REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE SAAD DAHLEB DE BLIDA

035/07

Faculté des sciences de l'ingénieur
Département d'Aéronautique

PROJET DE FIN D'ÉTUDE

EN VUE DE L'OBTENTION DE DIPLOME D'INGENIEUR D'ÉTAT EN AERONAUTIQUE

Filière : Navigation aérienne en Aéronautique
Spécialité : Installation

THEME

ETUDE ET IMPLEMENTATION SUR FPGA D'UN CODE CORRECTEUR D'ERREURS DE TYPE REED-SOLOMON

Présentés par :

- LAYADI Bachreddine
- BENTOUCHA Mohamed Amine

Encadré par :

Mr. ALLAILOU BOUFELDJA

Année Universitaire 2006/2007

RESUME

Dans ce travail nous allons étudier et analyser le codage Reed–Solomon. Le codeur ainsi que le décodeur seront simulés sous VHDL afin de réaliser une implémentation FPGA.

Mots Clés :

FPGA, VHDL.

ABSTRACT

In the work we will study and analyse the Reed-Solomon coding. We will write a program with VHDL to simulate encoder and decoder, and then we discuss the hardware implementation on FPGA

Key words:

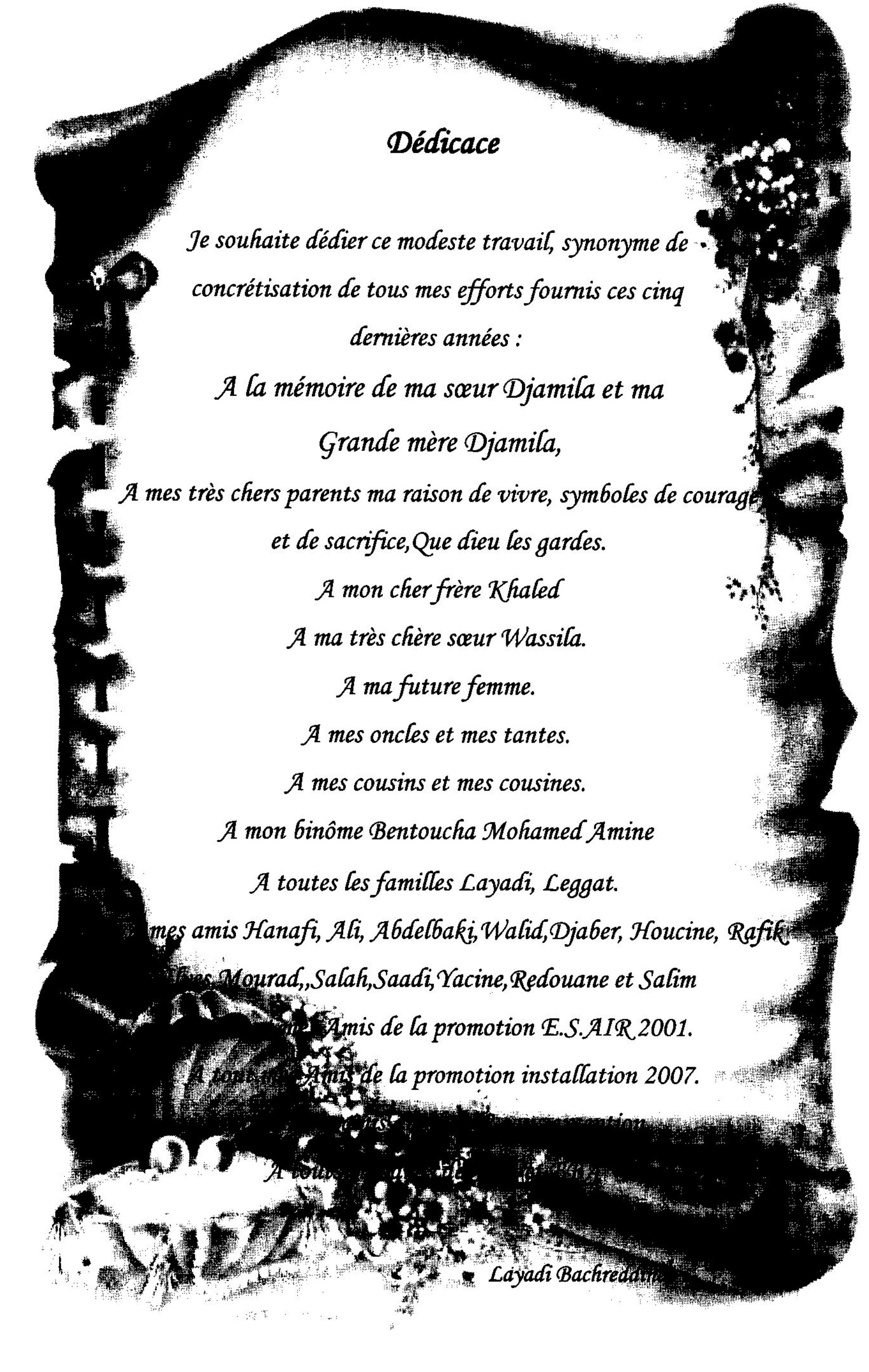
FPGA, VHDL

ملخص :

في هذا العمل سنقوم بدراسة وتحليل تشفير Reed-Solomon, سنقوم بكتابة برنامج باستعمال لغة VHDL لمحاكاة المشفر وفك الشفرة .ثم نقوم بمناقشة تنفيذها على أرض الواقع على FPGA .

مفاتيح

FPGA, VHDL



Dédicace

Je souhaite dédier ce modeste travail, synonyme de concrétisation de tous mes efforts fournis ces cinq dernières années :

A la mémoire de ma sœur Djamila et ma Grande mère Djamila,

A mes très chers parents ma raison de vivre, symboles de courage et de sacrifice,Que dieu les gardes.

A mon cher frère Khaled

A ma très chère sœur Wassila.

A ma future femme.

A mes oncles et mes tantes.

A mes cousins et mes cousines.

A mon binôme Bentoucha Mohamed Amine

A toutes les familles Layadi, Leggat.

[illegible] mes amis Hanafi, Ali, Abdelbaki,Walid,Djaber, Houcine, Rafik [illegible] Mourad,,Salah,Saadi,Yacine,Redouane et Salim

[illegible] Amis de la promotion E.S.AIR 2001.

A [illegible] Amis de la promotion installation 2007.

[illegible]

A [illegible]

Layadi Bachreddin[illegible]

Dédicace

Je souhaite dédier ce modeste travail, synonyme de concrétisation de tous mes efforts fournis ces cinq dernières années :

A mes très chers parents ma raison de vivre, symboles de courage et de sacrifice,Que dieu les gardes.

A ma grande mère,

A mes chers frères

A mes très chères soeurs.

A mes oncles et mes tantes.

A mes cousins et mes cousines.

A mon binôme Layadi Bachreddine

A toutes les familles Bentoucha, Bouchakour,

A mes amis (es).

A tout mes Amis de la promotion E.S.AIR 2001.

A tout mes Amis de la promotion installation 2007.

[illegible] mes amis du bloc 02 sans exception.

A tout mes amis de Chlef

REMERCIEMENTS

Avant tout je tiens à remercier Dieu tout puissant de nous avoir donné cette foi et cette confiance en soi, ainsi que cette volenté et ce courage, pour pouvoir nous voir un jour devant un jury, afin d'obtenir un tel diplôme. Je remercie encore tous ceux qui m'ont aidé de prés ou de loin, par leurs encouragements, leurs conseils et leurs critiques.

Qu'il nous soit permis en jour mémorable d'exprimer nos plus vifs remerciements et notre gratitude, à tous ceux qui ont accepté de juger ce travail, ainsi qu'à tous ceux ont aidé de mener à bien.

A notre promoteur **Mr. ALLAILOU Boufeldja**, **Mr. BENACHENHOU Kamel** et aussi **Mr. Amara Mouncef**, qui ont bien voulus diriger notre travail et aussi pour les conseils qu'ils n'ont pas cessés de nous les prodiguer.

Nous adressons également nos vifs remerciements et notre Profonde gratitude :

Au président et aux membres du jury pour l'honneur qu'ils Nous font, en acceptant de juger notre travail. Nous remercions aussi tous le personnels du centre de recherche CRD pour leurs soutient et aide durant notre formation et tout les professeurs et les membres d'encadrement qui nous ont soutenus à l'université et tous ceux qui nous ont aidé de près ou de loin à l'élaboration de ce mémoire.

A tous les enseignants qui on contribué à notre formation depuis notre premier pas a l'université.

Remerciement

SOMMAIRE

Introduction Générale..........1

Chapitre I :

Etat de l'art du codage

I-1/ Introduction..........3
I-2/ Le Codage dans l'informatique..........3
I-3/ Pourquoi coder ?..........4
I-4/ L'histoire du codage..........4
I-5/ Le codage des caractères..........5
I-5-1/ Codage ASCII étendu..........6
I-5-2/ Autres codes existants..........7
I-6/ Le contrôle d'erreurs : Codes de contrôle..........7
I-6-1/ Les Codes détecteurs..........8
I-6-1-1/ Principe du contrôle CRC..........10
I-6-1-2/ Algorithme pour le calcul du CRC..........11
I-6-1-3/ Les polynômes générateurs..........12
I-6-2/ Les Codes correcteurs..........13
I-6-2-1/ Le codage à triple répétition..........13
I-6-2-2/ Codes de Hamming..........13
I-6-2-2-1/ Un exemple d'utilisation..........14
I-6-2-3/ Les codes de Reed-Solomon..........16
I-7/ Outils mathématiques..........17
I-7-1/ Introduction..........17
I-7-2/ Groupe..........17
I-7-3/ Anneau..........17
I-7-4/ Corps ou champs..........18
I-7-5/ Champs de Galois..........18
I-7-5-1/ Elément des champs de Galois..........19
I-7-5-1-1/ Addition dans GF..........19
I-7-5-1-2/ Soustraction dans GF (2)..........20
I-7-5-2/ Polynôme primitif..........20
I-7-6/ Dérivation formelle d'un polynôme dans un champ de Galois..........21
I-8/ Conclusion..........22

Chapitre II :

Principe pratique de Reed-Solomon

II-1 /Introduction..........24
II-2/ Propriété des codes Reed Solomon..........24
II-2-1/ Exemple de polynôme..........25
II-3/ Codage..........25

II-3-1/ Théorie du codage 26
II-3-1-1/ Polynôme générateur 26
II-3-2/ Implémentation hardware du codeur 26
II-3-2-1/ Schéma 27
II-3-2-1-1/ Addition.....28
II-3-2-1-2/ Multiplication 28
II-4/ Décodage 28
II-4-1/ Généralités du décodage 30
II-4-2/ Calcul du syndrome 31
II-4-2-1/Schéma de calcul du syndrome 32
II-4-3/ Euclide.....32
II-4-3-1/ Généralité du théorème d'Euclide.....32
II-4-3-2/Correction d'erreur avec Euclide 33
II-4-4/ Chien search 36
II-4-4-1/ Schéma du chien de search 36
II-4-5/ Algorithme de Forney 37
II-4-5-1/ Schéma de l'algorithme de Forney et de corrections d'erreurs 38
II-4-5-1-1/ Evaluation du polynôme d'amplitude.....38
II-4-5-1-2/ Calcul des Inverses.....38
II-4-5-1-3/ Multiplication.....39
II-4-5-1-4/Détection du zéro 39
II-4-5-1-5/La porte logique « AND » 39
II-4-5-1-6/La porte logique « XOR ».....39
II-5/Conclusion.....40

Chapitre III :

Les circuits reconfigurables

III-1/ Introduction 42
III-2/ Les circuits logiques programmables.....43
III-3/ Les circuits FPGAs.....44
III-3-1/ Définition.....44
III-3-2/ Architecture.....45
III-3-3/ Les éléments structurels 46
III-3-3-1/ Les éléments logiques.....46
III-3-3-2/ Les éléments de mémorisation.....47
III-3-3-3/ Les éléments de routages 47
III-3-3-4/ Les éléments d'entrées/sorties 47
III-3-3-5/ Les éléments de contrôle et d'acheminement des horloges 48
III-3-4/ Configuration des circuits FPGAs 48
III-3-4-1/ Circuit configurable 48
III-3-4-2/ Circuit reconfigurable 48
III-3-4-3/ Circuit partiellement reconfigurable 48
III-3-4-4/ Circuit dynamiquement reconfigurable 49
III-3-5/ Architecture de la série XC4000E de Xilinx 49
III-3-5-1/ Les éléments de base 49
III-3-5-2/ Les CLB (configurable logic bloc) 50

III-3-5-3/ Les IOBs (input output bloc) 51
III-3-5-4/ Le système d'interconnexion 51
III-3-6/ Nomenclature des circuits FPGAs 52
III-4/ Le langage VHDL 52
III-4-1/ Présentation 53
III-4-2/ Objectifs du langage VHDL....53
III-4-3/ Développement d'un projet en VHDL 54
III-5/ L'outil de conception ISE 56
III-5-1/ Les environnements de développement 56
III-5-1-1/ Environnement de conception 56
III-5-1-2/ Environnement de simulation et de vérification 56
III-5-1-3/ Environnement de synthèse 57
III-5-1-4/ Environnement de l'implantation physique 57
III-6/ Conclusion 58

Chapitre IV :

Implémentation, Simulation et résultats

IV -1/ Introduction....60
IV -2/ Construction d'un GF(2^3)....60
IV -3/ Construction d'un champ de GF(2^3) sous Matlab....61
IV -4/ Exemple du nombre de symboles corrigibles 61
IV -5/Codage 62
IV -5-1/ Calcul des coefficients du polynôme générateur pour RS (7,5) 62
IV -5-2/ Implémentation Hardware du codeur....63
IV -5-2-1/Addition 64
IV -5-2-2/Multiplication....65
IV -6/Décodage....68
IV-6-1/Calcul du syndrome 68
IV -6-1-1/ 1er cas 69
IV -6-1-2/ 2er cas 69
IV -6-2/Correction d'erreurs avec Euclide 72
IV -6-3/ Chien search....73
IV -6-4/ Algorithme de Forney....77
IV -6-4-1/Schéma de l'algorithme de Forney et de la correction des erreurs...77
IV -6-4-1-1/ Evaluation du polynôme d'amplitude....77
IV -6-4-1-2/ Inversion avec ROM....78
IV -6-4-2/Calcul du polynôme de correction selon le schéma de Forney....79
IV -7/Implémentation Hardware....80
IV -7-1/ Introduction....80
IV -7-2/ Flux de conception hardware....80
IV -7-3/ Codeur....82
IV -7-3-1/ Eléments du codeur....82
IV -7-3-2/ Signaux de commande et d'entrée/sortie du codeur 82
IV -7-3-3/ Test Bench du codeur 83
IV -7-3-4/ performance du codage 84
IV -7-3-4-1/ Ressource hardware....84

