant le moins de ressources matérielles possible, tout en s'accommodant de technologies peu coûteuse de type FPGA (*Field Programmable Gate Array*). Selon ce critère, les meilleures architectures sont celles qui présentent les rapports débit/surface les plus élevés.
- flexibilité : les architectures doivent pouvoir être configurés simplement en vue de leur synthèse et paramétrées en ligne facilement ;
- tolérance aux fautes : les architectures doivent être capable de fonctionner correctement malgré

la présence de fautes temporaires. Les modèles de fautes considérés sont les SEU (*Single Event Upset*) et MBU (*Multiple Bit Upset*).

Concernant les objectifs de rapidité et de faible coût, nous avons proposé des architectures avancées de codeur rapides (parallèles-pipeline) pour les code convolutif récursif. Les deux types de codeurs MTO (*Many To One*) et OTM (*One To Many*) ont été étudiés. Les résultats atteints s'avèrent très bons, en étant meilleurs que ceux obtenus précédement. Pour les configurations étudiées, la structure OTM s'est avérée meilleure que MTO.

Les architectures proposées ont été généralisées aux filtres récursifs RII (*Réponse Impulsionnelle Infinie*) avec succès. Deux type de structures, directe et transposée, ont été étudiées, correspondant respectivement aux structures MTO et OTM des codeurs convolutifs récursifs. Comme pour les codeurs, de bons compromis débit/surface ont pu être obtenus après synthèse et implantation sur FPGA pour les deux types de structures (directe et transposée). De même, ces résultats montrent que les architectures parallèles-pipeline des filtres offrent de meilleurs compromis débit/surface pour les structures transposées.

Concernant les objectifs de sûreté de fonctionnement, nous avons proposé une méthodologie pour concevoir des architectures de codeurs tolérantes aux fautes pour les codes cycliques en blocs, ces architectures devant être capables de détecter l'occurrence de fautes temporaires de type SEU ou MBU et le cas échéant, de reprendre l'opération de codage depuis le début.

Nous avons étudié le comportement de nos architectures en présence de fautes, de manière formelle et par injection de fautes. L'approche formelle nous a permis d'identifier une classe d'erreurs pour laquelle la détection est toujours assurée. L'approche par injection de fautes a été employée pour évaluer les taux de détection d'erreurs pour différentes classes d'erreurs : différentes campagnes d'injection de fautes simples (de type SEU – poids 1) et multiples (de type MBU à poids faibles 2, 3 et à poids élevés supérieurs à 6) ont été réalisées. Les performances temporelles et coûts matériels ont été évalués pour des implantation sur FPGA de la famille Stratix II.

Les résultats obtenus montrent que nos architectures possèdent une bonne capacité de détection d'erreurs, supérieures à 90% pour des fautes simples et supérieures à 50% pour le cas défavorable de fautes aléatoires de poids supérieur à 6. Les implantations sur FPGA montrent que le surcoût matériel par rapport aux architectures non sécurisées reste limité (15% à 45% en moyenne) et toujours inférieur à celui induit par la sécurisation par duplication ($> 100\%$). La réduction de performances temporelles (fréquence et débit maximaux de fonctionnement) induite par la sécurisation reste raisonnable (inférieure à 50%). Afin de limiter cette réduction de performances, nous avons appliqué des techniques de "pipelining" aux architectures sécurisées précédentes. L'approche s'est avérée efficace uniquement dans certaines conditions (quand le degré de parallélisme des architectures est supérieur au degré du polynôme générateur du code correcteur d'erreur traité par le codeur.

L'étude d'architectures tolérantes aux fautes a ensuite été étendue aux décodeurs parallèles-pipeline pour les codes cycliques en blocs. L'approche choisie repose sur une légère modification des architectures parallèles-pipeline développées au laboratoire LICM, l'objectif étant de permettre aux architectures de s'auto-contrôler. Les implantations sur FPGA démontrent l'efficacité de méthode proposée,

le surcoût matériel requis pour la sécurisation étant faible et les performaces temporelles étant globalement identiques.

Les perspectives à cours terme concernent d'une part, l'adaptation aux codeurs pour turbo-codes de l'approche proposée pour les codeurs convolutifs récursifs, et d'autre part, la prise en compte de modèles de fautes multiples plus élaborés tels que les fautes affectants plusieurs registres simultanément (MCU – *Multiple Cell Upset*).

À plus long terme, il serait intéressant d'étendre l'étude de SdF aux architectures rapides de codeurs et décodeurs pour codes convolutifs récursifs et non-récursifs. Un autre objectif intéressant est la prise en compte d'autres types de modèles de fautes tels que les fautes de délai et dérives d'horloge, attaques laser dont la probabilité d'occurence devient de plus en plus importante dans les nouvelles technologies.

Bibliographie

[1] F. Lustenberger, « On the Design of Analog VLSI Iterative Decoders », Dissertation ETH No. 13879, Serie in *Signal and Information Processing* : Volume 2, Hartung Gorre, Konstanz, Allemagne, novembre 2000.

[2] E. Normand, « Single event upset at ground level », *IEEE Trans. Nuclear Science*, vol. 43, n^{o}6, pp. 2742–2750, Dec. 1996.

[3] P. Thitimajshima, « Les codes Convolutifs Récursifs Systématiques et leur application à la concaténation parallèle », Thèse de Doctorat en Electronique, Université de Bretagne Occidentale, France, 1993.

[4] S. Philip, « Etude et développement d'un nouvelle architecture de processeur DSP dédiée aux applications modem en télécommunication sur câble T.V », manuscrit de thèse, LICM, Univérsité de Metz, (Metz, France), Déc. 1999.

[5] C. R. Baumann, « Radiation-induced soft errors in advanced semiconductor technologies », *IEEE Trans. Device and Materials Reliab.*, vol. 5, pp. 305–316, Sept. 2005.

[6] S. Lin and D. J. Costello, Jr., *Error Control Coding : Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, N.J., 1983.

[7] J. Oswald, *Théorie de l'Information ou Analyse Diacritique des systèmes*, Ed. Masson, 1986.

[8] A. Poli et Li. Huguet, *Codes Correcteurs, Théorie et Applications*, Ed. Masson, 1989.

[9] Proakis, J. G., 1989, *Digital Communication*, 4th ed., McGraw-Hill, NY, USA.

[10] J.L. Massey. « Step-by-step decoding of the decoding BCH codes », *IEEE Trans. Inform Theory*, pp.580–585, Oct. 1965.

[11] E. R. Berlekamp. *Algebraic Coding Theorie*. Aegean Park Press, 1968.

[12] A.J. Viterbi. Error bounds for convolutionnal codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inform. Theory*, vol. 13, pp. 260–269, 1967.

[13] D. Chase. A class of algorithms for decoding block codes with channel measurement information. *IEEE Trans. Inform. Theory*, pp. 170–182, janvier 1972.

[14] G.D. Forney. *Concatenated codes.* Cambridge MA, MIT Press, 1966.

[15] Erwan Piriou, « Apport de la modélisation et de la synthèse haut niveau dans la conception d'architecture flexible dédiée aux turbocodes en blocs », Thèse de doctorat : Sciences pour l'ingénieur : Institut TELECOM/ TELECOM Bretagne, Université de Bretagne Sud : 2007.

[16] L. Lampe, M. Jain, R. Schober, « Improved decoding for Bluetooth systems », *Proc. ICC 2005*, 16-20 May 2005, pp. 2511–2515, Vol. 4.

[17] L. Yin, J. Lu, L., K.B., Y. Wu, « Burst-error-correcting algorithm for Reed-Solomon codes and its performance over a bursty channel », *Proc. IEEE Int. C. on Circuits and Systems and West Sino Expositions'02*, Volume 1, Issue , 29 June-1 July 2002, pp.77–81.

[18] Y. Fei Guo, Z. C. Li, Q. Wang, « An area-efficient reed-solomon decoder for HDTV channel demodulation », *Proc. of the 2nd IEEE ASME'06*, pp. 1–5, Aug. 2006.

[19] S.S. Alekseev, A.V. Krivosheikin, « Digital implementation of DVB-C reverse channel receiver », *IEEE ISCE'06*, pp. 1–3, 2006.

[20] D. Declercq, « Optimisation et performances des codes LDPC pour les canaux non-standards », Habilitation à diriger les Recherches, Université de Cergy-Pontoise, Décembre 2003.