IV -7-3-4-2/ débit binaire..84
IV-7-3-5/conclusion du codage..86
IV -7-4/ Décodeur 87
IV -7-4-1/ Syndrome_bloc...87
IV -7-4-2/ Signaux de commande et d'entrée/sortie syndrome _bloc...........87
IV -7-4-3/ Test Bench..88
IV -7-4-4/ Conclusion du décodage...88

Conclusion générale..90

Bibliographie 91

LISTE DES FIGURES

Chapitre I :

Figure I.1: Schéma fondamental du codage..4

Chapitre II :

Figure II.1: schéma général...24
Figure II.2: mot-code de Reed – Salomon..24
Figure II.3: schéma de codage...27
Figure II.4: schéma du décodage...29
Figure II.5: schéma pour le calcul du syndrome...32
Figure II.6: algorithme d'Euclide pour le calculant du polynôme de localisation et pour le polynôme d'amplitude...35
Figure II.7: schéma du bloc Chien Search...37
Figure II.8: schéma de l'algorithme de Forney..38
Figure II.9: schéma pour le calcul des racines du polynôme d'amplitude..............38

Chapitre III :

Figure III.1: Les différentes classes des PLD...43
Figure III.2: Les différentes architectures d'un circuit FPGA 45
Figure III.3: Structure interne d'une architecture...46
Figure III.4: Architecture interne de la série XC4000E...................................49
Figure III.5: Cellule logique (CLB)..50
Figure III.6:Input Output Block (IOB)...51
Figure III.7: Système d'interconnexion...52
Figure III.8: Etapes de développement d'un projet en VHDL...........................55
Figure III.9: Les environnements de l'outil de conception ISE 7.1....................57

Chapitre IV :

Figure IV.1: Schéma du codage du RS (7,5)..63
Figure IV.2: Schéma de l'addition en GF (2^3)..65
Figure IV.3: Schéma de multiplication en GF (2^3)...67
Figure IV.4: L'outil de programmation VHDL...67
Figure IV.5: L'outil de simulation "ISE Simulator "..68
Figure IV.6: Schéma pour le calcul du syndrome..70
Figure IV.7: Schéma avec signaux détaillés pour le calcul du syndrome S1.........71

Figure IV.8: Schéma avec signaux détaillés pour le calcul du syndrome S2.........72
Figure IV.9: Schéma du bloc Chien Search..74
Figure IV.10: Schéma avec signaux détaillés pour l'évaluation du polynôme de...76
Figure IV.11: Schéma avec signaux détaillés pour l'évaluation du polynôme d'amplitude..77
Figure IV.12: Schéma de l'algorithme de Forney détaillé..............................79
Figure IV.13: Etape d'implémentation d'un circuit logique sur FPGA ou CPLD...81
Figure IV.14: Schéma du codeur 82
Figure IV.15: Schéma du syndrome...87

LISTE DES TABLEAUX

Chapitre I :

Tableau I.1: Table de codage ASCII sur 7bits...5
Tableau I.2: Un extrait de schéma de codage ASCII...6
Tableau I.3: Schéma du LRC et VRC...10
Tableau I.4: Addition de deux éléments dans un GF (2)...19
Tableau I.5: Soustraction de deux éléments dans un GF (2)...20
Tableau I.6: Polynômes primitifs dans GF (2^m)...21

ChapitreIII :

Tableau III.1: Analogie entre la logique câblée et la logique programmable...54

ChapitreIV :

Tableau IV.1: Eléments de GF (2^3)...61
Tableau IV.2: Tableau du calcul du syndrome S_1...71
Tableau VI.3: Tableau du calcul du syndrome S_2...72
Tableau VI.4: Racine du polynôme de localisation des erreurs...76
Tableau IV.5:Calcul des racines du polynôme de localisation des erreurs...77
Tableau IV.6:Evaluation du polynôme d'amplitude...78
Tableau IV.7: Tableau d'inversion pour GF (2^3)...79
Tableau IV.8: Calcul de l'algorithme de Forney selon schéma...80

Code IV.1 : éléments dans GF (2^3) sous Matlab...61

LISTE DES ABREVIATIONS

ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Bloc
CPLD	Complex Programmable Logic Device
CRC	Cyclic Redundancy Check
EBCDIC	Extended Binary-Coded Decimal Interchange Code
FPGA	Field-Programmable Gate Array
GAL	Generic Array Logic
HDLC	High Level Data Link Control
IEEE	Institute of Electrical and Electronics Enginneers
IOB	Input Output Bloc
ISE	Integrated Software Environement
LRC	Longitudinal Redundancy Check
LSI	Large Scale Integration
LUT	Look Up Table
MSI	Medium Scale Integration
PAL	Programmable Array Logic
PLD	Programmable Logic Device
PSM	Programmable Switch Matrix
SDF	Standard Delay File
SPGA	System-Programmable Gate Array
SPLD	Simple Programmable Logic Device
SPGA	System Programmable Gate Array
SRAM	Static Random Access Memory
SSI	Small Scale Integration
VHDL	VHSIC (Very-High-Speed-Integrated-Circuit)
VLSI	Very Large Scale Integration
VRC	Vertical Redundancy Check

INTRODUCTION GENERALE

INTRODUCTION GENERALE

De nos jours, nous vivons dans un monde où les communications jouent un rôle primordial tant par la place qu'elles occupent dans le quotidien de chacun, que par les enjeux économiques et technologiques dont elles font l'objet. Nous avons sans cesse besoin d'augmenter les débits de transmission tout en gardant ou en améliorant la qualité de ceux-ci. Mais sans un souci de fiabilité, tous les efforts d'amélioration seraient vains car cela impliquerait forcément à ce que certaines données soient retransmises. C'est dans la course au débit et à la fiabilité que les codes correcteurs entrent en jeu...

Un code correcteur d'erreur permet de corriger une ou plusieurs erreurs dans un mot-code en ajoutant aux informations des symboles redondants, autrement dits, des symboles de contrôle. Différents codes possibles existent mais dans ce mémoire on traitera seulement les codes de Reed–Solomon car pour le moment, ils représentent le meilleur compromis entre efficacité (symboles de parité ajoutés aux informations) et complexité (difficulté de codage).

Les codes de Reed–Solomon sont des codes non binaires qui travaillent dans les « champs de Galois ». Au début, on tentera d'expliquer brièvement les mathématiques qui se cachent derrière la théorie de Reed – Solomon. Ensuite, on traitera cette théorie par des mathématiques simples et la circuiterie hardware correspondante. Cette dernière sera le plus souvent suivie par des exemples consacrés à la compréhension des circuits logiques utilisés.

La théorie présentera une méthode de décodage des codes de Reed–Solomon. La solution présentée sera la méthode de la division Euclidienne.

Un chapitre sera consacré à la réalisation hardware des circuits traités en théorie. L'implémentation hardware sera effectuée en utilisant des circuits logiques, le tout traduit en langage VHDL. L'implémentation VHDL sera traitée à l'aide du logiciel Xilinx 7.1i. L'implémentation hardware sera toujours suivie d'une simulation temporelle effectuée à l'aide du Simulateur « ISE Simulator » du logiciel précédant.

CHAPITRE I

Dans ce chapitre :

- Introduction
- Le Codage dans l'informatique
- Pourquoi coder ?
- L'histoire du codage
- Le codage des caractères
- Le contrôle d'erreurs : Codes de contrôle
- Outils mathématiques
- Conclusion

I-1/ Introduction

L'étude de la transmission de messages entre un émetteur et un récepteur via un canal de transmission introduit les questions suivantes :

- Le canal permet de transmettre des signaux (souvent un signal électrique binaire). Il s'agit alors de transformer le message à émettre (par exemple un texte en français) en une séquence de signaux : on parle de codage. Le récepteur doit être capable de décoder la séquence de signaux reçus pour pouvoir lire le message émis.
- Le médium (le canal de transmission) peut introduire des erreurs : certains signaux émis peuvent être perdus ou altérés lors de la transmission. Dans ce cas, pour que le récepteur puisse décoder correctement le message reçu, on utilise des codages spécifiques, permettant de détecter, voir corriger les erreurs. On parle de code détecteur/ correcteur d'erreurs.
- Le canal est généralement partagé par plusieurs émetteurs et récepteurs. Pour qu'un émetteur puisse envoyer un message qui ne puisse être lu que par un récepteur spécifique (bien que la séquence de signaux associée soit visible par d'autres individus), le message doit être crypté grâce à un codage spécifique. La cryptographie étudie des codages et des protocoles permettant de communiquer des messages de manière secrète, de signer des documents ou d'authentifier un émetteur.

Le premier chapitre de ce mémoire va donc s'intéresser au codage des informations.

I-2/ Le Codage dans l'informatique

Vers la fin des années 30, Claude Shannon démontra qu'à l'aide de "contacteurs" (interrupteurs) fermés pour vrai et ouverts pour faux ; il était possible d'effectuer des opérations logiques en associant le nombre 1 pour vrai et 0 pour faux.

Ce codage de l'information à deux états est nommé base binaire. C'est grâce à ce codage que fonctionnent les ordinateurs. Il consiste à utiliser deux états électriques différents pour différencier les deux états, le 0 et le 1, et ainsi coder les informations.

Nous allons tout d'abord faire un bref historique sur l'évolution du codage et ses applications puis nous nous intéresseront aux différents codes présents de nos jours sur

nos ordinateurs, comme le codage des caractères; les codes détecteurs d'erreurs et les codes correcteurs d'erreurs.

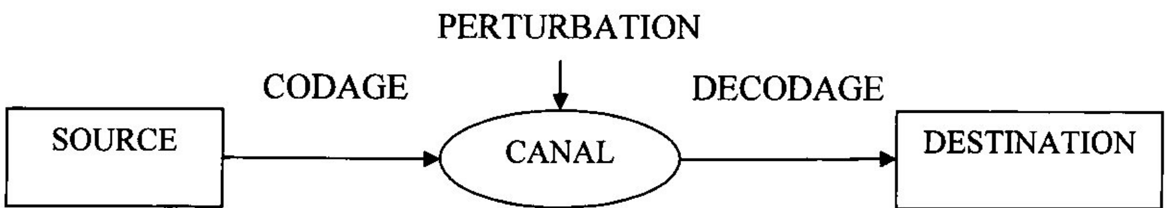


***Figure I.1** : Schéma d'une chaîne de transmission*

I-3/ Pourquoi coder ?

La communication nécessite la compréhension entre les deux entités communicantes. L'émetteur envoie l'information au récepteur qui doit savoir l'interpréter pour la comprendre. Le codage de l'information est la première étape de toute communication. Ainsi pour comprendre les 0 et les 1 de nos machines, il a bien fallu un codage et décodage pour que l'information arrive jusqu'à nos écrans de manière compréhensible, l'information a été traduite.

I-4/ L'histoire du codage

Depuis toujours le codage des informations a eu une très grande importance pour l'homme ; de l'apprentissage des signes, de la parole puis de l'écrit. Au début, l'écrit consistait essentiellement dans des dessins puis vint un alphabet plus simple à utiliser qui offrait de multiples combinaisons pour une plus grande richesse de l'expression. En réalité, les caractères de l'écrit ne sont que des symboles interprétables.

Le morse a été le premier codage à permettre une communication longue distance. C'est Samuel F.B.Morse qui l'a mis au point en 1844. Ce code est composé de points et de tirets (équivalent à un codage binaire). Il permit d'effectuer des communications beaucoup plus rapides que ne le permettait le système de courrier de l'époque aux Etats-Unis : le Pony Express. L'interpréteur était l'homme à l'époque, il fallait donc une bonne connaissance du code.

De nombreux codes furent inventés dont le code Baudot du nom de son inventeur Emile Baudot, et appelé Murray Code par les anglais.

Le 10 mars 1876, le Dr Graham Bell met au point le téléphone, une invention révolutionnaire qui permet de faire circuler de l'information vocale dans des lignes métalliques.

Ces lignes permirent l'essor des téléscripteurs, des machines permettant de coder et décoder des caractères grâce au code Baudot (les caractères étaient alors codés sur 5 bits, il y avait donc 32 caractères uniquement...).

Dans les années 60, le code ASCII (American Standard Code for Information Interchange) est adopté comme standard. Il permet le codage de caractères sur 8 bits, soit 256 caractères possibles. Nous allons donc voir dans cette partie tout d'abord comment sont codés les caractères, puis comment se déroule le contrôle d'erreurs lors de transmissions grâce aux codes détecteurs et correcteurs d'erreurs.

I-5/ Le codage des caractères

La mémoire de l'ordinateur conserve toutes les données sous forme numérique. Il n'existe pas de méthode pour stocker directement les caractères. Chaque caractère possède donc son équivalent en code numérique : c'est le code ASCII. Le code ASCII de base représentait les caractères sur 7 bits, c'est-à-dire 128 caractères possibles, de 0 à 127.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	**NUL**	**SOH**	**STH**	**ETH**	**EOT**	**ENQ**	**ACK**	**BEL**	**BS**	**HT**	**LF**	**VT**	**FF**	**CR**	**SO**	**SI**
1	**DLE**	**DC1**	**DC2**	**DC3**	**DC4**	**NAK**	**SYN**	**ETB**	**CAN**	**EM**	**SUB**	**ESC**	**FS**	**GS**	**RS**	**US**
2	SPC	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	O
7	p	q	r	s	t	u	v	w	x	y	z	{	/	}	~	**DEL**

Tableau I.1 *: Table de codage ASCII sur 7bits*

Pour bien lire le tableau, il faut construire le code hexadécimal en prenant d'abord le digit de la ligne, puis le digit de la colonne. Par exemple, la lettre 'n' a pour code hexadécimal 6E. Les caractères en gras sont les caractères non imprimables appelés caractères de contrôle car ils permettent de faire des actions telles que : retour à la ligne (CR), bip sonore (BEL). Les codes 65 à 90 représentent les majuscules. Les codes 97 à 122 représentent les minuscules. Il suffit donc de modifier le 6ème bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale.

I-5-1/ Codage ASCII étendu

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Pour coder ce type de caractère, le code ASCII à 7 bits ne suffit pas, il faut recourir à un autre codage. Il a donc été étendu à 8 bits (un octet) pour pouvoir coder plus de caractères, d'où son nom code ASCII étendu. Ce code attribue les valeurs 0 à 255 aux lettres majuscules et minuscules, aux chiffres, aux marques de ponctuation et aux autres symboles (Les caractères accentués dans le cas du code iso-latin1).

Le code ASCII étendu n'est pas unique et dépend fortement de la plateforme utilisée, d'où tous les problèmes de caractères changés lors d'un passage d'une plateforme à une autre.

A	01000001	**J**	01001010	**S**	01010011
B	01000010	**K**	01001011	**T**	01010100
C	01000011	**L**	01001100	**U**	01010101
D	01000100	**M**	01001101	**V**	01010110
E	01000101	**N**	01001110	**W**	01010111
F	01000110	**O**	01001111	**X**	01011000
G	01000111	**P**	01010000	**Y**	01011001
H	01001000	**Q**	01010001	**Z**	01011010
I	01001001	**R**	01010010	**espace**	00100000

Tableau I.2 *: Un extrait de schéma de codage ASCII*

Par exemple, le codage ASCII du message : UNIVERSITE DE BLIDA est la chaîne :

01010101 01001110 01001001 01010110 01000101 01010010 01010011 01001001 01010100 01000101 00100000 01000100 01000101 00100000 01000010 01001100 01001001 01000100 01000001

I-5-2/ Autres codes existants

Malgré le fait que le code ASCII soit le standard pour le codage de caractères, il en existe bien d'autres. Les plus connus sont le code EBCDIC (**E**xtended **B**inary-**C**oded **D**ecimal **I**nterchange **C**ode), développé par IBM, qui permet de coder des caractères sur 8 bits et le code Unicode sur 16 bits mis au point en 1991 qui permet de représenter n'importe quel caractère indépendamment de tout système d'exploitation ou langage de programmation mais qui a le gros désavantage de doubler la taille de n'importe quel fichier de texte.

I-6/ Le contrôle d'erreurs : Codes de contrôle

Le codage binaire est très pratique pour une utilisation dans des appareils électroniques tels qu'un ordinateur, dans lesquels l'information peut être codée grâce à la présence ou non d'un signal électrique. Cependant le signal électrique peut subir des perturbations (rayonnements électromagnétiques, distorsion, présence de bruit), notamment lors du transport des données sur un long trajet et transmettre des bits erronés. Ainsi, le contrôle de la validité des données est nécessaire pour certaines applications (professionnelles, bancaires, industrielles, ...).

C'est pourquoi il existe des mécanismes permettant de garantir un certain niveau d'intégrité des données, c'est-à-dire de fournir au destinataire une assurance que les données reçues sont bien similaires aux données émises. La protection contre les erreurs peut se faire de deux façons :

- Soit en fiabilisant le support de transmission, c'est-à-dire en se basant sur une protection physique ;
- Soit en mettant en place des mécanismes logiques de détection et de correction des erreurs.

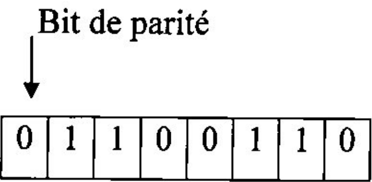
La plupart des systèmes de contrôle d'erreurs au niveau logique sont basés sur un ajout d'information (on parle de redondance) permettant de vérifier l'intégrité des données. On appelle somme de contrôle cette information supplémentaire. Il existe deux sortes de code de contrôle : les codes détecteurs qui détectent seulement l'erreur et les codes correcteurs qui eux détectent puis corrigent l'erreur. Il est beaucoup plus difficile de construire un code correcteur qu'un code détecteur d'erreurs.

I-6-1/ Les Codes détecteurs

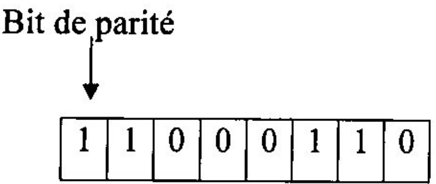
Il existe plusieurs méthodes de détection et /ou de correction d'erreurs dans le monde des communications, allant du plus simple au plus complexe. Le contrôle de parité, appelé parfois VRC (Vertical Redundancy Check), est un des systèmes de contrôle les plus simples. Il consiste à ajouter un bit supplémentaire (appelé bit de parité) à un certain nombre de bits de données appelé mot de code (généralement 7 bits, pour former un octet avec le bit de parité) dont la valeur (0 ou 1) est telle que le nombre total de bits à 1 soit pair. Pour être plus explicite il consiste à ajouter un 1 si le nombre de bits du mot de code est impair, 0 dans le cas contraire.

Prenons l'exemple suivant, on veut transmettre la séquence 1100110 (qui correspond dans le codage ASCII au caractère f)

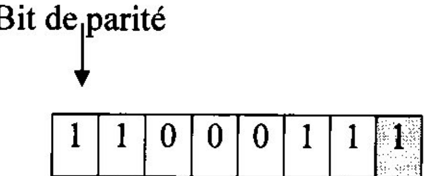
Dans cet exemple, le nombre de bits de données à 1 est pair, le bit de parité est donc positionné à 0.



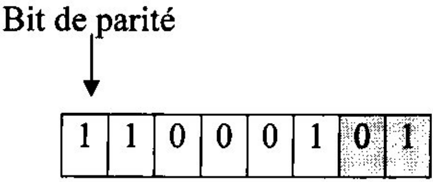
Maintenant on veut transmettre 1000110 (ce qui correspond maintenant au caractère F (changement du 6ème bit)), les bits de données égaux à 1 étant en nombre impair, le bit de parité sont ainsi positionnés à 1.



Imaginons ; désormais qu'après transmission le bit de poids faible (le bit situé à droite) de l'octet précédent soit victime d'une interférence :



Le bit de parité ne correspond alors plus à la parité de l'octet : une erreur est détectée. Toutefois, si deux bits (ou un nombre pair de bits) venaient à se modifier simultanément lors du transport de données, aucune erreur ne serait alors détectée...



Le système de contrôle de parité ne détectant que les erreurs en nombre impair, il ne permet donc de détecter que 50% des erreurs. Ce système de détection d'erreurs possède également l'inconvénient majeur de ne pas permettre de corriger les erreurs détectées (le seul moyen est d'exiger la retransmission de l'octet erroné...).

Le contrôle de parité croisé (aussi appelé contrôle de redondance longitudinale, noté LRC) consiste non pas à contrôler l'intégrité des données d'un caractère, mais à contrôler l'intégrité des bits de parité d'un bloc de caractères. Comme avant, à chaque octet est ajouté un bit de parité horizontale (LRC). Mais en plus, chaque groupe d'octets est aligné pour un contrôle de parité verticale (VRC). A tous les bits de rang 0 , on ajoute Un bit de parité, puis de même aux bits de rang 1, etc. A la fin, le contrôle de parité du contrôle vertical doit être le même que celui du contrôle horizontal.

Lettre	Code ASCII (sur 7 bits)	Bit de parité (LRC)
V	1010110	0
A	1000001	0
L	1001100	1
R	1010010	1
O	1001111	1
S	1010011	0
E	1000101	1
VRC	1010000	0

***Tableau I.3** : schéma du LRC et VRC*

Les parités du LRC et du VRC sont égales, il n'y a donc pas d'erreur apparente. Une autre méthode beaucoup utilisée est le CRC (**C**yclic **R**edundancy **C**heck). Le contrôle de redondance cyclique est un moyen de contrôle d'intégrité des données puissant et facile à mettre en oeuvre. Il représente la principale méthode de détection d'erreurs utilisée dans les télécommunications.

I-6-1-1/ Principe du contrôle CRC

L'idée du CRC est de transmettre, conjointement aux données que l'on désire envoyer, une valeur calculée par l'émetteur à partir des données à transmettre et basée sur la division de polynômes binaires.

Ensuite, l'émetteur envoie les données et le CRC calculé. A la réception, les données sont lues et le récepteur effectue le même calcul de son côté avec le même algorithme. Si le résultat est identique à celui qui a été transmis, alors on est sûr à plus de 99.9% (dépend des algorithmes employés) que les données sont correctes et donc que le message n'a pas été altéré. Sinon, il y a eu une ou plusieurs erreurs lors de la transmission.

Le principe du CRC consiste à traiter les séquences binaires comme des polynômes binaires, c'est-à-dire des polynômes dont les coefficients correspondent à la séquence binaire. Ainsi la séquence binaire 0110101001 peut être représentée sous la forme polynomiale suivante :

$$0 * x^9 + 1 * x^8 + 1 * x^7 + 0 * x^6 + 1 * x^5 + 0 * x^4 + 1 * x^3 + 0 * x^2 + 0 * x^1 + 1 * x^0$$

Soit :

$$x^8 + x^7 + x^5 + x^3 + x^0$$

Ou encore :

$$x^8 + x^7 + x^5 + x^3 + 1$$

De cette façon, le bit de poids faible de la séquence représente le degré 0 du polynôme, le 4ème bit en partant de la droite représente le degré 3 du polynôme (x^3) et ainsi de suite. Une séquence de n bits constitue donc un polynôme de degré maximal $n-1$. Toutes les expressions polynomiales sont manipulées par la suite avec une arithmétique modulo 2.

Dans ce mécanisme de détection d'erreur, un polynôme prédéfini (appelé polynôme générateur et noté $g(x)$) est connu de l'émetteur et du récepteur. La détection d'erreur consiste pour l'émetteur à effectuer un algorithme sur les bits de la trame afin de générer un CRC, et de transmettre ces deux éléments au récepteur. Il suffit alors au récepteur d'effectuer le même calcul afin de vérifier que le CRC est valide. Le contrôle d'erreur CRC est très précis, si un seul bit est incorrect, la valeur du CRC sera invalide.

I-6-1-2/ Algorithme pour le calcul du CRC

Le calcul va utiliser une opération de base : la division des polynômes.

Appelons $M(x)$ le polynôme représentant les données à coder. Son degré est représenté par m.Chacune des deux extrémités connaissent un polynôme de degré k, appelé générateur, que l'on nommera $G(x)$.

La première étape pour l'émetteur consiste à multiplier $M(x)$ par 2^k.

Le résultat

$$P(x) = 2^k * M(x)$$

est un polynôme de degré $m+k$. En fait, cela revient à rajouter k zéros à droite de $M(x)$. Cela permet de garantir que la soustraction entre $P(x)$ et un polynôme de degré k est toujours positive.

Ensuite, il effectue la division euclidienne de $P(x)$ par $G(x)$. Le calcul fournit : un Quotient $Q(x)$, et un Reste $R(x)$ de degré $k-1$.

On a alors :

$$P(x) = Q(x) * G(x) + R(x).$$

Le message à transmettre est alors *M(x)* auquel on ajoute en queue cette fois-ci le reste de la division *R(x)*.

$$T(x)=R(x) \oplus 2^k * M(x) \quad (T(x) \text{ est donc divisible par } G(x)).$$

$$\begin{aligned} T(x) / G(x) &= (R(x) \oplus 2^k * M(x)) / G(x) \\ &= (R(x) \oplus Q(x) * G(x) + R(x)) / G(x) \\ &= Q(x) + (R(x) \oplus R(x)) / G(x) \\ &= Q(x) \end{aligned}$$

(Car pour tout nombre binaire x, $x \oplus x = 0$)

Le récepteur reçoit le message. Il lui suffit de calculer le CRC du bloc transmis : $T(x) / G(x)$. Si le reste est égal à 0, il n'y a pas d'erreur de transmission, tout autre reste indique une ou plusieurs erreurs.

I-6-1-3/ Les polynômes générateurs

Au plus la probabilité d'apparition d'une erreur est importante ou la longueur du bloc à protéger est importante, au plus on utilise un polynôme générateur de degré important. Les polynômes générateurs les plus couramment employés pour le calcul du CRC sont :

CRC-12 : $x^{12} + x^{11} + x^3 + x^2 + x + 1$; Pour les caractères codés sur 6 bits.

CRC-16: $x^{16} + x^{15} + x^2 + 1$

CRC CCITT V41: $x^{16} + x^{12} + x^5 + 1$; Ce code est notamment utilisé dans la procédure HDLC (High Level Data Link Control).

CRC ARPA : $x^{24} + x^{23} + x^{17} + x^{16} + x^{15} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^5 + x^3 + 1$

CRC-32 (Ethernet):= $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

I-6-2/ Les Codes correcteurs

Le calcul CRC à beau être très performant pour trouver des erreurs, il est incapable de les corriger. D'où la nécessité dans certaines situations de trouver des codes capables non seulement de détecter des erreurs mais aussi de pouvoir les corriger sans avoir à retransmettre le message original. Le principe est d'ajouter suffisamment de redondance pour pouvoir récupérer des erreurs n'importe où dans le message.

L'inconvénient est que toutes les transmissions sont alors alourdies par cette redondance (longueur du message et calculs lors de la réception). Ces codes sont surtout utilisés dans les situations où la fiabilité est essentielle à tout moment comme par exemple dans les liaisons satellites. Ainsi grâce aux codes correcteurs, les lecteurs de CD ROM (Audio) performants peuvent lire un CD sur lequel il y a une rayure sans que le son en soit affecté.

I-6-2-1/ Le codage à triple répétition

C'est un code très simple, qui consiste à répéter trois fois chaque mot. Il permet évidemment de corriger une erreur dans chaque symbole de donnée, mais est très coûteux, puisqu'il multiplie par 3 la taille des données.

I-6-2-2/ Codes de Hamming

Les codes de Hamming partent du principe où les bits de contrôle sont situés aux bits de puissance de 2, les autres étant les bits de données. Donc suivant la longueur du mot à coder, on a plus ou moins de bits de contrôle de Hamming.