[21] C. Poulliat, D. Declercq, C. LamyBergot, I. Fijalkow, « Analysis and optimization of irregular LDPC codes for joint source-channel decoding », *IEEE Communications Letters*, Vol. 9, No. 12, pp. 1064–1066, December 2005.

[22] T. Tian, C.R. Jones, J.D. Villasenor, R.D. Wesel, « Selective avoidance of cycles in irregular LDPC code construction », *IEEE Trans. on Commun.*, Vol. 52, No. 8, pp. 1242–1247, Aug. 2004.

[23] F.J. MacWilliams and N.J.A. Sloane. « The Theory of error correcting codes », *North Holland Publishing Company*, 1978.

[24] M. C. Valenti, « Iterative detection and decoding for wireless communication », manuscrit de thèse PhD, Virginia Polytechnic Institute and State University, (Blacksburg, Virginia), July 1999.
Disponible sur http ://scholar.lib.vt.edu/theses/available/etd-071399-133447, consulté en 2009.

[25] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Inf. and Control*, 3 pp. 68–79, Mars 1960.

[26] A. Hocquengehm. « Codes correcteurs d'erreurs », *Chiffres*, 2, pp. 147–156, 1959.

[27] G.C Clark and J.B Cain. Error-Correction Coding for Digital Communications. *Plenum Press*, 1981.

[28] W. W. Peterson. Encoding and error correcting procedures for the Bose-Chaudhuri codes. *IRE Trans. Theory*, IT-6, pp. 459–470, Sept. 1960.

[29] D. Gorenstein and N. Zierler. A class of error correcting codes in p^m symbols. *J. SIAM*, 9, pp. 207–214, 1961.

[30] S.M. Reddy and J.P. Robinson. Random error and burst correction by iterated codes. *IEEE Trans. Inform. Theory*, 18, pp. 182–185, 1972.

[31] V. Cappellini, *Data Compression and Error Control Techniques with Applications*, Academic Press, 1985.

[32] A. J. Viterbi. « Error bounds for convolutional codes and an asymptotically optimum decoding algorithm ». *IEEE Trans. on Inf. Theory*, IT-13, pp. 260–269, April. 1967.

[33] J. Jin, C-Y. Tsui, « A low power viterbi decoder implementation using scarce state transition and path runing scheme for high throughput wireless applications », *Proc. IEEE ISLPED'06*, pp. 406–411, Tegernsee, Hong Kong, 4–6 Oct. 2006.

[34] Siu Manhung , A. Chan, « A Robust viterbi algorithm against impulsive noise with application to speech recognition », *IEEE Trans. on ASLP*, vol. 6, pp. 2122–2133, Nov. 2006.

[35] L-F. Chen, Ch-Y. Lee, « Design of a DVB-T/H COFDM receiver for portable video applications », *IEEE Communications Magazine*, vol. 45, pp. 112–120, Aug. 2007, Toronto, Ont., Canada.

[36] A. Cardenal-López, C. García-Mateoa, L. Docío-Fernández, « Weighted viterbi decoding strategies for distributed speech recognition over IP networksstar », *Speech Communication*, vol. 48, pp. 1422–1434, Nov. 2006.

[37] J. G. Wade, *Codage et Traitement du Signal, L'Exemple des Systèmes Vidéo Numériques*, Ed. Masson, 1991.

[38] A.D. Liveris, Xiong Zixiang, C.N. Georghiades, « Distributed compression of binary sources using conventional parallel and serial concatenated convolutional codes », *Proc. IEEE DCC'03*, pp. 193–202, March 2003.

[39] M. Eroz, F-W. Sun, L-N. Lee, « DVB-S2 low density parity check codes with near Shannon limit performance », *International Journal of Satellite Communications and Networking*, vol. 22, pp. 269–279, June 2004.

[40] Reimers, U.H. « DVB-The family of international standards for digital video broadcasting », *Proc. of the IEEE*, vol. 94, pp. 173–182, Jan. 2006.

[41] P. Elias, « Error Free Coding », *IRE Transaction on Inf. Theory*, vol. IT-4, pp. 29–37, September 1954.

[42] C. Berrou and A. Glavieux. « Near optimum error correcting coding and decoding : turbo-codes », *IEEE Transactions on Communications*, Vol. 44, pp. 1261–1271, Oct. 1996.

[43] C. E. Shannon. « A mathematical theory of communication », *Bell System Technical Journal*, 27, pp. 623–656, October 1948.

[44] ETSI tr 101 790 v1.1.1 digital video broadcasting, « interaction channel for satellite distribution systems », Guidelines for the use of en301 790.

[45] CCSDS 101.0-b-6. telemetry channel coding. blue book. issue 6. october 2002.

[46] T. Richardson and R. Urbanke. « The capacity of low-density parity check codes under message-passing decoding », *IEEE Transactions on Information Theory*, 47, pp. 599-618, February 2001.

[47] ETSI en 302 307 v1.1.1 digital video broadcasting, second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications

[48] Jason P. Woodard and Lajos Hanzo. « Comparative study of turbo decoding techniques : An overview », *IEEE Transactions on Vehicular Technology*, 49(6), pp. 2208–2233, November 2000.

[49] J. J. Boutros, « Les turbo codes parallèles et séries, décodage SISO et performance ML », rapport, Oct. 1998.

[50] M. M'sir, « Conception d'architectures rapides pour codes convolutifs en télécommunications : Application aux turbo-codes », Thèse de doctorat (LICM), 2003, Metz.

[51] A. M'sir, F. Monteiro, A.Dandache, B. Lepley, « Designing a high speed decoder for cyclic codes », *proc. IEEE IOLTS'05*, pp. 129–134, 12-14 July 2004.

[52] Cheng Chao, K.K. Parhi, « High-Speed Parallel CRC Implementation Based on Unfolding, Pipelining, and Retiming », *IEEE Transactions on Circuits and Systems*, pp. 1017–1021, vol. 53, issue 10, Oct. 2006.

[53] F. Monteiro, A. Dandache, A. M'sir and B. Lepley, « A fast CRC implementation on FPGA using a pipelined architecture for the polynomial division », *Proc. 8th ICECS*, St Julian, Malta, Sept. 2001, pp.1231–1234.

[54] T. Vallino,S. J. Piestrak, A. Dandache, F. Monteiro, B. Lepley, « Study of a new parallel architecture dedicated to the family of the difference set cyclic codes », *Proc. 5th IEEE Int. On-Line Testing Workshop*, (Rhodes, Greece), pp. 237–239, July 1999.

[55] R. J. Glaise, "two-step computation of cyclic redundancy code CRC-32 for ATM networks," *IBM J. Res. Develop.*, vol. 41, pp. 705–709, 1997.

[56] R. F. Hobson and K. Cheung, "A high-performance CMOS 32-bit parallel CRC engine," *IEEE J. Solid State Circuits*, vol. 34, pp. 233–235, Feb. 1999.

[57] F. Monteiro, A. Dandache and B. Lepley, « Fast configurable polynomial division for error control coding applications », *in Proc., IOLTW 2001*, (Taormina, Italy), pp. 158–161, July 2001.

[58] Jun Zhang, Zhi-Gong Wang, Qing-Sheng Hu, Jie Xiao, « Optimized design for high-speed parallel BCH encoder », *IEEE Transactions on Circuits and Systems*, pp. 427–431, May 2005.

[59] Cho Junho, Sung Wonyong, « Strength-Reduced Parallel Chien Search Architecture for Strong BCH Codes », *Proc. IEEE VLSI Design and Video Technology*, pp. 97–100, volume 55, issue 5, May 2008.

[60] Y.S. Kavian, A. Falahati, A. Khayatzadeh, M. Naderi, « High speed Reed-Solomon decoder with pipeline architecture », *proc. IEEE WOCN'05*, pp. 415–419, March 2005.

[61] Hanho Lee, Member, IEEE « A high-speed low-complexity Reed-Solomon decoder for optical communications », *in 10th IOLTS*, pp.129–134, Funchal, Madeira Island, Portugal, July 2004.

[62] T. K. Matsushima, T. Matsushima, and S. Hirasawa, « Parallel encoder and decoder architectures for cyclic codes », *IEICE Trans. A*, vol. E79–A, pp. 1313–1323, September 1996.

[63] K. Kobayashi, K. Yamano, H. Kokubun, and K. Kobayashi, “A 50 MHz CMOS pipelined majority logic decoder for (1 057 813) difference-set cyclic code," *IEICE Trans. A*, vol. E79–A, pp. 1060–1067, July 1996.

[64] A. M’sir, F. Monteiro, A. Dandache, B. Lepley, « Design of a high speed parallel encoder for convolutional codes », *Microelectronics Journal,* vol. 35(2), pp. 151–166, Feb. 2004.

[65] Y. Robert, M. Tchuente, “An efficient systolic array for The 1–D convolution problem", *J. VLSI Comput. Syst.*, pp. 398–407, 1986.

[66] H. T. Kung, “Why systolic architectures ?", Computer 15 (1), 1982, pp. 37–46.

[67] H. Jaber, F. Monteiro, A. Dandache, « Low-Complexity Parallel-Pipeline Architecture for MTO-Convolutional Encoders », *IEEE NEWCAS’09*, June 28 - July 1, Touloulse, France.

[68] H. Jaber, F. Monteiro, A. Dandache, « An Effective Fast and Small-Area Parallel-Pipeline Architecture for OTM-Convolutional Encoders », *IEEE IOLTS’09*, June 24 - 27, Lisbon, Portugal.

[69] H. Jaber, F. Monteiro, A. Dandache, « A New Effective Parallel-Pipelined Architectural Scheme for High-Speed Small-Area IIR Digital Filters », *DCIS’09*, November 18 - 20, Zaragoza – Spain.

[70] F. Monteiro, S. J. Piestrak, H. Jaber, A. Dandache, « Fault-secure interface between fault-tolerant RAM and transmission channel using systematic cyclic codes », *Proc. 13th IEEE IOLTS’07*, Crete, Greece, Jul. 2007, pp. 199–200.

[71] H. Jaber, et al., « Design of Parallel Fault-Secure Encoders for Systematic Cyclic Block Transmission Codes », *Microelectronics Journal*, volume 40, issue 12, pp. 1686–1697, Dec. 2009.

[72] H. Jaber, F. Monteiro, A. Dandache, « Improving the design of parallel-pipeline cyclic decoders towards fault-secure versions », *IEEE APCCAS’08*, pp. 324-327, Nov. 30 - Dec. 3 2008, Macao, China.

[73] J.C. Laprie, J. Arlat, J-P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J-C. Fabre, H. Guillermain, M. Kaaniche, C. Mazet, D. Powell, C. Rabéjac et P. Thévenod., « Guide de la Sûreté de Fonctionnement », 2ème édition (Cépaduès), 1996.

[74] J.C. Laprie, « Sûreté de fonctionnement des systèmes : concepts de base et terminologie », *REE : Revue de l’Electricité et de l’Electronique*, Dec. 2004.

[75] R. SCHOENIG, « Définitions d’une méthodologie de conceptions de systèmes mécatroniques sûrs de fonctionnement ». Thèse de Doctorat. 2004. Institut National Polytechnique de Lorraine.

[76] A. Villemeur, « Sûreté de fonctionnement des systèmes industriels », Ed. EYROLLES, Paris, Fance, mars 1997.

[77] J. Arlat, « Informatique sûre de fonctionnement : défis et solutions. Sûreté des procédés industriels : journées CNRS/CRIN du 11 octobre 1995.

[78] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. *In Proceedings of the 12thACM Symp. Principles of Programming Languages (POPL'85)*, pages 97–107, New Orleans, LA, USA, 1985.

[79] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *In ACM Transactions on Programming Languages and Systems*, volume 8, pages 244–263, April 1986.

[80] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *In Proceedings of the 1st IEEE Symp. Logic in Computer Science (LICS'86)*, pages 332–344, Cambridge, MA, USA, June 1986.

[81] A. Avižienis, Jean-Claude Laprie, B. Randell, C. Landwehr, « Basic Concepts and Taxonomy of Dependable and Secure Computing », *IEEE Trans. on Dependable and Secure Computing*, 1 (1), pp. 11–33, janv. 2004.

[82] JF. AUBRY, « Conception des systèmes de commande numériques des convertisseurs électromécaniques : vers une méthodologie intégrant la sûreté de fonctionnement ». Thèse d'Etat. 1987. Institut National Polytechnique de Lorraine.

[83] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, D. Powell, « Fault Injection For Dependability Validation - A Methodology and Some Applications », *IEEE Trans. on Software Engineering*, vol. 16, pp. 166–182, Février 1990.

[84] H. Madeira, M. Rela, F. Moreira, J.G. Silva, « RIFLE : A general purpose pin-level fault injector », *Proc. First European Dependable Computing Conference*, pp. 199–216, 1994.

[85] R.J. Martinez, P.J. Gil, G. Martin, C. Pérez, J.J. Serrano, « Experimental validation of high-speed fault tolerant systems using physical fault injection », *Proc. Dependable Computing for Critical Applications*, San Jose, Californie, p. 249, Janvier 1999.

[86] T. Michel, R. Leveugle, G. Saucier, R. Doucet, P. Chapier, « Taking advantage of ASICs to improve dependability with very low overheads », *Proc. European Design and Test Conference*, pp. 14–18, 1994.

[87] U. Gunneflo, J. Karlsson, J. Torin, « Evaluation of error detection schemes using fault ijection by heavy-ion radiation », *Proc. 19th Symp. Fault-Tolerant Computing (FTC-19)*, pp. 340–347, 1989.

[88] J. Karlsson, U. Gunneflo, P. Lidén, J. Torin, « Two fault injection techniques to test of fault handling mechanisms », *Proc. Test Conference (ITC'91)*, pp. 140–149, 1991.

[89] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, « Comparison and integration of three diverse physical fault injection techniques », *Predictably Dependable Computing Systems*, pp. 309–327, 1995.

[90] J.R. Sampson, W. Moreno, F. Falquez, « Validating fault tolerant designs using laser fault injection (LFI) », *Proc. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT'97)*, Paris, France, pp. 175–183, Octobre 1997.

[91] G.A. Kanawati, N.A. Kanawati, J.A. Agraham, « FERRARI : a tool for the validation of system dependability properties », *Proc. 22nd Symp. on Fault-Tolerant Computing (FTCS-22)*, pp. 336–344, 1992.

[92] J. Carreira, H. Madeira, J.G. Silva, « Xception : A technique for the experimental evaluation of dependability in modern computers », *IEEE Trans. on Software Engineering*, vol. 24, pp. 125–136, Feb. 1998.

[93] A. Benso, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, « EXFI : A low-cost fault injection system for embedded microprocessor-based boards », *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, vol. 3, issue 4, pp. 626–634, Octobre 1998.

[94] E. Cabau, « Etude de sûreté des installations electriques », Cahier Technique Schneider Electric, ^{o}184, Janvier, 1999.

[95] J. Faucher, « Pratique de l'AMDEC », Ed. Dunod, Paris 2004.

[96] J.P. Calvez, « Spécification et conception des systèmes – une méthodologie ». Masson, Paris 1992.

[97] F.X. Lenoir, « S'engager sur la sécurité de fonctionnement. Industries et Techniques ». n^{o}818, 2000.

[98] L. Anghel, R. Leveugle, and P. Vanhauwaert, « Evaluation of SET and SEU effects at multiple abstraction levels », *Proc. 11th IEEE Int. On-Line Testing Symposium (IOLTS'05)*, pp. 309–312, July 2005.

[99] P. Vanhauwaert, « Fault-injection based dependability analysis in a FPGA-based environment », *Thèse de doctorat*, Institut Polytechnique de Grenoble, April 2008.

[100] L. Anghel, « Test et Conception en vue du Test des Circuits Intégrés ». Cours ENSERG, Grenoble, Sept. 2002.

[101] X. Wendling, « Synthèse de circuits PC/PO à haute sûreté de fonctionnement ». Rapport de Projet. Laboratoire CSI-INPG, Mars 1994.

[102] Y. Monnet, « Etude et modélisation de circuits résistants aux attaques non intrusives par injection de fautes ». Thèse de doctorat, Institut National Polytechnique de Grenoble, Avr. 2007.

[103] K.S. Papadomanolakis, A.P. Kakarountas, V. Kokkinos, N. Sklavos, C.E. Goutis. « A comparative study on fault secure signed multiplication designs », *Proc. 11th International Conference on VLSI, The Global System On Chip Design & CAD Conference (IFIP VLSI SOC '01)*, pp. 183–188, Dec. 3-5 2001, Montpellier, France.