Prenons pour exemple, un code de Hamming [7, 4, 3]. Il prend donc en entrée des mots de 4 bits de données, et ajoute 3 bits de contrôle pour donner des mots de 7 bits.

Plus précisément, si l'on veut transmettre un message formé des bits $B_1B_2B_3B_4$, on envoie $C_1C_2B_1C_3B_2B_3B_4$ avec :

$$C_1=B_1\oplus B_2\oplus B_4$$

$$C_2=B_1\oplus B_3\oplus B_4$$

$$C_3=B_2\oplus B_3\oplus B_4$$

Les bits de donnée B_1, B_2 et B_3 apparaissent dans deux équations, le bit de donnée B_4 dans trois.

Pour le décodage on est confronté à plusieurs cas : Aucune équation fausse : pas d'erreur (ou plusieurs qui se compensent).

- Une équation fausse : seuls les bits de contrôles interviennent dans une seule équation, c'est donc le bit de contrôle en cause qui est erroné, le message est correct.

- Deux équations fausses : seuls les bits de donnée B_1, B_2 et B_3 interviennent dans deux équations exactement, c'est donc le bit de donnée commun aux deux équations fausses qui est erroné.

- Trois équations fausses : seul le bit de donnée B_4 intervient dans les trois équations, c'est lui qui est erroné.

Le codage de Hamming reste cependant médiocre par rapport aux critères de Varshamov et Gilbert (borne optimale pour un code correcteur d'erreur), c'est pourquoi de nouveaux codes correcteurs ont été recherchés et sont en train de voir effectivement le jour.

I-6-2-2-1/ Un exemple d'utilisation

On veut transmettre le message 1011001. Pour cela on insère donc les bits de contrôle aux positions des puissances de 2 comme indiqué sur le schéma.

11	10	9	8	7	6	5	4	3	2	1
1	0	0	*x*	1	1	0	*x*	1	*x*	*x*

On considère les bits de données égaux à 1 et la représentation en base deux de leurs indices. On les additionnent comme suit :

$$
\begin{array}{rcl}
11 = & 1\ 0\ 1\ 1 \\
7 = & 0\ 1\ 1\ 1 \\
6 = & 0\ 1\ 1\ 0 \\
3 = & \underline{0\ 0\ 1\ 1} \\
= & 1\ 0\ 0\ 1
\end{array}
$$

Le code de Hamming ajouté est donc ce résultat : 1001.

Le code transmis est le suivant :

11	10	9	8	7	6	5	4	3	2	1
1	0	0	1	1	1	0	0	1	0	1

Le receveur reçoit le message et passe à la vérification :

Il considère les bits de données et de contrôle qui sont égaux à 1 et calcule leur représentation en base 2.

11 = 1 0 1 1

8 = 1 0 0 0

7 = 0 1 1 1

6 = 0 1 1 0

3 = 0 0 1 1

1 = 0 0 0 1

= 0 0 0 0

La somme de contrôle est égale à 0 → il n'y a pas eu d'erreur de transmission, le message est valide.

Maintenant voyons ce qu'il se passe si le message est corrompu.

Changeons le bit N° 11 en 0 comme ceci :

11	10	9	8	7	6	5	4	3	2	1
	0	0	1	1	1	0	0	1	0	1

On refait le même calcul que précédemment en ajoutant les indices des bits à 1.

8 = 1 0 0 0

7 = 0 1 1 1

6 = 0 1 1 0

3 = 0 0 1 1

1 = 0 0 0 1

= 1 0 1 1

Le contrôle d'erreur n'est plus égal à 0 ➔ Il y a eu une erreur.
Hamming nous donne même l'endroit où se situe l'erreur 1011 = 11. L'erreur se situe au bit No 11.
Par contre si le message comporte 2 erreurs, Hamming détectera qu'il y a eu un problème mais ne pourra le corriger ; il ne pourra détecter les erreurs sur plus de 2 bits de transmission (valide seulement pour cet exemple, c'est à dire avec cette longueur de message).

I-6-2-3/ Les codes de Reed-Solomon

Les codes de Reed-Solomon sont des codes correcteurs basés sur les erreurs de blocs. Ils ont beaucoup d'utilisation incluant :
– périphériques de stockage ; lecteurs cassettes, CD, DVD, codes-barres...
– communications radios ou mobiles comme les téléphones portables.
– communications satellites.
– télévision numérique.
– modems haut-débit comme ADSL, xDSL ...

Le principe est de remplacer l'alphabet binaire par un alphabet q-aire. Typiquement $q = 256$ si on utilise des bytes.

On veut transmettre un message composé de m symboles q-aires.

Ce message est représenté comme un polynôme P de degré $m - 1$.

$$P(x) = m_0 + m_1 x + m_2 x^2 + \ldots + m_{m-1} x^{m-1}$$

Plutôt que de transmettre les coefficients m_0 jusqu'à m_{m-1}, on pourrait évaluer le polynôme $P(x)$ pour m valeurs distinctes de x.

Pour corriger e erreurs, on transmet $n = m + 2 * e$ évaluations de couples $(x_i, y_i = P(x_i))$, et donc on utilise $k = 2 * e$ symboles supplémentaires. Notons qu'il faut $n = q$.

Pour décoder, on cherche un sous-ensemble de $m + e$ points (x_i, y_i) par lesquels passe un polynôme de degré $m - 1$. Si on suppose qu'il y a au maximum e erreurs, il y a au moins m points communs avec $P(x)$ et les polynômes sont identiques.

I-7/ Outils mathématiques

I-7-1/ Introduction

On consacrera cette partie à une brève introduction aux mathématiques utilisées dans les codes de Reed – Solomon. On introduira la notion de groupe, d'anneau et de champ dans les mathématiques abstraites. Par la suite, on se concentrera sur les « champs de Galois » utilisés pour les codes de Reed – Solomon.

I-7-2/ Groupe [1]

Un groupe G est un ensemble d'éléments défini par l'opération binaire • ; Chaque paire d'éléments, a et b, a un seul élément c défini par l'opération binaire • dans G, $c=a.b$

Un groupe doit respecter les contraintes suivantes:

- Le groupe doit être fermé, pour n'importe quel élément a et b dans G, l'opération binaire $c=a\bullet b$ doit toujours retourner un élément dans le même groupe G.
- La loi associative doit être applicable ; $a\,(bc) = (ab)\,c$ pour tout a, b, c compris dans G.

- Pour n'importe quel élément a dans G, il doit exister un élément tel que $a.e = e.a = a$. L'élément e est appelé l'élément identité de G.
- Pour tout élément a dans G, il doit exister son élément inverse a' tel que $a\,.\,a' = e$.

Un groupe est dit « abélien » si la loi de commutation est valable, $ab = ba$ pour tous les éléments compris dans G.

I-7-3/ Anneau [2] :

Un anneau R est un ensemble d'éléments défini par deux opérations binaires appelées, addition et multiplication. Un anneau R est un groupe « abélien » selon l'opérateur d'addition. La multiplication doit respecter les contraintes suivantes:

- La multiplication doit être associative, $a\,(bc) = (ab)\,c$ pour tout a, b, c compris dans R.
- La multiplication doit être distributive par rapport à l'addition, $a\,(b+c) = ab + ac$ et $(a+b)\,c = ac + bc$ pour tout a, b, c compris dans R.

Un anneau est dit commutatif si sa multiplication est commutative, $ab = ba$ pour tout a, b compris dans R.

Un anneau est appelé domaine d'intégrité s'il possède un élément unité pour la multiplication et qu'il n'admet pas de division par zéro.

- Il y a un élément unité pour la multiplication tel que $a.1=1.a=a$ pour tout a compris dans R.
- Pour tout a, b dans R, $ab = 0$, il y a alors la possibilité que $a = 0$ ou $b = 0$.

I-7-4/ Corps ou champs

Un corps ou champ C est un domaine d'intégrité dans lequel tous les éléments non nuls sont inversibles. Pour un a non nul dans C, il existe un élément a^{-1} dans C tel que $aa^{-1} = 1$. Un corps possède toutes les propriétés définies aux sous-chapitres **(I-7-2)** et **(I-7-3)**

I-7-5/ Champs de Galois [3]

Les « champs de Galois » font partie d'une branche particulière des mathématiques qui modélise les fonctions du monde numérique. Ils sont à la source de nombreuses applications de transmission, parmi lesquelles le codage, comme le codage correcteur d'erreurs. Les deux types d'opérations réalisables dans un champ de Galois sont l'addition et la multiplication. Combinées, elles permettent de calculer des codes (par exemple, codage en bloc du Reed-Solomon). Or les champs de Galois offrent une structure naturellement adaptée au traitement numérique binaire de l'information : un registre de m bits représente un élément de $GF(2^m)$. Une unité arithmétique capable de calculer efficacement des produits et des sommes sur $GF(2^m)$ permet ainsi de réaliser en matériel les fonctions élémentaires de codage en transmission numérique.

La dénomination « champ de Galois » provient du mathématicien français Galois qui en a découvert les propriétés fondamentales.

Il y a deux types de champs, les champs finis et les champs infinis. Les « champs de Galois » finis sont des ensembles d'éléments fermés sur eux-mêmes. L'addition et la multiplication de deux éléments du champ donnent toujours un élément du champ fini.

I-7-5-1/ Elément des champs de Galois

Un « champ de Galois » consiste en un ensemble de nombres, ces nombres sont constitués à l'aide de l'élément base α comme suit :

$$0,\ 1,\ \alpha,\ \alpha^2,\ \alpha^3 \ldots \alpha^{N-1} \qquad \textbf{(I-1)}$$

En prenant $N = 2^m - 1$, on forme un ensemble de 2^m éléments. Le champ est alors noté $GF(2^m)$.

$GF(2^m)$ est formé à partir du champ de base $GF(2)$ et contiendra des multiples des éléments simples de $GF(2)$.

En additionnant les puissances de α, chaque élément du champ peut être représenté par une expression polynomiale du type :

$$\alpha^{m-1} x^{m-1} + \alpha^{m-2} x^{m-2} + \ldots + \alpha x + \alpha^0 \qquad \textbf{(I-2)}$$

Avec : $\alpha^{m-1} \ldots \alpha^0$: éléments bases du $GF(2)$ (valeurs : 0,1)

Sur les « champs de Galois », on peut effectuer toutes les opérations de base :

- L'addition dans un champ fini $GF(2)$ correspond à faire une addition modulo 2 «XOR»
- La soustraction effectuera la même opération qu'une addition, c'est à dire, la fonction logique«XOR».
- La multiplication et la division seront des opérations modulo « grandeur du champ », donc $mod\ (2^{m-1})$.

I-7-5-1-1/ Addition dans *GF(2)*

Dans ce sous-chapitre, on cherche à comprendre comment l'affirmation du chapitre I-7-5-1 est possible. Considérons le tableau ci-dessous dans lequel'on fait l'addition binaire entre les deux éléments A et B du $GF(2)$:

A	*B*	Reste	Résultat
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

***Tableau I-4**: addition de deux éléments dans un GF (2)*

En négligeant le reste dans le résultat final de l'addition, on constate que la somme entre deux éléments dans un *GF(2)* donne une addition modulo2, c'est-à-dire, une fonction logique « XOR ». Comme *GF(2)* est le champ de base, cette relation sera valable pour Tous les champs dérivés, c'est-à-dire, pour $GF(2^m)$.

I-7-5-1-2/ Soustraction dans *GF(2)*

Ici, on cherche à comprendre comment une soustraction dans *GF(2)* correspond à faire une addition dans le même champ. Considérons le tableau suivante dans lequel on fait la soustraction binaire entre les deux éléments *A* et *B* :

A	*B*	Emprunte	Résultat
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

***Tableau I-5**: soustraction de deux éléments dans un GF (2)*

On constate que la soustraction dans *GF(2)* effectue la même opération que l'addition dans le même champ, c'est-à-dire une opération logique « XOR ».

I-7-5-2/ Polynôme primitif

Ce polynôme permet de construire le « champ de Galois » souhaité. Tous les éléments non nuls du champ peuvent être construits en utilisant l'élément α comme racine du polynôme primitif. Chaque *m* a peut être plusieurs polynômes primitifs *P(x)*, mais dans le tableau ci-dessous, on mentionne seulement les polynômes ayant le moins d'éléments. Les polynômes primitifs pour les principaux « champs de Galois » sont les suivants:

m	$P(x)$	m	$P(x)$
3	$1+x+x^3$	14	$1+x+x^6+x^{10}+x^{14}$
4	$1+x+x^4$	15	$1+x+x^{15}$
5	$1+x^2+x^5$	16	$1+x+x^3+x^{12}+x^{16}$
6	$1+x+x^6$	17	$1+x^3+x^{17}$
7	$1+x^3+x^7$	18	$1+x^7+x^{18}$
8	$1+x^2+x^3+x^4+x^8$	19	$1+x+x^2+x^5+x^{19}$
9	$1+x^4+x^9$	20	$1+x^3+x^{20}$
10	$1+x^3+x^{10}$	21	$1+x^2+x^{21}$
11	$1+x^2+x^{11}$	22	$1+x+x^{22}$
12	$1+x+x^4+x^6+x^{12}$	23	$1+x^5+x^{23}$
13	$1+x+x^3+x^4+x^{13}$	24	$1+x+x^2+x^7+x^{24}$

Tableau I-6**: polynômes primitifs dans GF (2^m)[4]

I-7-6/ Dérivation formelle d'un polynôme dans un champ de Galois

Un polynôme d'ordre m est défini comme:

$$f(x) = f_m x^m + f_{m-1} x^{m-1} + \ldots\ldots + f_1 x + f_0 \quad \textbf{(I-3)}$$

La dérivée formelle **[5]** du polynôme **(I-3)** dans un « champ de Galois » $GF(2^m)$ est définie avec les puissances paires de **(I-3)**:

$$f'(x) = f_1 + 2f_2 x + 3 f_3 x^2 + \ldots\ldots + m f_m x^{m-1}$$

$$f'(x) = f_1 + 3f_3 x^2 + 5 f_5 x^4 + \ldots\ldots$$

La dérivée est définie seulement par les puissances paires car les puissances impaires sont précédées de coefficients pairs qui annulent les termes:

$2 = 1+1 = 0$

$4 = 2+2 = 1+1+1+1 = 0$

....

La dérivation doit respecter les règles suivantes :

- Si $f^2(x)$ divise $a(x)$ alors $f(x)$ divise $a'(x)$.
- La dérivation est une opération linéaire, elle doit respecter la règle de linéarité suivante :

$$[f(x)\, a(x)]' = f'(x)\, a(x) + f(x)\, a'(x).$$

I-8/ Conclusion

Les codes correcteurs d'erreurs sont donc encore un sujet de recherche active, visant principalement une mise en oeuvre efficace. Ils sont peu connus mais très utilisé, car la fiabilité des transmissions et du stockage est une nécessité.

CHAPITRE II

PRINCIPE PRATIQUE DE REED-SOLOMON

Dans ce chapitre :

- Introduction
- Propriété des codes Reed-Solomon
- Codage
- Décodage
- Conclusion

II-1 /Introduction

Les codes de Reed – Solomon sont des codes de détection et de correction des erreurs. Se sont des codes particuliers des codes BCH.

Les messages sont divisés en blocs dont on a ajouté des informations redondantes à chaque bloc permettant ainsi de diminuer la possibilité de retransmission. La longueur des blocs dépend de la capacité du codeur.

Le décodeur traite chaque bloc et corrige les éventuelles erreurs. A la fin de ce traitement, les données originales seront restaurées.

Typiquement, la transmission des données dans un canal avec cette méthode est effectuée ainsi :

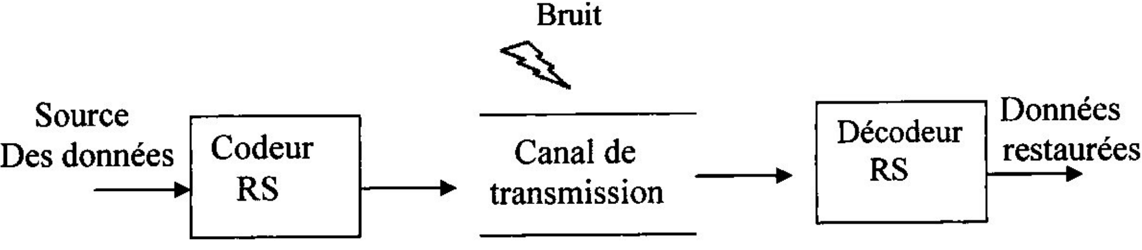


***Figure II-1**: schéma général*

II-2/ Propriété des codes Reed Solomon

Ces codes ont une propriété importante, ils sont linéaires et font partie des codes BCH.

Le codeur prend k symboles de donnée (chaque symbole contenant m bits) et calcule les informations de contrôle pour construire n symboles, ce qui donne $(n-k)$ symboles de contrôle. Le décodeur peut corriger au maximum t symboles, ou $2t = n-k$.

Le diagramme ci-dessous montre une trame constituée avec le codeur Reed – Solomon :

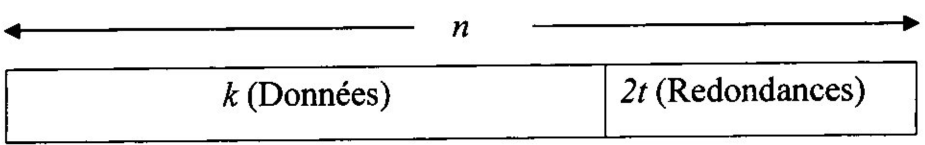


***Figure II-2**: mot-code de Reed – Salomon*

La longueur maximale d'un code de Reed – Solomon est définie comme suit:

$$n = k + 2t = 2^m - 1$$

Avec :

k : nombre de symboles d'information

$2t$: nombre de symboles de contrôle

m : nombre de bits par symbole

La distance minimale d'un code Reed – Solomon est :

$$d_{min} = 2t + 1$$

Autre propriété des codes Reed–Solomon, ils sont cycliques, c'est-à-dire, que chaque mot-code décalé engendre un autre mot-code. Tous les codes cycliques peuvent être réduits en gardant la même capacité d'erreur, mais le nouveau code formé n'est alors pas cyclique.

De plus, les codes de Reed – Solomon sont des codes non-binaires. Les codes sont représentés sur des «champs de Galois » de $GF(2^m)$ et non pas sur des champs de $GF(2)$. Les symboles sont définis comme les coefficients du polynôme et le degré de x indique l'ordre. Ainsi, le symbole avec l'ordre le plus élevé est reçu/envoyé en premier et le dernier symbole reçu/envoyé est celui dont l'ordre est moindre.

II-2-1/ Exemple de polynôme

Prenons le polynôme suivant :

$$\alpha^2 x^2 + \alpha^3 x + \alpha^6 \qquad \textbf{(II-1)}$$

Le premier coefficient entrant ou sortant du codeur/décodeur sera α^2, suivi de α^3 puis α^6.

II-3/ Codage

Le codage avec les codes de Reed – Solomon est effectué de la même façon que le codage à l'aide du CRC. La seule différence est que les codes de Reed – Solomon sont non-binaires (Reed – Solomon $GF(2^m)$), alors que le CRC est binaire, ($GF(2)$).

II-3-1/ Théorie du codage

L'équation clé définissant le codage systématique de Reed – Solomon *(n, k)* est :

$$c(x) = i(x)\, x^{n-k} + [i(x)\, x^{n-k}]\, mod(g(x)) \qquad \textbf{(II-2)}$$

Avec

$c(x)$: polynôme du mot-code, degré $n-1$

$i(x)$: polynôme d'information, degré $k-1$

$[i(x)\, x^{n-k}] mod(g(x))$: Polynôme de contrôle, degré $n-k-1$

$g(x)$: polynôme générateur, degré $n-k$

Le codage systématique signifie que l'information est codée dans le degré élevé du mot-code et que les symboles de contrôle sont introduits après les mots d'information.

II-3-1-1/ Polynôme générateur

Les symboles de contrôle sont générés à l'aide de polynômes particuliers, appelés polynômes générateurs. Tous les codes Reed – Solomon sont valables si et seulement si ils sont divisibles par leur polynôme générateur,$c(x)$ doit être divisible par $g(x)$.
Pour la génération d'un correcteur d'erreurs de t symboles, on devrait avoir un polynôme générateur de puissance α^{2t}. La puissance maximale du polynôme est déterminée grâce la distance minimale qui est $d_{min} = 2t + 1$. On devrait avoir $2t + 1$ termes du polynôme générateur.

Le polynôme générateur est sous la forme :

$$g(x) = (x - \alpha^1)(x - \alpha^2)\ldots\ldots(x - \alpha^{2t})$$

$$g(x) = g_{2t}\, x^{2t} + g_{2t-1}\, x^{2t-1} + \ldots\ldots + g_1\, x + g_0$$

II-3-2/ Implémentation hardware du codeur

Pour comprendre comment la partie hardware fonctionne, on doit comprendre la définition mathématique du codage **(II-2)** et les différentes opérations effectuées.

Le codage est systématique, on doit effectuer une opération de décalage pour placer les informations dans le degré élevé du mot-code de sortie. Mathématiquement, le décalage est effectué selon la fonction :

$$i(x)\,x^{n-k} \qquad \textbf{(II-3)}$$

Avec :

$i(x)$: polynôme d'information

x^{n-k} : Décalage du polynôme d'information de $n-k$ positions vers la gauche

Le deuxième terme de l'équation **(II-2)** est le reste de la division de $\frac{i(x)x^{n-k}}{g(x)}$

Cette division donnera les symboles de contrôle.

L'implémentation du codeur demandera deux opérations : un décalage et une division. Ces deux opérations peuvent être effectuées grâce à des registres à décalage et à des multiplexeurs.

II-3-2-1/ Schéma

Schéma général du codage:

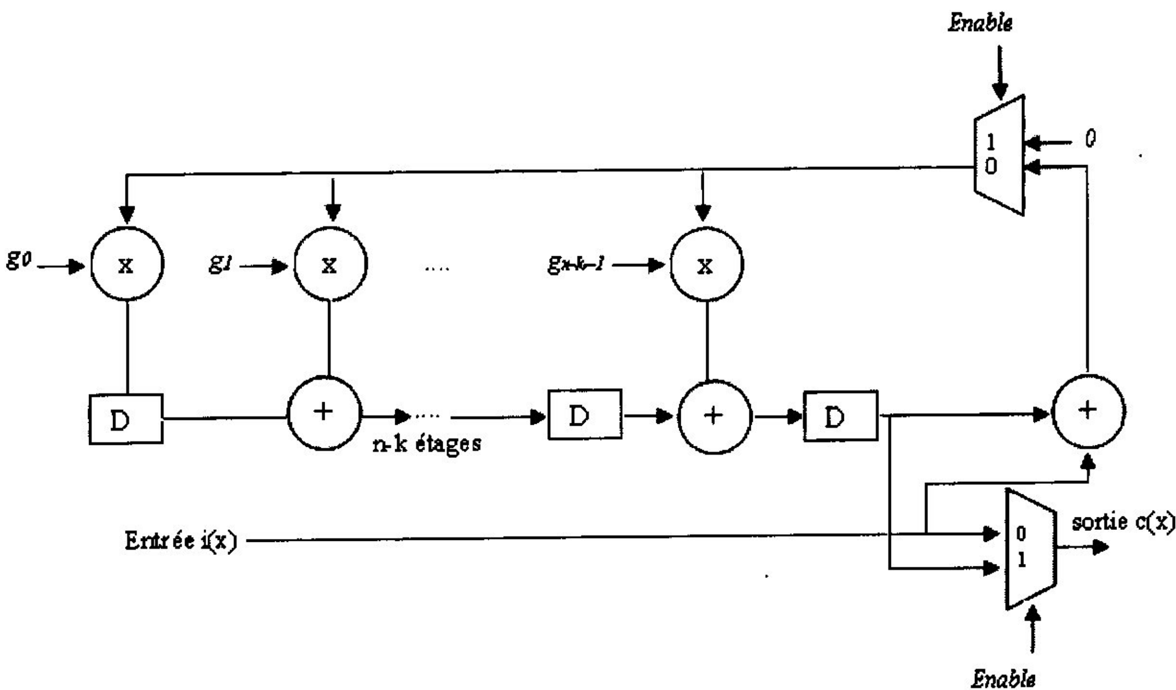


***Figure II-3**: schéma de codage* **[6]**

Avec :

$\oplus$ Addition dans le <champ de Galois>

$\otimes$ Multiplication dans le <champ de Galois>

[D] Registre à *m* bits

Les éléments utilisés pour le codage sont:

- Des additionneurs dans le « champ de Galois » $GF(2^m)$
- Des multiplicateurs dans le « champ de Galois » $GF(2^m)$
- Des multiplexeurs
- Des registres

II-3-2-1-1/ Addition

Les additions sont définies dans le « champ de Galois » $GF(2^m)$, donc pour additionner deux éléments, on prendra la notation binaire de chaque élément et on les additionnera modulo 2(est une opération logique définie par l'opérateur logique « XOR » bit à bit). L'addition entre deux éléments d'un champ fini donnera toujours un élément dans le même champ.

II-3-2-1-2/ Multiplication

Les multiplications utilisées dans les codes de Reed – Solomon sont des multiplications dans le « champ de Galois » $GF(2^m)$. La multiplication dans le « Champ de Galois » est une opération modulaire, c'est-à-dire que la multiplication entre deux éléments d'un champ fini donnera toujours un élément dans le même champ. Il y a plusieurs techniques pour effectuer ce calcul, dans le cadre de ce projet on choisira de développer la multiplication entre deux polynômes **[7]** et ensuite de normaliser le résultat par rapport au polynôme primitif du champ choisi.

II-4/ Décodage

L'idée de base du décodeur de Reed – Solomon est de détecter une séquence erronée avec peu de termes, qui sommée aux données reçues, donne lieu à un mot-code valable. Plusieurs étapes sont nécessaires pour le décodage de ces codes :

- Calcul du syndrome.
- Calcul des polynômes de localisation des erreurs et de l'amplitude.
- Calcul des racines et évaluation des deux polynômes.
- Sommation du polynôme constitué et du polynôme reçu pour reconstituer l'information de départ sans erreur.

Le schéma de décodage est montré dans la figure **(II-4)** ci-dessous :

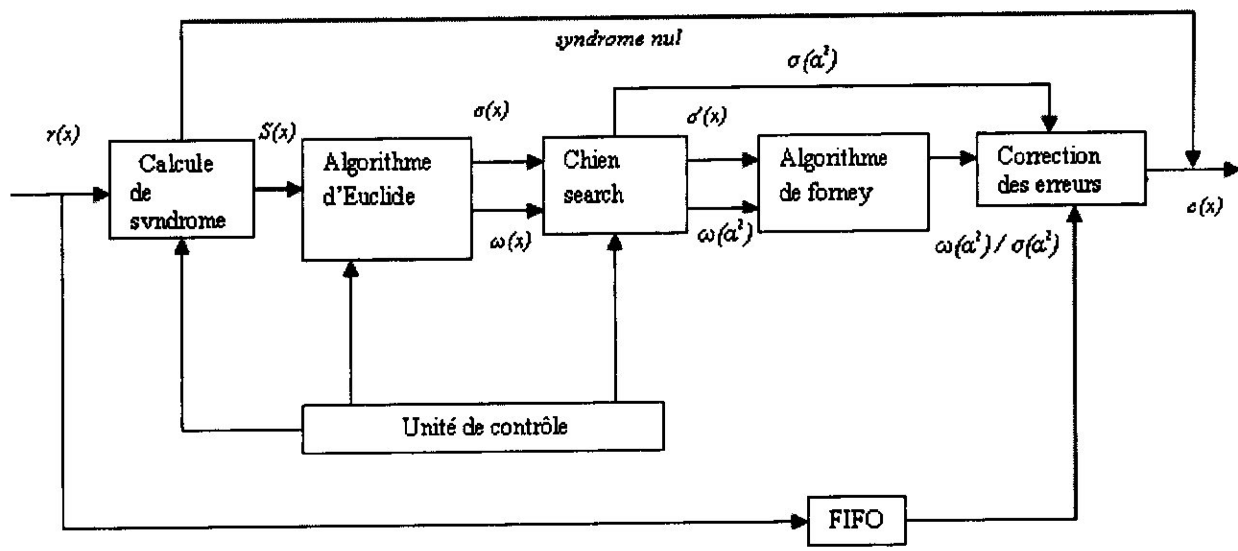


***Figure II-4**: schéma du décodage*

Avec :

r(x) : mot-code reçu

S(x) : syndrome calculé

$\omega(x)$: polynôme d'amplitude des erreurs

$\omega(\alpha_i)$: polynôme d'amplitude des erreurs évalué pour tous les éléments compris dans $GF(2^m)$

$\sigma(x)$: polynôme de localisation des erreurs

$\sigma\left(\frac{\omega(\alpha^i)}{\sigma'(\alpha^i)}\right)$: polynôme de localisation des erreurs évalué pour tous les éléments compris dans $GF(2^m)$.