[104] L. Anghel, M. Nicolaidis, « Cost reduction and evaluation of a temporary faults detecting technique », *Design, Automation and Test in Europe Conference (DATE), Conference IEEE CS*. pp. 591–597. March 2000. Paris, France.

[105] F. Vargas, A. Amory, « Recent improvements on the specification of transient fault-tolerant VHDL descriptions : a case-study for area overhead analysis », *proc. 13th Symposium on Integrated Circuit and Systems Design (SBCCI'00)*. pp. 249–254. September 18-24 2000, Manaus, Brazil.

[106] M. Pflanz, K. Walther, C. Galke, H.T. Vierhaus, « On-Line Detection and Correction in Storage Elements with Cross-Parity Check », *Proc.* 8^{th} *IEEE Int. On-Line Testing Workshop (IOLTW'02)*. pp. 69–73, July 08-10, 2002, Isle of Bendor, France.

[107] R. Rochet, « Synthèse Automatique de contrôleurs avec contraintes de Sûreté de Fonctionnement », Thèse de doctorat (CSI-INPG), 1996, Grenoble.

[108] R. Leveugle, « Automatic modifications of high level VHDL descriptions for fault detection or tolerance », *Proc. Design, Automation and Test in Europe Conference (DATE'02)*. pp. 837–841. March 4-8 2002.

[109] C. Galke, M. Pflanz, H. T. Vierhaus, « On-line detection and compensation of transient errors in processor pipeline-structures », *Proc.* 8^{th} *IEEE Int. On-Line Testing Workshop (IOLTW'02)*. July 08/10 2002. Isle of Bendor, France.

[110] Eiji Fujiwara, Kohji Matsuoka, « A Self-Checking Generalized Prediction Checker and Its Use for Built-In Testing ». *IEEE Trans. Computers*, vol. 36(1), pp. 86–93, 1987.

[111] J. Khakbaz, E. J. McCluskey, « Self-testing embedded parity checkers ». *IEEE Trans. Computers*, vol 33(8), pp. 753–756, 1984.

[112] M. Tremblay, Y. Tamir, « Fault-tolerance for high-performance multi-module VLSI systems using micro rollback », The MIT Press, March 1989, pp. 297–316.

[113] F. Lima, E. Costa, L. Carro, M. Lubaszewski, R. Reis, S. Rezgui, R. Velazco, « Designing and testing a radiation hardened 8051-like micro-controller », *Proc. 3rd Military and Aerospace Applications of Programmable Devices and Technologies International Conference*. v.1. 2000. Maryland.

[114] D.A Anderson, « Design of self-checking digital networks using coding techniques ». Coordinated Sciences Laboratory, Report R/527. University of Illinois, Urbana, September 1971.

[115] D.A Anderson, G. Metz, « Design of totally self-checking check circuits for m-out-of-n codes », *IEEE Trans. on Comput.*, vol. C-22, pp. 263–269, March 1973.

[116] M.J. Ashjaee, S.M. Reddy, « On-totally self-checking checkers for separable codes », *IEEE Trans. on Comp.*, vol. C-26, pp. 737–744, Aug. 1977.

[117] J.E. Smith , G. Metz, « The design of totally self-checking combinatorials circuits », *Proc. 7th Fault Tolerant Computing Symposium*, Loss Angeless, USA, June 1997.

[118] J.E Smith , G Metz. « Strongly fault secure logic networks », *IEEE Trans. on Comp.*, vol. C-27, n^{o}6, June 1978.

[119] N.K. Jha , S-J. Wang, « Design and synthesis of self-checking VLSI circuits », *IEEE Trans. Comp. Aided Des.*, vol. 12, pp. 878–887, June 1993.

[120] W.W. Peterson, « On checking an adder », *IBM J. Res. Develop. 2*, pp. 166–168, April 1958.

[121] W.W. Peterson, E.J. Weldon , « Error-Correcting Codes », *second Ed., The MIT Press,* Cambridge, Massachusetts, 1972.

[122] A. Avizienis, « Arithmetic algorithms for error-coded operands », *IEEE Trans. on Comput.*, vol. C-22, No. 6, pp. 567–572, June 1973.

[123] I. Alzaher, M. Nicolaidis, « A Tool for automatic generation of self-checking multipliers based on residu arithmetic codes », *Proc. Design, Automation and Test in Europe Conference*, Munich, Germany, March 1999.

[124] L. Anghel « Les limites technologiques du silicium et tolérance aux fautes », Thèse de Doctorat. 2000. Laboratoire *INPG*, TIMA, Grenoble.

[125] W.C. Carter , P.R. Schneider, « Design of dynamically checked computers », *Proc. 4th Congress IFIP*, vol. 2, Edinburgh, Scotland, Aug. 5-10 1968, pp. 878–883.

[126] E.J. Mccluskey, F.W. Clegg, « Fault equivalence in combinational logic networks », *IEEE Trans. Comp.*, Vol. C-20, pp. 1286-1293, Nov. 1971.

[127] M. Nicolaidis, « Shorts in self-checking circuits », *Proc. Int. Test Conf., Washington, D.C.*, Sept. 1987, pp. 257–273.

[128] J. A. Clark, D. K. Pradhan, « Fault injection a method for validating computer-system dependability », *Computer*, Vol. 28, No. 6, June 1995, pp. 47–56.

[129] D. P. Siewiorek, R. S. Swarz, « Reliable computer systems, design and evaluation », second edition, Digital Press, 1992.

[130] A. Dantec, « Le phénomène de latchup dans les circuits integers CMOS », *Toute l'électronique*, TLE, 1981, 465, p. 45.

[131] JEDEC Standard, « Measurement and reporting of alpha particles and terrestrial cosmic ray-induced soft errors in semiconductor devices », JESD89, Août 2001.

[132] M. Baze, S. Buchner, « Attenuation of single event induced pulses in CMOS combinational logic », *IEEE Trans. on Nuclear Science*, Vol. 44, No 6, pp. 2217–2223, Dec. 1997.

[133] M. Nicolaidis, « Time redundancy based soft-error tolerance to rescue nanometer technologies », *Proc. 17th IEEE VLSI Test Symposium*, April 26-30, 1999, p.86–94.

[134] JEDEC Standard, « Test method for alpha source accelerated soft error rate », JESD89-2A, Octobre 2007.

[135] X. Bai, S. Dey, « High level crosstalk defect simulation for system on-chip interconnects », *Proc. 19th VLSI Test Symposium (VTS'01)*, Los Angeles, USA, Avril 2001.

[136] J. R. Samson et al., « Validating fault tolerance designs using laser fault injection », *Proc. IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, Paris, France, October 20-22, 1997, pp. 175–183.

[137] Damien Leroy, « Étude des modes de perturbation et de susceptibilité des circuits numériques aux collisions de particules et aux attaques laser », Thèse de Doctorat, Université de Metz, 1er décembre 2006.

[138] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, « Fault injection into VHDL models : A fault injection tool and some preliminary experimental results »,*Proc. IEEE Workshop on Integrating Error Models with Fault Injection*, April 1994, pp. 13–14.

[139] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, « Fault injection into VHDL models : the MEFISTO tool », *Proc. 24th FTC Symposium*, 1994, pp. 66–75.

[140] T. A. Delong, B. W. Johnson, J. A. Profeta III, « A fault injection technique for VHDL behavioral-level models », *IEEE Design & Test of Computers*, vol. 13, Winter 1996, pp. 24–33.

[141] J.S. Melinger, S. Buchner, D. McMorrow, W.J. Stapor, T.R. Weatherford, A.B. Campbell, H. Eisen, « Critical evaluation of the pulsed laser method for single event effects testing and fundamental studies », *IEEE Trans. on Nucl. Sci.*, vol. 41, n^{o}6, Dec. 1994, pp. 2574–2584.

[142] F. Vargas, D.L. Cavalcante, E. Gatti, D. Prestes, D. Lupi, « On the proposition of an EMI-based fault injection approach », *Proc. 11th IEEE Int. On-Line Testing Symposium (IOLTS'05)*, Saint-Raphael, France, pp. 207–208, July 2005.

[143] S. Duzellier, D. Falguère, L. Guibert, V. Pouget, P. Fouillat, R. Ecoffet, « Application of laser testing in study of SEE mechanisms in 16-Mbit DRAMs », *IEEE Trans. on Nuclear Science*, vol. 47, n^{o}6, 2392–2399, Déc. 2000.