$\sigma'(\alpha^i)$: dérivée du polynôme de localisation des erreurs évalué pour tous les éléments compris dans $GF(2^m)$.

$\frac{\omega(\alpha^{i})}{\sigma'(\alpha^{i})}$: Division entre le polynôme d'amplitude et la dérivée du polynôme de localisation des erreurs.

c(x) : mot-code reconstitué.

II-4-1/ Généralités du décodage

Considérons un code de Reed – Solomon $c(x)$ correspondant au code transmis et soit $r(x)$ le code que l'on reçoit. Le polynôme d'erreurs introduit par le canal est défini comme:

$$\begin{aligned} e(x) &= r(x) - c(x) \\ &= r(x) + c(x) \\ &= e_0 + e_1 x + \ldots\ldots + e_{n-1} x^{n-1} \end{aligned} \quad \textbf{(II-4)}$$

Supposant que le polynôme des erreurs contiennent v erreurs aux positions $x^{j_1}, x^{j_2}, x^{j_3}, \ldots, x^{j_v}$ avec $0 \le j_1 \le j_1 \le j_2 \le \ldots\ldots \le j_v \le n-1$. On peut donc redéfinir le polynôme des erreurs comme :

$$e(x) = e_{j_1} x^{j_1} + e_{j_2} x^{j_2} + \ldots\ldots\ldots\ldots + e_{j_v} x^{j_v} \quad \textbf{(II-5)}$$

Avec : $e_{j_1}, e_{j_2}, \ldots, e_{j_v}$: Valeurs d'amplitude des erreurs

$x^{j_1}, x^{j_2}, \ldots, x^{j_v}$: emplacements des erreurs

A partir du polynôme *r(x)* reçu, on peut calculer le polynôme du syndrome *S(x)* qui nous indiquera la présence d'éventuelles erreurs. Si tous les coefficients du syndrome sont nuls, alors les étapes suivantes du décodage n'ont pas lieu d'être car le mot-code reçu ne contiendra pas d'erreurs. Par contre, si le syndrome est non nul, on devra calculer le polynôme de localisation des erreurs et le polynôme d'amplitude des erreurs. Il y a plusieurs méthodes de calcul de ces deux polynômes, dans le cadre de ce projet on ne traitera que le décodage selon l'algorithme d'Euclide. Une fois les polynômes calculés en utilisant l'algorithme de Forney, on calculera les valeurs à soustraire pour obtenir le mot-code sans erreur.

II-4-2/ Calcul du syndrome

Le calcul du syndrome est défini comme le reste de la division entre le polynôme reçu *r(x)* et le polynôme générateur *g(x)*. Le reste indiquera la présence d'erreurs. Comme l'opération division est toujours une opération complexe par rapport à des sommes et des additions, on est amené à chercher une autre méthode pour le calcul du syndrome **[8].**

Le calcul du syndrome peut aussi être effectué par un processus itératif. Avant de pouvoir calculer le polynôme du syndrome, on doit attendre que l'on ait reçu tous les éléments du polynôme *r(x)*.

Comme :

$$r(x) = c(x) + e(x) \qquad \textbf{(II-6)}$$

On obtient :

$$S_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) \qquad \textbf{(II-7)}$$

Par la relation **(II-7)** on peut définir les différentes équations qui lient le polynôme d'erreurs au syndrome :

$$S_1 = e_{j_1}x^{j_1} + e_{j_2}x^{j_2} + \ldots\ldots\ldots\ldots + e_{j_\upsilon}x^{j_\upsilon}$$

$$S_2 = e_{j_2}x^{2j_1} + e_{j_2}x^{2j_2} + \ldots\ldots\ldots\ldots + e_{j_\upsilon}x^{2j\upsilon}$$

$$\ldots\ldots\ldots\ldots$$

$$S_{2t} = e_{j_1}x^{2tj_1} + e_{j_2}x^{2tj_2} + \ldots\ldots\ldots\ldots + e_{j_\upsilon}x^{2tj_\upsilon}$$

Le syndrome sous forme polynomiale sera :

$$S(x) = \ldots\ldots S_{2t+1}x^{2t} + S_{2t}x^{2t-1} + \ldots\ldots\ldots\ldots + S_2x + S_1$$

Seuls les premiers $2t$ symboles du syndrome sont connus.

Si le code reçu $r(x)$ n'est pas affecté par des erreurs alors tous les coefficients du syndrome seront nuls ($r(x) = c(x)$).

II-4-2-1/Schéma de calcul du syndrome

Le schéma de la figure **(II-5)** calcule un symbole du syndrome de façon itérative :

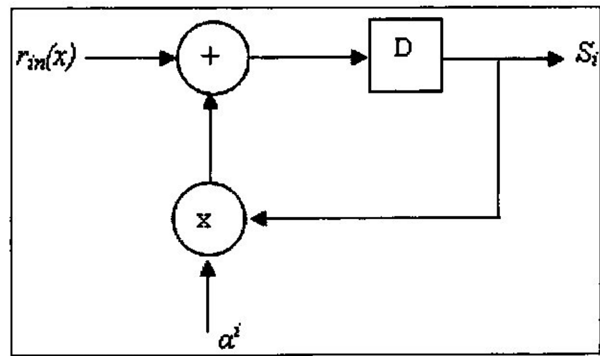


***Figure II-5**: schéma pour le calcul du syndrome*

On aura besoin de $2t$ schémas, comme celui de la figure **(II-5)**, pour avoir le syndrome complet.

II-4-3/ Euclide

II-4-3-1/ Généralité du théorème d'Euclide

L'algorithme d'Euclide **[9]** est un algorithme récursif qui permet de trouver le plus grand diviseur commun de deux polynômes $r_0(x)$ et $r_1(x)$ dans le « champ de Galois » $GF(q)$.

Il existe deux polynômes $a(x)$ et $b(x)$ en $GF(q)$ tels que :

$$MCD\,(r_0(x), r_1(x)) = a(x)\,r_0(x) + b(x)\,r_1(x)$$

Avec :

$a(x)$ et $b(x)$ peuvent être calculés selon l'algorithme d'Euclide.

En donnant deux polynômes non nuls $r_0(x)$ et $r_1(x)$ en $GF(q)$, l'algorithme d'Euclide fonctionnent de la façon suivante :

$$deg\,(r_1(x)) \leq deg\,(r_0(x))$$

$$a_0(x) = 1, \; b_0(x) = 0$$

$$a_1(x) = 1, \; b_1(x) = 1$$

Avec :

$deg\ (r_1(x))$: degré du polynôme $r_1(x)$

$deg\ (r_0(x))$: degré du polynôme $r_0(x)$

Pour $i \geq 2$, on calcule le quotient $q_i(x)$ et le polynôme restant $r_i(x)$, la division est Effectuée sur $r_{i-2}(x)$ et $r_{i-1}(x)$

Avec :

$$0 \leq deg\ (r_i(x)) \leq deg\ (r_{i-1}(x))$$
$$a_i(x) = a_{i-2}(x) - q_i(x)\ a_{i-1}(x)$$
$$b_i(x) = b_{i-2}(x) - q_i(x)\ b_{i-1}(x)$$

Les calculs se terminent lorsque $deg\ (r_i) = 0$, le dernier polynôme non nul indique le plus grand diviseur commun.

II-4-3-2/Correction d'erreur avec Euclide [10]

Le polynôme de localisation des erreurs est défini comme :

$$\sigma(x) = \prod_{k=1}^{v} (1 - \alpha^{i_k} x)$$
$$= \sigma_v x^v + \sigma_{v-1} x^{v-1} + \ldots + \sigma_1 x + 1$$

Le polynôme d'amplitude des erreurs se calculera de la façon suivante :

$$\omega(x) = S(x)\ \sigma(x)$$

Avec :

$\sigma(x)$:polynôme de localisation des erreurs, inconnu à ce stade.

$\omega(x)$:polynôme d'amplitude, inconnu à ce stade.

$S(x)$: polynôme syndrome, connu.

Comme on connait seulement $2t$ symboles du polynôme du syndrome $(x^0 \ldots x^{2t-1})$, on devrait limiter le résultat à $2t$:

$$S(x)\ \sigma(x) = \omega(x)\ mod(x^{2t}) \qquad \textbf{(II-8)}$$

Cette expression est l'équation clé pour les codes de Reed – Solomon. Si le nombre d'erreurs v dans le mot-code transmis $c(x)$ est plus petit ou égal à t, l'équation clé a une seule paire de solution $\sigma(x)$ et $\omega(x)$. Les deux degrés des polynômes doivent respecter la contrainte qui suit :

$$deg(\omega(x)) < deg(\sigma(x)) \leq t \qquad \textbf{(II-9)}$$

L'équation clé peut être résolue selon l'algorithme d'Euclide en appliquant $r_0(x) = x^{2t}$ et $r_1(x) = S(x)$. Le calcul du théorème d'Euclide nous donnera comme solution le polynôme de localisation des erreurs et le polynôme d'amplitude.

L'algorithme d'Euclide, pour le calcul du polynôme de localisation des erreurs et le polynôme d'amplitude, est montré dans la figure **(II-6)**.

Diagramme en flèches de l'algorithme d'Euclide:

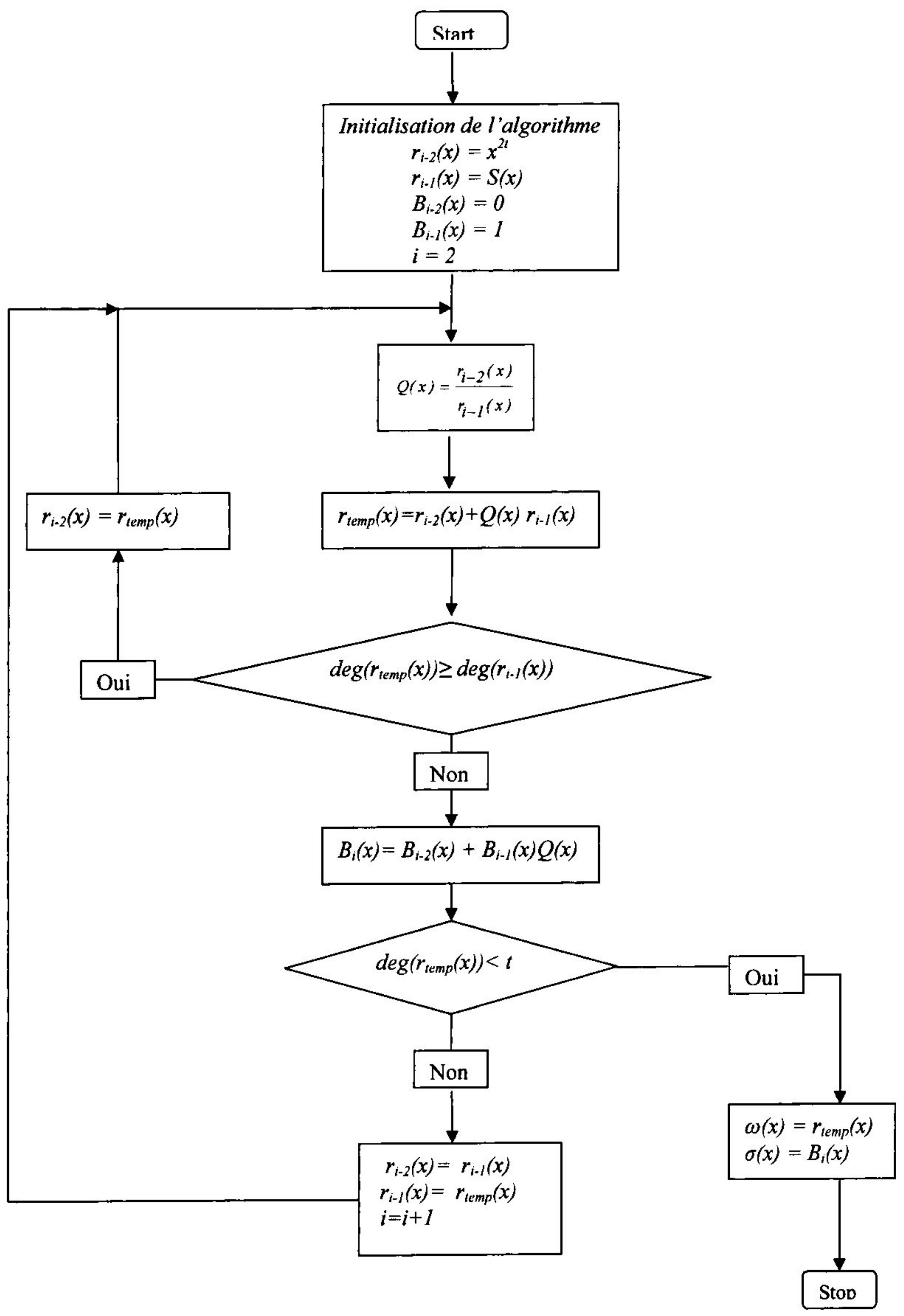


Figure II-6: *algorithme d'Euclide pour le calculant du polynôme de localisation et pour le polynôme d'amplitude*

Le dernier reste de la division nous donnera le polynôme d'amplitude. Le polynôme de localisation des erreurs est donné selon la relation :

$$\sigma_i (x) = \sigma_{i-2} (x)+ \sigma_{i-1}(x)Q_i(x) \qquad \textbf{(II-10)}$$

Avec :

$$\sigma_i (x) = B_i(x)$$

La théorie montre que l'on est obligé d'avoir deux blocs dans l'implémentation hardware.

Un bloc qui effectue la division et qui donnera le polynôme d'amplitude des erreurs, et un bloc de multiplication qui donnera le polynôme de localisation des erreurs.