[144] D. Lewis, V. Pouget, F. Beaudoin, G. Haller, P. Perdu, P. Fouillat, « Implementing laser-based failure analysis methodologies using test vehicles », *IEEE Trans. on Semiconductor Manufacturing*, vol. 18, n^{o}2, pp. 279–288, Mai 2005.

[145] R. Koga, « Single event effect ground test issues », *IEEE Trans. on Nuclear Science*, vol. 43, issue 2, pp. 661–670, Avril 1996.

[146] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, D. Rancey, A. Robinson, T. Lin, « FIAT - Fault Injection Based Automated Test Environment », *Proc. 22th FTC Symposium*, Tokyo, Japan, pp. 102–107, June 1988.

[147] R. Velazco, S. Rezgui, R. Ecoffet, « predicting error rate for microprocessor-based Digital Architecture through C.E.U (Code Emulating Upsets) Injection », *IEEE Trans. Nuclear Science*, vol. 47, n^{o}6, pp. 2405–2411, Décembre 2000.

[148] M. Lajolo, M. Rebaudengo, M. Sonza-Reorda, M. Violante, L. Lavagno, « Evaluating system dependability in a co-design framework », *Proc. of IEEE Design, Automation and Test in Europe (DATE'00)*, Paris, France, pp. 586–590, Mars 2000.

[149] C. Bolchini, L. Pomante, F. Salice, D. Sciuto, « Reliability properties assessment at system level : a co-design framework », *Proc. 7th Int. On-Line Testing Workshop (IOLTW'01)*, Taormina, Italie, pp. 165-171, Juillet 2001.

[150] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, « A Smart card test environment using multi-level fault injection in system C », *Proc. 6th Latin-American Test Workshop (LATW'05)*, Salvador, Brésil, pp. 103–108, Mars 2005.

[151] F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero, « RT-level fault simulation techniques based on simulation command scripts », *Proc. XV Conf. on Design of Circuits and Integrated Systems (DCIS'00)*, Montpellier, France, Nov. 2000, pp. 825-830.

[152] L. Berrojo, I. Gonzales, F. Corno, M. Sonza-Reorda, G. Squillero, L. Entrena, C. Lopez, « New Techniques for Speeding Up Fault Injection Campaigns », *Proc. Design, Automation and Test in Europe Conference (DATE'02)*, Paris, France, pp. 847–852, Mars 2002.

[153] J. Boué, P. Pétillon, Y. Crouzet, « MEFISTO-L : a VHDL-based Fault Injection Tool for the Experimental Assessment of Fault Tolerance », *Proc. 28th IEEE Int. Symposium on Fault- Tolerant Computing (FTCS-28)*, Munich, Allemagne, pp. 168–173, Juin 1998.

[154] J. Gracia, J. C. Baraza, D. Gil, P. J. Gil, « Comparison and Application of different VHDL-Based Fault Injection Techniques », *IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT'01)*, San Francisco, USA, Octobre 2001.

[155] R. Leveugle, K. Hadjiat, « Multi-level fault injections in VHDL descriptions : alternative approaches and experiments », *Journal of Electronic Testing : Theory and Applications (JETTA)*, vol. 19, pp. 559–575, Octobre 2003.

[156] H. Cha, E.M. Rudnick, J.H. Patel, R.K. Iyer, G.S. Choi, « A gate-level simulation environment for alpha-particle-induced transient faults », *IEEE Trans. on Computers*, vol. 45, issue 11, pp. 1248–1256, Novembre 1996.

[157] Alexandrescu, L. Anghel, M. Nicolaidis, « Simulating single event transients in VDSM ICs for ground level radiation », *Journal of Electronic Testing : Theory and Applications (JETTA)*, pp. 413–421, Août 2004.

[158] R. Leveugle, « Towards modeling for dependability of complex integrated circuits », *Proc. 5th IEEE Int. On-Line Testing Workshop (IOLTW'99)*, Rhodes, Grèce, pp. 194–198, Juillet 1999.

[159] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, « Exploiting FPGA for accelerating fault injection experiments », *Proc. 7th Int. On Line Testing Workshop (IOLTW'01)*, Taormina, Italie, pp. 9–13, Juillet 2001.

[160] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, « FPGA-based fault injection for microprocessor systems », *Proc. 10th Asian Test Symp. (ATS'01)*, Kyoto, Japon, p. 304, Novembre 2001.

[161] M. Portela-Garcia, C. López-Ongil, M. García-Valderas, L. Entrena-Arrontes, « Analysis of transient fault emulation techniques in platform FPGAs », *XIX Conf. on Design of Circuits and Integrated System (DCIS'04)*, Bordeaux, France, Novembre 2004.

[162] C. López-Ongil, M. García-Valderas, M. Portela-García, L. Entrena-Arrontes, « Autonomous transient fault emulation on FPGAs for accelerating fault grading », *Proc. 11th IEEE Int. On-Line Testing Symposium (IOLTS'05)*, Saint-Raphael, France, pp. 43–45, Juillet 2005.

[163] D. Divsalar and R. J. McEliece, « On the design of concatenated coding systems with interleavers », *TMO progress report 42-134*, Aug. 1998.

[164] Sh. He, Wen Wang, Sh. Li, J. Liu, Y. Liang, « 21 channel SAW channelizer filter bank », *Proc. Ultrasonics Symposium'04*, vol. 3, pp. 1918–1921, Aug. 23-27, 2004.

[165] Kee Kim, Sung-Ho Jang, Jong Sik Lee, « Adaptive distance filter-based traffic reduction for mobile grid », *Proc. MICDCS Workshop 2007,* vol. 22-29, pp. 8, Jun. 2007.

[166] J.I. Acha, « Computational structures for fast implementation of L-path and L-block digital filters », *IEEE Tran. on Circ. and Sys.*, vol. 36, pp. 805–812, Jun. 1989.

[167] K.K. Parhi, *VLSI Digital Signal Processing Systems : Design and Implementation*. Hoboken, NJ : Wiley, 1999.

[168] D.A. Parker, K.K. Parhi, « Low-area/power parallel FIR digital filter implementations », *IEEE Trans. Circuits and Syst.*, vol. 17, no. 1, pp. 75–92, 1997.

[169] MA.U. Adanayake, L. Bruton, « A high performance distributed-parallel-processor architecture for 3D IIR digital filters », *Proc. ISCAS'05*, vol. 2, pp. 1457–1460, May 2005.

[170] N.R. Shanbhag, Im. Gi-Hong, « Pipelined adaptive IIR filter architectures using scattered and relaxed look-ahead transformations », *IEEE Trans. on Signal Processing*, vol. 44, July 1996, pp. 1841–1847.

[171] A. Golmohammadi, M.T. Manzuri, S. Ayat, « A new pipeline implementation of an adaptive IIR filter for noise reduction application », *Proc. ISCIT'04*, vol. 1, pp. 577–58, Oct. 2004.

[172] Chao Cheng, Keshab K. Parhi, « Low-cost parallel FIR filter structures with 2-stage parallelism », *IEEE Trans. on Circuits and Syst.*, vol. 54, pp. 280–290, Feb. 2007.

[173] D. Gorinevsky, S. Boyd, « Optimization-based design and dmplementation of multi-dimensional zero-phase IIR filters », *IEEE Trans. on Circuits and Syst.*, vol. 53, pp. 372–383, Feb. 2006.

[174] R.A. Hawley, B.C. Wong, Lin Thu-Ji, J. Laskowski, H. Samueli, « Design techniques for silicon compiler implementations of high-speed FIR digital filters », *IEEE Trans. on Circuits and Syst.*, vol. 31, pp. 656–667, May 1996.

[175] Yen-Liang Chen, Chun-Yu Chen, Kai-Yuan Jheng, An-Yeu Wu, « A universal look-ahead algorithm for pipelining IIR filters », *IEEE VLSI-DAT'08*, pp. 259–262, Apr. 23-25, 2008.

[176] A.V. Openheim, R.W. Schafer, J.R. Buck, *Discrete-Time Signal Processing.* Prentice Hall, 1999.

[177] P. Reutz, « The architectures and design of a 20-MHz real-time DSP chip set », *IEEE J. Solid-State Circuits,* vol. 24, pp. 338–348, Apr. 1989.

[178] D. Larner, « Sun flips bits in chips », *Electronics Times*, no. 878, pp. 72–24, Nov. 1997.

[179] J.F. Ziegler, « Terrestrial cosmic rays intensities », *IBM J. Res. Develop.*, vol. 42, pp. 117–139, Jan. 1998.

[180] R.W. Hamming, « Error detecting and error correcting codes », *Bell System Tech. J.*, vol. 29, no. 2, pp. 147–160, Apr. 1950.

[181] M. Y. Hsiao, « A class of optimal minimum odd-weight-column SEC-DED codes », *IBM J. Res. Develop.*, vol. 25, no. 5, pp. 395–401, July 1970.

[182] C.L. Chen and M.Y. Hsiao, « Error-correcting codes for semiconductor memory applications : A state-of-the-art review », *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 124–134, Mar. 1984.

[183] F.J. Aichelmann, Jr., « Fault-tolerant design techniques for semiconductor memory applications », *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 177–183, Mar. 1984.

[184] T.R.N. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1989.

[185] G.C. Cardarilli *et al.*, « Design of a fault tolerant solid state mass memory », *IEEE Trans. Reliab.*, vol. 52, pp. 476–491, Dec. 2003.

[186] H. Kaneko, « Error control coding for semiconductor memory systems the space radiation environment », in *Proc. DFT'05*, Oct. 2005, pp. 93–101.

[187] M.Y. Hsiao, A.M. Patel, and D.K. Pradhan, « Store address generator with on-line fault-detection capability », *IEEE Trans. Comput.*, vol. C-26, pp. 1144–1147, Nov. 1977.

[188] L.T. Wang, « Autonomous linear feedback shift register with on-line fault-detection capability », *Dig. Pap. 12th Int. FTC Symp.*, Santa Monica, CA, Jun. 1982, pp. 311–314.

[189] C.W. Slayman, « Cache and memory error detection, correction, and workstations », *IEEE Trans. Device and Materials Reliab.*, vol. 5, no. 3, pp. 397–404, Sept. 2005.

[190] P. Koopman, « 32-bit cyclic redundancy codes for Internet applications », *Proc. 2002 Int. Conf. on Dependable Systems and Networks (DSN'02)*, 23–26 June 2002, pp. 459–468.

[191] P. Koopman and T. Chakravarty, « Cyclic redundancy code (CRC) polynomial selection for embedded networks », *Proc. 2004 Int. Conf. on Dependable Systems and Networks (DSN'04)*, 28 June – 01 July 2004, pp. 145–154.

[192] J. Ray and P. Koopman, « Efficient high Hamming distance CRCs for embedded networks », *Proc. 2006 Int. Conf. on Dependable Systems and Networks (DSN'06)*, 25–28 June 2006, pp. 3–12.

[193] S.J. Piestrak, A. Dandache, and F. Monteiro, « Designing fault-secure parallel encoders for systematic linear error correcting codes », *IEEE Trans. Reliab.*, vol. 52, pp. 492–500, Dec. 2003.

[194] G.R. Redinbo, « Fault-tolerant decoders for cyclic error-correcting codes », *IEEE Trans. Comput.*, vol. C-36, pp. 47–63, Jan. 1987.

[195] G.R. Redinbo, L.M. Napolitano, Jr., and D.D. Andaleon, « Multibit correcting data interface for fault-tolerant systems », *IEEE Trans. Comput.*, vol. 42, pp. 433–446, Apr. 1993.

[196] I.M. Boyarinov, « Self-checking decoding algorithm for Reed-Solomon codes », *Lecture Notes in Computer Science*, vol. 829, pp. 63–68, 1994.

[197] G.C. Cardarilli, S. Pontarelli, M. Re, and A. Salsano, « Concurrent error detection in Reed-Solomon encoders and decoders », *IEEE Trans. VLSI Systems*, vol. 15, no. 7, pp. 842–846, July 2007.

[198] H. Naeimi and A. DeHon, « Fault secure encoder and decoder for memory applications », *Proc. 22nd IEEE Int. Symp. on Defect and Fault-Tolerance in VLSI Systems (DFT'07)*, 26–28 Sept. 2007, pp. 409-417.

[199] S. Mitra, N.R. Saxena, and E.J. McCluskey, « A design diversity metric and analysis of redundant systems », *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 498–510, May 2002.

[200] C. Stroud *et al.*, « A parameterized VHDL library for on-line testing », *Proc. Int. Test Conf.*, 1–6 Nov. 1997, pp. 479–488.

[201] K. Mohanram, C.V. Krishna, and N.A. Touba, « A methodology for automated insertion of concurrent error detection hardware in synthesizable Verilog RTL », *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, 2002, pp. 577–580.

[202] Y. Makris, I. Bayraktaroglu, and A. Orailoglu, « Enhancing reliability of RTL controller-datapath circuits via invariant-based concurrent test », *IEEE Trans. Reliab.*, vol. 53, no. 2, pp. 269–278, June 2004.

[203] P. Oikonomakos and M. Zwolinski, « An integrated high-level on-line test synthesis tool », *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2479–2491, Nov. 2006.

[204] P. Oikonomakos and M. Zwolinski, « On the design of self-checking controllers with datapath interactions », *IEEE Trans. Comput.*, vol. 55, no. 11, pp. 1423–1434, Nov. 2006.

[205] M.C. Hsueh, T.K. Tsai, and R.K. Iyer, « Fault injection techniques and tools », *IEEE Computer*, vol. 30, no. 4, pp. 75–82, April 1997.

[206] L. Anghel, R. Leveugle, and P. Vanhauwaert, « Evaluation of SET and SEU effects at multiple abstraction levels », in *Proc. 11th IEEE Int. On-Line Testing Symposium (IOLTS'05)*, July 2005, pp. 309–312.

Site internet consulté en 2009

[207] http ://www.themanualpage.org/glossaire.

[208] Samuele Dietler'05 : http ://www.sweegy.ch/fileadmin/documents/documents/reports/Projet.

Table des figures

1.1 Modélisation d'une chaîne de transmission numérique 14
1.2 Graphe de transition du canal binaire symétrique 16
1.3 P_{E_b} en fonction du E_b/B pour un code Golay (23,12) avec décision ferme pour une modulation de type BPSK 17
1.4 Codage en bloc 19
1.5 Représentation d'un mot de code pour un code systématique 20
1.6 Multiplieur série MTO pour $H(x) = h_0 + h_1x^1 + h_2x^2 + h_3x^3$ 22
1.7 Multiplieur série OTM pour $H'(x) = h'_0 + h'_1x^1 + h'_2x^2 + h'_3x^3$ 23
1.8 Diviseur série MTO pour $G(x) = g_0 + g_1x^1 + g_2x^2 + g_3x^3$ 23
1.9 Diviseur série OTM pour $G(x) = g_0 + g_1x^1 + g_2x^2 + g_3x^3$ 24
1.10 Architecture série du décodeur cyclique 24
1.11 Implantation série de la division du polynôme par $G(x)$ 26
1.12 Codage convolutif 30
1.13 Exemple d'un diagramme d'état 31
1.14 Exemple d'une structure en arbre $(R = 1/2,\ m = 2)$ 32
1.15 Exemple d'une représentation en treillis $(R = 1/2,\ m = 2)$ 33
1.16 Exemple d'un codeur convolutif $(R = 1/2,\ m = 3)$. 33
1.17 Forme canonique du codeur convolutif MTO pour $m = 3$. 34
1.18 Forme canonique du codeur convolutif OTM pour $m = 3$. 35
1.19 Architecture CRC série pour $G(X) = 1 + x + x^8 + x^9$ 37
1.20 Architecture CRC série pour $G(X) = 1 + x + x^8 + x^9$ 38
1.21 Architecture parallèle-pipeline d'un CRC pour $p = 3$ 38
1.22 Architecture parallèle-pipeline du décodeur cyclique pour $p = 3$ 39
1.23 Schéma bloc d'un décodeur RS haut débit 40
1.24 Schéma fonctionnel du codeur systématique parallèle 41
1.25 Codeur parallèle MTO pour $R = 1/2$ et $m = 3$ 41
1.26 Codeur parallèle OTM pour $R = 1/2$ et $m = 3$ 42