II-4-4/ Chien search

Une fois le polynôme de localisation des erreurs calculé, on doit évaluer ses racines et sa dérivée. La dérivée formelle d'un polynôme dans $GF(2^m)$ est définie au chapitre **(I-7-6)**

L'évaluation des racines est effectuée avec l'algorithme appelé « Chien Search » qui est du type « brute force », c'est-à-dire, qu'il évalue toutes les possibilités. Par exemple pour un RS (7,5), on évalue le polynôme de localisation des erreurs et sa dérivée pour tous les éléments du « champ de Galois » $GF(2^3)$, sauf pour l'élément nul.

A la sortie de ce bloc, on obtiendra une séquence de symboles. Lorsque les symboles sont nuls, ceux-ci nous indiqueront qu'une racine a été détectée.

II-4-4-1/ Schéma du chien de search

Ce schéma permet de calculer les racines pour un polynôme de localisation des erreurs et pour sa dérivée (dérivation dans $GF(2^m)$ voir **(I-7-6)**). Le schéma de la figure **(II-7)** calcule les racines d'un code de RS(7,5), donc en $GF(2^3)$.

Pendant n coups d'horloge, on évaluera le polynôme de localisation des erreurs et son polynôme dérivé. Chaque coup d'horloge indiquera une valeur différente de α^i, où i indique le numéro du coup d'horloge.

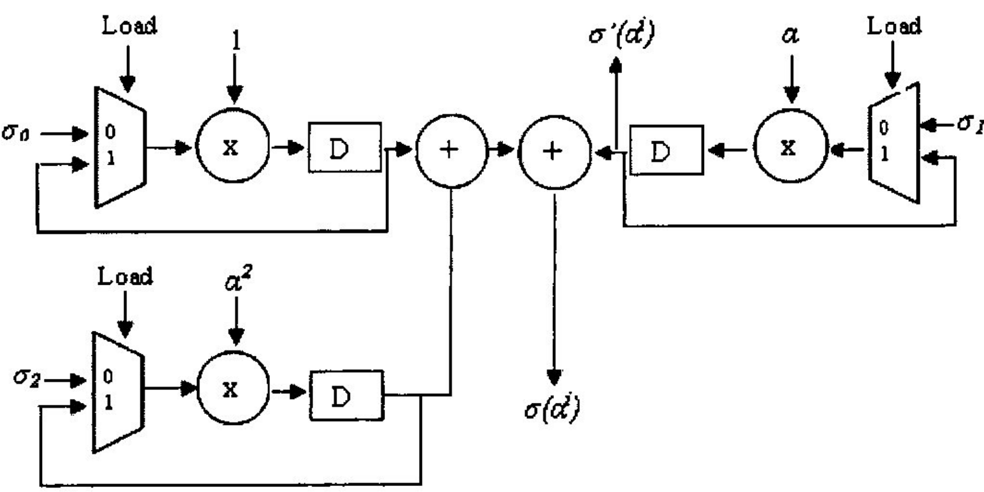


Figure II-7: schéma du bloc Chien Search

II-4-5/ Algorithme de Forney

Cet algorithme permet de construire le polynôme d'erreurs $e(x)$ à additionner avec le polynôme reçu $r(x)$ pour reconstituer le polynôme $c(x)$. Pour le calcul du polynôme $e(x)$, les polynômes $\sigma(\alpha^i)$, $\sigma'(\alpha^i)$, $\omega(\alpha^i)$ sont nécessaires. Le polynôme de localisation des erreurs et sa dérivée sont déjà évaluées pour les différentes valeurs de α, il nous reste à évaluer $\omega(\alpha^i)$.

Une fois les différentes valeurs de $\omega(\alpha^i)$ calculées, on applique l'algorithme de Forney. Cet algorithme est défini comme :

$$e_i = \frac{\omega(\alpha^i)}{\sigma'(\alpha^i)}$$

Avec :

$\omega(\alpha^i)$: polynôme d'amplitude évalué pour les valeurs de $GF(2^3)$

$\sigma'(\alpha^i)$: dérivée du polynôme de localisation des erreurs pour les valeurs de $GF(2^3)$

II-4-5-1/ Schéma de l'algorithme de Forney et de corrections d'erreurs

Le schéma pour le calcul de l'algorithme de Forney et pour la correction des erreurs est :

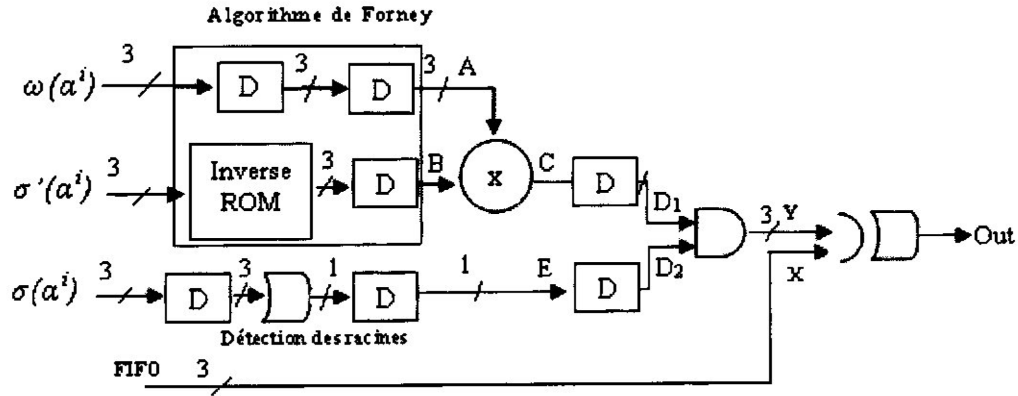


Figure II-8: schéma de l'algorithme de Forney **[11]**

II-4-5-1-1/ Evaluation du polynôme d'amplitude

L'évaluation du polynôme d'amplitude $\omega(\alpha^i)$ peut être effectuée avec un schéma semblable à celui du « Chien Search ». Le schéma de la figure **(II-9)** évalue le polynôme d'amplitude pour un code RS (7,5).

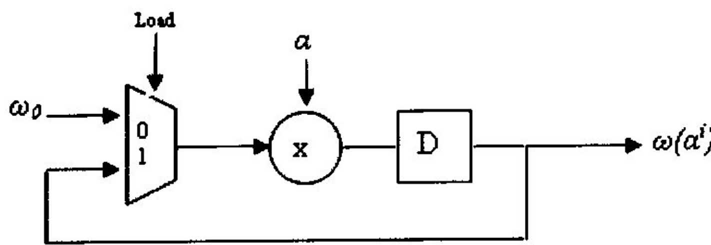


***Figure II-9**: schéma pour le calcul des racines du polynôme d'amplitude*

II-4-5-1-2/ Calcul des inverses

Le calcul de l'algorithme de Forney requiert une division entre deux valeurs,

$$e_i = \frac{\omega(\alpha^i)}{\sigma'(\alpha^i)}$$

Une division peut être calculée en faisant une multiplication par l'élément inverse du dénominateur. On crée une mémoire avec tous les éléments inverses de $GF(2^m)$ de manière à pouvoir effectuer la division avec une multiplication par un symbole inverse.

Le calcul de l'inverse dans un « champ de Galois » est défini comme suit :

$$\alpha^{-i} = \alpha^{element_max-i} \qquad \textbf{(II-11)}$$

II-4-5-1-3/ Multiplication

La multiplication entre la valeur inverse de $\sigma'(\alpha^i)$ et $\omega(\alpha^i)$ nous donne la valeur de e_i

$$e_i = \frac{\omega(\alpha^i)}{\sigma'(\alpha^i)} = \omega(\alpha^i)\sigma'(\alpha^{-i}) \cdot \qquad \textbf{(II-12)}$$

Effectuer une multiplication au lieu d'une division sert à économiser des coups d'horloge et à simplifier les circuits. La division faite à l'aide d'une multiplication inversée requiert un coup d'horloge, pour la lecture dans la mémoire, puis la multiplication est effectuée avec de la logique combinatoire, comme vue dans le paragraphe **(II-3-2-1-2)**.

II-4-5-1-4/Détection du zéro

La détection du zéro est effectuée avec une porte logique « NOR ». Lorsque l'on aura un élément nul à l'entrée, donc une racine, on aura un « 1 » à la sortie. Cette détection sert uniquement à sélectionner les éléments pour la correction des erreurs.

II-4-5-1-5/La porte logique « AND »

La porte logique « AND » sert à sélectionner seulement les symboles qui devraient être corrigés. De cette façon on éliminera les erreurs du mot-code reçu.

II-4-5-1-6/La porte logique « XOR »

L'addition modulo 2 donnera toujours le symbole du polynôme reçu lorsque la valeur de $e_i = 0$. Quand la valeur de $e_i \neq 0$, on additionnera le symbole de e_i et de r_i pour éliminer l'erreur.

II-5/ Conclusion

Les codes de Reed – Solomon sont des codes correcteurs d'erreurs utilisés dans tous les domaines requérant des données fiables. Typiquement, dans les communications aéronautique, télévision numérique et stockage de données.

Les codes de Reed – Solomon permettent de corriger des erreurs grâce à des symboles de contrôle ou de redondances ajoutés après l'information. Dans ce chapitre on a essayé de traiter l'aspect mathématique du codage et du décodage de ces codes.

CHAPITRE III

Dans ce chapitre :

- Introduction
- Les circuits logiques programmables
- Les circuits FPGAs
- Le langage VHDL
- L'outil de conception ISE
- Conclusion

III-1/ Introduction

L'électronique moderne se tourne de plus en plus vers le numérique qui présente de nombreux avantages sur l'analogique : grande insensibilité aux parasites et aux dérives diverses, modularité et reconfigurabilité, facilité de stockage de l'information etc. Cependant, les circuits numériques nécessitent une architecture plus lourde, et leur mode de traitement de l'information mis en œuvre nécessite plus de fonctions élémentaires que l'analogique, ce qui implique des temps de traitement plus long.

Aussi les fabricants de circuits intégrés numériques s'attachent à fournir des circuits présentant des densités d'intégration toujours plus élevée, pour des vitesses de fonctionnement de plus en plus grandes. D'abord réalisées avec des circuits SSI (Small Scale Integration), les fonctions logiques intégrées se sont développées avec la mise au point du transistor MOS, dont la facilité d'intégration a permis la réalisation de circuits MSI (**M**edium Scale **I**ntegration), LSI (Large Scale **I**ntegration) et VLSI (Very Large Scale Integration)**[12]**. Ces dernières générations ont vu l'avènement des microprocesseurs et microcontrôleurs.

Au début des années 70 les premiers composants (en technologie bipolaire) entièrement configurable par programmation sont apparus. La nouveauté résidait dans le fait qu'il était possible d'implanter physiquement par simple programmation, au sein du circuit, n'importe quelle fonction logique, et non plus de se contenter de faire réaliser une opération logique par un microprocesseur dont l'architecture est figée.

Dans un premier temps, ces circuits étaient dédiés à des fonctions simples en logique combinatoire, ils offrent aujourd'hui aux concepteurs la possibilité d'implanter des composants divers tel qu'un inverseur et un microprocesseur au sein d'un même boîtier; le circuit n'est plus limité à un mode de traitement séquentiel de l'information comme avec les microprocesseurs **[12]**.L'intégration des principales fonctions numériques d'une carte au sein d'un même boîtier permet de répondre à la fois aux critères de densité et de rapidité (les capacités parasites étant plus faibles, la vitesse de fonctionnement peut augmenter). La plupart de ces circuits sont maintenant programmés à partir d'un simple ordinateur type PC directement sur la carte où ils vont être utilisés. En cas d'erreur, ils sont reprogrammables électriquement sans avoir à

extraire le composant de son environnement. De nombreuses familles de circuits sont apparues depuis les années 70, avec des noms très divers suivant les constructeurs : des circuits très voisins pouvaient être appelés différemment par deux constructeurs concurrents. De même une certaine inertie dans l'évolution du vocabulaire a fait que certains circuits technologiquement différents ont le même nom **[13]**.

III-2/ Les circuits logiques programmables

Un circuit logique programmable PLD (**P**rogrammable **L**ogic **D**evice) est un assemblage d'opérateurs logiques combinatoires et de bascules dans lequel la fonction réalisée n'est pas fixée lors de la fabrication. Il contient potentiellement la possibilité de réaliser toute une classe de fonctions, plus ou moins large suivant son architecture.

Ces circuits programmables sont des circuits rapides, flexibles et à très haute densité d'intégration. Ils peuvent être aussi de technologie PROM et EPROM, implémentant n'importe quelle table de vérité d'une fonction logique. Ces composants ne nécessitant aucune étape technologique supplémentaire pour être personnalisé. Nous y trouvons les circuits logiques programmables incluant un grand nombre de solutions, toutes basées sur des variantes de l'architecture des portes "ET" et "OU". Le nombre d'entrée, de sortie, de connexions programmables et le niveau d'intégration conduisent à la classification suivante des PLD **[14]**:

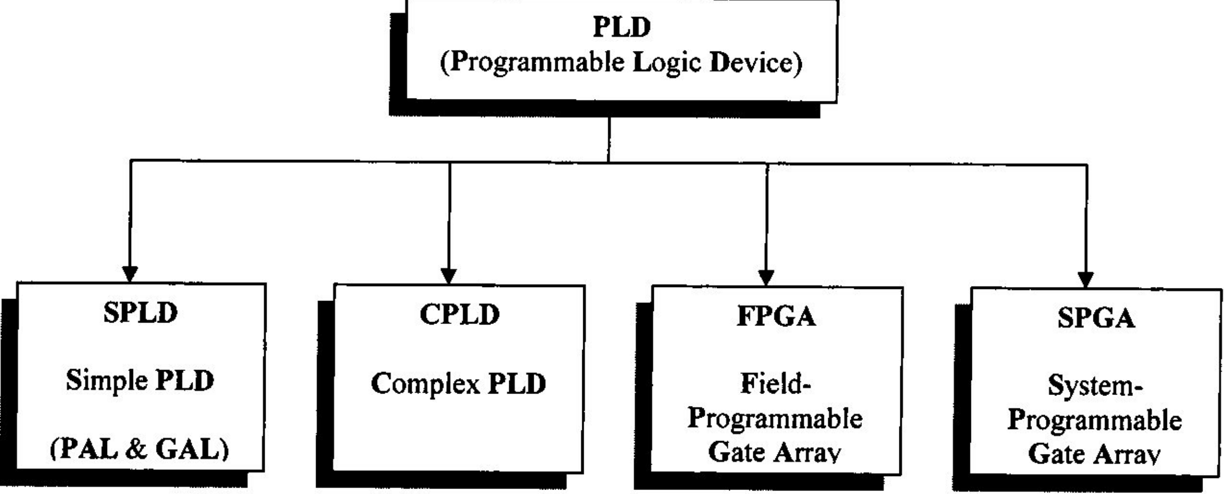


***Figure III.1** : Les différentes classes des PLD*

- SPLD (Simple Programmable Logic Device), ces circuits sont basés sur un réseau logique programmable et un module de sortie d'utilisation souple .Ils groupent les PALs et les GALs.