2.1 Les techniques de vérification [73] 48
2.2 Reconfiguration dynamique 49
2.3 Diagramme des temps moyens 50

2.4 Schéma faute-erreur-défaillance 51
2.5 Différents raisonnements logiques 52
2.6 Chronologie d'une étude de la SdF 53
2.7 Cycle de développement en V d'un système embarqué 54
2.8 Architecture Duplication et Comparaison 57
2.9 Architecture de contrôle concurrente basé sur la redondance temporelle 58
2.10 Architecture TMR 62
2.11 La structure générale d'un circuit auto-contrôlable 64
2.12 Circuit sûr en présence de fautes 64
2.13 Single Event Upset (SEU) 67
2.14 Propagation d'un SET (a) exemple d'un circuit ; (b) formes d'onde des entrées de bascules 68
2.15 Faute due au couplage 69
2.16 Faute due au variations de l'horloge 70

3.1 Additionneur à 2 opérandes. 78
3.2 Structure combinatoire d'un additionneur à 4 opérandes. 79
3.3 Structure pipeline d'un additionneur à 4 opérandes. 79
3.4 Courbes caractéristiques du débit en fonction du degré de parallélisme. 80
3.5 Schéma fonctionnel du codeur MTO parallèle-pipeline 82
3.6 Schéma fonctionnel du codeur OTM parallèle-pipeline 83
3.7 Flot de conception 84
3.8 Codeur parallèle MTO pour $m = 3$ et $p = 3$ 86
3.9 Codeur parallèle OTM pour $m = 3$ et $p = 3$ 87
3.10 Simulation du codeur MTO série pour les polynômes G_1/H_1 de tableau 3.1 89
3.11 Simulation du codeur MTO parallèle-pipeline pour $m = 16$ et $p = 3$ et les polynômes G_1/H_1 de tableau 3.1 89
3.12 Simulation du codeur OTM série pour les polynômes G_2/H_1 de tableau 3.1 89
3.13 Simulation du codeur OTM parallèle-pipeline pour $m = 16$ et $p = 3$ et les polynômes G_2/H_1 de tableau 3.1 89
3.14 Comparatif compromis vitesse/surface pour les codeurs MTO-PPFC et MTO-PP . . 90
3.15 Comparatif compromis surface/parallélisme pour les codeurs MTO-PPFC et MTO-PP 90
3.16 Comparatif compromis vitesse/parallélisme pour les codeurs MTO-PPFC et MTO-PP 91
3.17 Comparatif compromis vitesse/surface du codeur OTM-PPFC et OTM-PP 93
3.18 Comparatif compromis surface/parallélisme du codeur OTM-PPFC et OTM-PP . . . 93
3.19 Comparatif compromis vitesse/parallélisme du codeur OTM-PPFC et OTM-PP . . . 94
3.20 Structure directe SD du filtre récursif série du 3'ème-ordre 97
3.21 Structure transposée ST du filtre récursif série du 3ème-ordre 98
3.22 Filtre SD parallèle-pipeline 99

3.23 Filtre ST parallèle-pipeline 100

4.1 Codeur non protégé entre une RAM tolérante aux fautes et un chaine de transmission 107
4.2 Schéma général de l'implantions série de la division du polynôme par $G(x)$ 109
4.3 Schéma d'un codeur FS (*Fault-Secure*) proposé 110
4.4 Structure d'un bloc de transmission de données codées 111
4.5 Architecture détaillée d'un codeur FS (*Fault-Secure*) parallèle 114
4.6 Compromis surface/fréquence de codeurs S et FS pour les configurations des tableaux 4.6 et 4.7 128
4.7 Schéma bloc d'un codeur FS (*Fault-Secure*) parallèle-pipeline 131
4.8 Structure pipeline pour la matrice M 132
4.9 Comparatif des performances (débit) entre les codeurs FS et FS pipeline pour les configurations de tableaux 4.6 et 4.7 133
4.10 Comparatif du surcoût matériel entre les codeurs FS pipeline (FSp) et simple-pipeline dupliqué (Sp (duplication)) pour les mêmes configurations 134
4.11 Architecture FS parallèle-pipeline du décodeur cyclique pour $p = 3$ 135
4.12 Réseau de décalage pour 3-bits d'entées 136
4.13 Comparatif compromis vitesse/surface du décodeur Simple et FS 139
4.14 Décodeur cyclique parallèle-pipeline FS sans erreur 139
4.15 Décodeur cyclique parallèle-pipeline FS avec une erreur injectée 140

Liste des tableaux

3.1 Polynômes générateurs [50] . . . 88
3.2 Codeurs testés [50] . . . 88
3.3 Dégradation de $f_{mto\ (p)}$ en fonction du degré de parallélisme p . . . 91
3.4 Dégradation de $f_{otm\ (p)}$ en fonction du degré de parallélisme p . . . 94
3.5 Comparaisons entre les architectures SD et ST parallèle-pipeline . . . 101
3.6 Comparaison des résultats expérimentaux pour les filtres SD et ST . . . 102

4.1 Quelques polynômes génèrent codes Hamming . . . 121
4.2 Quelques polynômes génèrent codes CRC standards . . . 121
4.3 Quelques polynômes génèrent codes BCH et codes RS standards . . . 122
4.4 Taux de détection d'erreurs pour une injection d'erreur simple, double et triple . . . 123
4.5 Taux de détection d'erreur pour injection d'erreurs aléatoires . . . 124
4.6 Comparatif (surface et vitesse) du codeur FS et S lorsque $G(x)$ est multiple de $G'(x)$ 125
4.7 Comparatif (surface et vitesse) du codeur FS et S lorsque $G(x)$ n'est pas multiple de $G'(x)$ . . . 126
4.8 L'ensemble des polynômes restes $W(x)$ résultant de la division de $G(x)$ par $G'(x)$ . . 127
4.9 Dimensions des différentes matrices . . . 131
4.10 Effet du pipeline sur le codeur FS . . . 134
4.11 Codes simulés . . . 138
4.12 Complexité de décodeurs parallèle-pipeline simple et FS for les polynômes du tableau 4.11 . . . 138
4.13 Notation de simulation . . . 139

Annexe A

Liste des acronymes

ASM : Amplitude Shift Keying.
ATM : Asynchronous Transfer Mode.
ASIC : Application Specific Integrated Circuits.
BCH : Bose Hocquenghem Chaudhuri.
BBAG : Bruit Blanc Additif Gaussien.
BPSK : Binary Phase Shift Keying.
CCSDS : Consultative Committee for Space Data System.
CDPD : Cellular Digital Packet Data.
CMOS : Complementary Metal-Oxide Semiconductor.
CLBs : Configurable Logic Blocks.
CBS : Canal Binaire Symétrique.
CRC : Cyclic Redundancy Check.
CNR : Codeur Non Récursif.
CR : Codeur Récursif.
CFC : Control Flow Checking.
CP$_{otm}$: Codeur Parallèle One To Many.
CP$_{mto}$: Codeur Parallèle Many To One.
DVD : Digital Versatile Disk.
DVB : Digital Video Broadcasting.
DVB-C : Digital Video Broadcasting-Cable.
DVB-H : Digital Video Broadcasting-Handhelds.
DVB-T : Digital Video Broadcasting-Terrestrial.
DVB-T : Digital Video Broadcasting-Satellite version 2
DSN : Deep Space Network.
DWC : Duplication With Comparison.
DWCR : Duplication With Complementary Redundancy.
DRC : Double Rail Checker.
DED : Double Error Detection.

DFA : Differential Fault Analysis.
ECC : Error Correcting Codes.
EDC : Error Detecting Codes.
EPD : Error Pattern Detection.
ESA : European Space Agency.
ETSI : European Telecommunications Standards Institute.
FSK : Frequency Shift Keying.
FDDI : Fiber Distributed Data Interface.
FIR : Finite Impulse Response.
FMDS : Fiabilité, Maintenabilité, Disponibilité, Sécurité.
FPGA : Field Programmable Gate Array.
FAST : FAult Simulator for Transients.
FIFA : Fault Injection by means of FPGA).
GSM : Global System for Mobile Communications.
HDTV : High Definition TeleVision.
HF : High Frequencies.
HDL : High-level Description Language.
IEEE : Institute of Electrical and Electronics Engineers.
IIR : Infinite Impulse Response.
ISA : Instruction Set Architecture.
QoS : Quality of Service.
QAM : Quadrature Amplitude Modulation.
LICM : Laboratoire Interfaces Capteurs et Micro-électronique.
ME : Motif d'Erreur.
MLV : Majority Logic Voting.
MDS : Maximum Distance Separable.
MAP : Maximum A Posteriori.
MTO : Many To One.
MBU : Multiple Bit Upsets.
MCU : Multiple Cell Upsets.
MEFISTO : Multi-level Error/Fault Injection Simulation Tool.
NASA : National Aeronautics and Space Agency.
PSK : Phase Shift Keying.
PGZ : Peterson Gorenstein Zierler.
PGCD : Plus Grand Commun Diviseur.
PrME : Prime Modified Euclid.
PPFC : Parallèle-Pipeline à Faible Complexité.
PP : Parallèle-Pipeline.
OTM : One To Many.

RAM : Random Access Memory.
RS : Reed-Solomon.
RSC : Récursif Systématique Convolutif.
RTL : Register Transfer Level.
REDWC : REcomputing Duplication With Comparison.
SEAL : Simple and Efficient Adaptation Layer.
SEE : Single Event Effect.
SEL : Single Event Latchup.
SEU : Single Event Upset.
SET : Single Event Transient.
SISO : Soft Input Soft Output.
SEC : Single Error Correction.
TEB : Taux d'Erreur Binaire.
TDF : Télédiffusion De France.
TMR : Triple Modular Redundancy.
TCL : Tool Command Language.
UAL : Unité Arithmétique et Logique.
VBR : Variable Bit Rate.
V & V : Vérification et Validation.
VLSI : Very Large Scale Integration.
VERIFY : VHDL-based Evaluation of Reliability by Injecting Faults efficientlY.

Annexe B

Liste des publications

- Houssein Jaber, et al., « Design of Parallel Fault-Secure Encoders for Systematic Cyclic Block Transmission Codes », *Microelectronics Journal*, volume 40, issue 12, pp. 1686–1697, Dec. 2009.
- Houssein Jaber, Fabrice Monteiro, Abbas Dandache, « A New Effective Parallel-Pipelined Architectural Scheme for High-Speed Small-Area IIR Digital Filters », *DCIS'09*, Zaragoza – Spain, Nov. 18 – 20, 2009, pp. 86–90.
- Houssein Jaber, Fabrice Monteiro, Abbas Dandache, « Low-Complexity Parallel-Pipeline Architecture for MTO-Convolutional Encoders », *IEEE NEWCAS'09*, Touloulse – France, June 28 – July 1, 2009, pp. 1–4.
- Houssein Jaber, Fabrice Monteiro, Abbas Dandache, « An Effective Fast and Small-Area Parallel-Pipeline Architecture for OTM-Convolutional Encoders », *IEEE IOLTS'09*, Lisbon – Portugal, June 24 – 27, 2009, pp. 257–261.
- Houssein Jaber, Fabrice Monteiro, Abbas Dandache, « Improving the Design of Parallel-Pipeline Cyclic Decoders Towards Fault–Secure Versions », *IEEE APCCAS'08*, Macao – China, Nov. 30 – Dec. 3, 2008, pp. 324–327.
- Houssein Jaber, Fabrice Monteiro, Abbas Dandache, « Sécurisation de décodeurs parallèles-pipeline pour codes correcteurs cycliques », *GDR* System on Chip - System in Package, Paris, France, 4–6 juin 2008.
- F. Monteiro, S. J. Piestrak, H. Jaber, A. Dandache, « Fault-secure interface between fault-tolerant RAM and transmission channel using systematic cyclic codes », *in 13th IEEE IOLTS'07*, Crete – Greece, Jul. 8 – 11, 2007, pp. 199–200.
- Houssein Jaber, Fabrice Monteiro, Abbas Dandache, « Fault-Secure Interface Between Fault–Tolerant RAM and Transmission Channel Using Systematic Cyclic Codes », 2nd Winter School, *SOES'07*, Schloss Dagstuhl, Germany, Nov. 2007.

RÉSUMÉ

Les systèmes de communication modernes exigent des débits de plus en plus élevés afin de traiter des volumes d'informations en augmentation constante. Ils doivent être flexibles pour pouvoir gérer des environnements multinormes, et évolutifs pour s'adapter aux normes futures. Pour ces systèmes, la qualité du service (QoS) doit être garantie malgré l'évolution des technologies microélectroniques qui augmente la sensibilité des circuits intégrés aux perturbations externes (impact de particules, perte de l'intégrité du signal, etc). La tolérance aux fautes devient un critère important pour améliorer la fiabilité et par conséquence la qualité de service. Cette thèse s'inscrit dans la continuité des travaux menés au sein du laboratoire LICM concernant la conception architecturale d'une chaîne de transmission à haut débit, faible coût, et sûre de fonctionnement. Elle porte sur deux axes de recherche principaux :

– le premier axe porte sur les aspects rapidité et flexibilité, et en particulier sur l'étude et l'implantation d'architectures parallèles-pipelines dédiées aux codeurs convolutifs récursifs. Le principe repose sur l'optimisation des blocs calculant le reste de la division polynomiale qui constitue l'opération critique du codage. Cette approche est généralisée aux filtres récursifs RII. Les caractéristiques architecturales principales recherchées sont une grande flexibilité et une extensibilité aisée, tout en préservant la fonctionnalité ainsi qu'un bon équilibre entre quantité de ressources utilisées (et donc surface consommée) et performances obtenues (vitesse de fonctionnement) ;
– le deuxième axe de recherche porte sur le développement d'une méthodologie de conception de codeurs sûrs en présence de fautes, améliorant ainsi la tolérance de circuits intégrés numériques. L'approche proposée consiste à ajouter aux codeurs des blocs supplémentaires permettant la détection matérielle en ligne de l'erreur afin d'obtenir des architectures sûrs en présence des fautes. Les solutions proposées permettent d'obtenir un bon compromis entre complexité et fréquence de fonctionnement. Afin d'améliorer encore le débit du fonctionnement, nous proposons également des versions parallèles-pipelines des codeurs sûrs. Différents campagnes d'injection de fautes simples, doubles, et aléatoires ont été réalisées sur les codeurs afin d'évaluer les taux de détection d'erreurs. L'étude architectures sûrs de fonctionnement a ensuite été étendue aux décodeurs parallèles-pipeline pour les codes cycliques en blocs. L'approche choisie repose sur une légère modification des architectures parallèles-pipeline développées au laboratoire LICM, introduisant une certaine redondance spatiales afin de le rendre sûrs de fonctionnement.

Mots clés : Sûreté de fonctionnement, transmission haut débit, codes en blocs et codes convolutifs, architectures parallèle-pipeline, injection de fautes, modélisation RTL.

ABSTRACT

Nowadays, modern communication systems require higher and higher data throughputs to transmit increasing volumes of data. They must be flexible to handle multi-norms environments, and progressive to accommodate future norms. For these systems, quality of service (QoS) must be guaranteed despite the evolution of microelectronics technologies that increase the sensitivity of integrated circuits to external perturbations (impact of particles, loss of signal integrity, etc). Fault-tolerance techniques are becoming more and more an important criteria to improve the dependability and the quality of service. This thesis'work continues previous research undertaken at the LICM laboratory on the architectural design of high-speed, low-cost, and dependable transmission systems. It focuses on two principal areas of research :

– The first research area concerns the speed and flexibility aspects, particularly on the study and implementation of parallel-pipelined architectures dedicated to recursive convolutional encoders. The principle is based on the optimization of blocks that calculate the remainder of the polynomial division which constitute the critical operation of the encoding. This approach is generalized to recursive IIR filters. The main architectural characteristics being aimed are high flexibility and scalability, yet preserving a good trade-off between the amount of resources used (and hence, area consumption) and the obtained performance (operation speed).
– The second topic concerns the developing of a methodology for designing FS (fault-secure) encoders, improving the tolerance of digital integrated circuits. The proposed approach consists in adding an extra blocks to the encoders, allowing online error detection. The proposed solutions offer a good compromise between complexity and frequency operation. For even higher throughput, parallel-pipelined implementations of FS encoders were considered. Different fault injection campaigns of single, double, and random errors were applied to the encoders in order to evaluate error detection rates. The study of dependable architecture was extended to pipeline-parallel decoders for cyclic block codes. This approach is based on a slight modification of the parallel-pipeline architectures developed at LICM laboratory, introducing some redundancy in order to make it dependable.

Key words : Dependability, high speed transmission, blocks and convolutional codes, parallel-pipeline architectures, faults injection, RTL modeling.