- Les PALs (Programmable Array Logic), c'est des circuits qui utilisent la technologie des fusibles pour pouvoir réaliser des circuits logiques dont les interconnexions internes seraient ainsi rendues programmable. Ils sont constitués d'un plan de portes AND programmables suivi d'un plan de portes "OR" figé.

- Les GALs (Genniric Array Logic), ces circuits ne sont rien d'autres que des PALs "CMOS", c'est à dire des PALs programmables mais surtout effaçables électriquement.

- Les CPLDs (Complex Programmable Logic Device), ces circuits son formés par un assemblage de blocs équivalents à une GAL qui sont reliés par un réseau d'interconnexion internes. Ils utilisent les technologies EPROM (ultra violet) et EEPROM. Ils sont relativement complexes (jusqu'à une ou deux dizaines de milliers de portes).

- Les FPGAs (Field Programmable Gate Array) marquent un saut dans l'architecture et la technologie, ils désignent des circuits qui peuvent être très complexes (jusqu'à des millions de portes logiques).

- SPGA (System Programmable Gate Array), cette quatrième classe est en cours d'apparition.

III-3/ Les circuits FPGAs

III-3-1/ Définition

En 1985, la société Xilinx introduisait un nouveau type de circuit électronique appelé FPGA (Field-Programmable Gate Array) ou "réseaux logiques programmables" destiné à remplacer de petites quantités de logique discrète **[15]**. Le principe de ces circuits, dérivés des PAL, est de fournir une certaine quantité de logique, entièrement programmable : après une phase de configuration, le FPGA se comporte comme le circuit décrit par sa programmation. L'originalité des FPGAs Xilinx est de stocker la

configuration dans une mémoire vive statique (SRAM), permettant ainsi de reprogrammer ces circuits à volonté, y compris dans un système en train de fonctionner. Cette reprogrammabilité ne faisait pas partie des objectifs de Xilinx, mais elle a permis d'ouvrir les FPGAs à un nouveau champ d'applications.

Grâce aux évolutions de la technologie microélectronique, les FPGAs deviennent de plus en plus performants avec des capacités qui augmentent sans cesse. Longtemps, réalisés autour de blocs de logique configurable à base de LUT (Look Up Table), les FPGAs peuvent aujourd'hui comporter de larges mémoires RAM configurables des opérateurs arithmétiques complexes et des cœurs de microprocesseurs.

III-3-2/ Architecture

Il y a quatre principales catégories d'architecture des circuits FPGAs disponibles commercialement :

- Tableau symétrique (Symmetrical Array), utilisée par Xilinx.
- Mers de portes (Sea Of Gates).
- En colonne (Row Based), utilisée par Actel.
- Les PLD hiérarchiques (Hierachical PLD), utilisée par Altera.

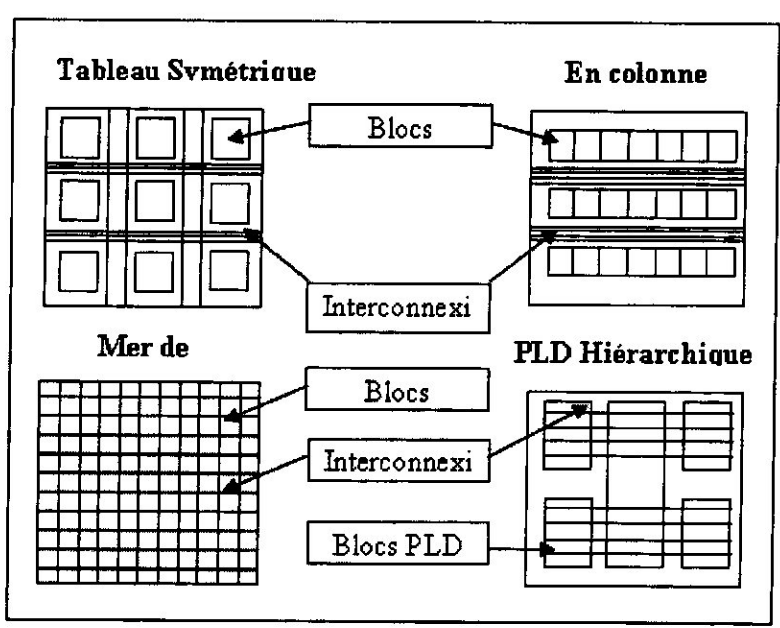


Figure III.2 *: Les différentes architectures d'un circuit FPGA*

Ces appellations sont dues à la manière dont sont distribués les CLB (Configurable Logic Bloc) sur la couche configurable du circuit. Tous les circuits FPGAs se présentent sous forme de deux couches figure **(III.3)** :

- Une couche de circuits configurables constituée d'une matrice de blocs logiques programmables (CLB) et entourée de blocs d'entrée-sortie programmables (IOB). L'ensemble est relié par un réseau d'interconnexions programmable (Route Matrix).

- Une couche de réseau mémoire (SRAM), dans laquelle est chargée la configuration de connexion à établir.

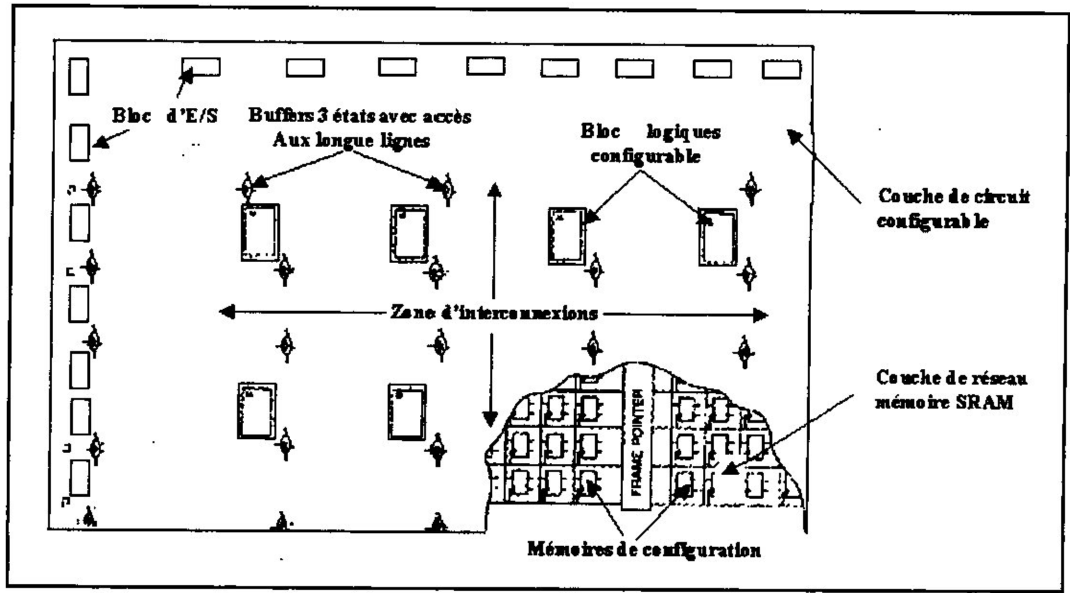


Figure III.3 *: Structure interne d'une architecture*

III-3-3/ Les éléments structurels

Les éléments constitutifs d'un FPGA sont toujours à peu près les mêmes pour les différentes architectures.

III-3-3-1/ Les éléments logiques

Ce sont les briques de bases de tout FPGA. Grâce à leur configuration on peut réaliser dans ces blocs toutes les opérations de la logique combinatoire (dans la limite d'un nombre d'entrées). Ces blocs ont souvent la même constitution malgré la

différence de fabricants et d'architectures. Ils sont généralement constitués d'une ou plusieurs LUTs (Look Up Table) qui contiennent, après la configuration, la table de vérité de la fonction logique qu'elles doivent réaliser ou alors un ensemble de valeurs qui sont mémorisées comme dans une ROM. La taille des LUTs est généralement de 4 entrées. Les LUTs sont généralement suivis d'un registre de sortie, ce qui permet de synchroniser, si nécessaire, la sortie sur une horloge.

III-3-3-2/ Les éléments de mémorisation

Pour des applications plus importantes, les FPGAs demandent souvent des capacités de stockage (citons en exemple les applications du traitement d'images). La nécessité d'intégrer des blocs de mémoires directement dans l'architecture des FPGAs est vite devenue cruciale. De cette façon les temps d'accès à la mémoire sont diminués puisqu'il n'est plus nécessaire de communiquer avec des éléments extérieurs au circuit.

III-3-3-3/ Les éléments de routages

Les ressources de routages représentent la plus grosse partie de silicium consommée sur la puce réalisant le circuit. Ces ressources sont composées de segments (de longueurs différentes) qui permettent de relier entre autres éléments eux les via des matrices de connexions. Si ces éléments de routages sont importants c'est parce qu'ils vont déterminer la vitesse et la densité logique du système.

III-3-3-4/ Les éléments d'entrées/sorties

Sur une carte, le circuit FPGA appartient à un système d'ensemble pouvant contenir des parties micro programmées. Le circuit doit donc avoir un lien avec son environnement, c'est le but des éléments d'entrées/sorties. Ceux-ci peuvent être protégés par des buffers ou par d'autres éléments permettant la gestion des entrées et des sorties.

III-3-3-5/ Les éléments de contrôle et d'acheminement des horloges

Les FPGAs sont prévus pour recevoir une ou plusieurs horloges. Des entrées peuvent être spécialement réservées à ce type de signaux, ainsi que des ressources de routages spécialement adaptées au transport d'horloges sur de longues distances.

III-3-4/ Configuration des circuits FPGAs

La configuration est le processus de chargement des données spécifiques à un design dans un ou plusieurs FPGAs pour définir l'opération fonctionnelle des blocs internes ainsi que leurs interconnexions. Le temps de chargement des données dépend du mode de configuration sélectionné. Selon le mode de configuration, on distingue quatre classes de circuits FPGAs **[16]**.

III-3-4-1/ Circuit configurable

Un circuit FPGA configurable est un circuit programmé et chargé par différentes données, où les interconnections de l'FPGA sont programmé d'une façon à donner au circuit un fonctionnement spécifique.

III-3-4-2/ Circuit reconfigurable

C'est le même principe de configuration, sauf que cette fois en reconfigurant le circuit FPGA une deuxième fois pour l'utiliser par une autre application, comme on peut garder la dernière configuration du circuit, ou l'effacer et l'en configurer une nouvelle fois.

III-3-4-3/ Circuit partiellement reconfigurable

Un circuit FPGA est dit partiellement reconfigurable s'il est possible de le reconfigurer sélectivement, tel que la partie non sélectionnée du circuit est inactive, mais elle conserve son information configurée.

III-3-4-4/ Circuit dynamiquement reconfigurable

Un circuit FPGA est dynamiquement reconfigurable s'il peut être partiellement reconfiguré durant son fonctionnement, c'est-à-dire qu'une partie du circuit correspond à un certain fonctionnement logique, tandis que les interconnections peuvent être changées sans affecter le fonctionnement de la logique restante.

On peut parler aussi de reconfiguration dynamique dans le cas ou plusieurs circuits FPGAs sont connectés entre eux, tel qu'il s'agit de reconfigurer un seul circuit FPGA tout en gardant les autres circuits en fonctionnement.

III-3-5/ Architecture de la série XC4000E de Xilinx

Dans cette section, nous nous intéressons qu'à l'architecture de la série XC4000E de Xilinx.

III-3-5-1/ Les éléments de base

L'architecture interne est composée d'une matrice de blocs logique (CLB) programmables, de blocs logique d'entrée/sortie (IOB) programmables disposés sur la périphérique, de buffers à trois états (deux par CLB) et de ressources d'interconnexion entre ces éléments.

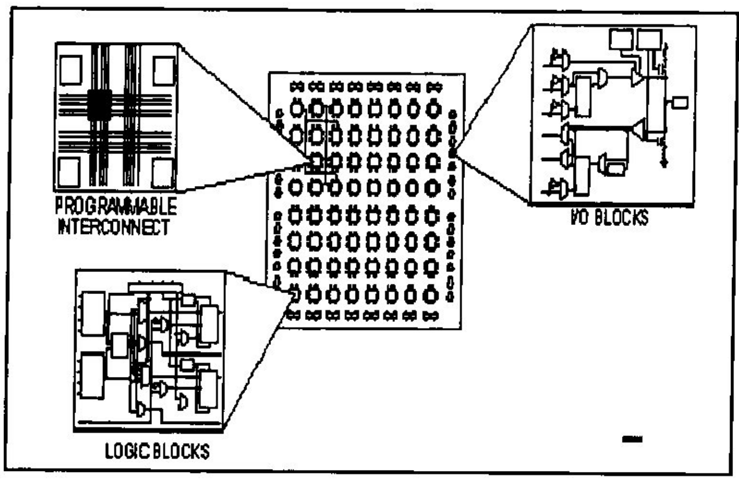


Figure III.4 *: Architecture interne de la série XC4000E*

III-3-5-2/ Les CLB (configurable logic bloc)

Ces blocs permettent de réaliser des fonctions combinatoires et des fonctions séquentielles. Chaque CLB est composé d'un bloc de logique combinatoire composé de deux générateurs de fonctions à quatre entrées et d'un bloc de mémorisation composé de deux bascules D. Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du CLB. La figure **(III.5)**, nous montre le schéma d'un CLB.

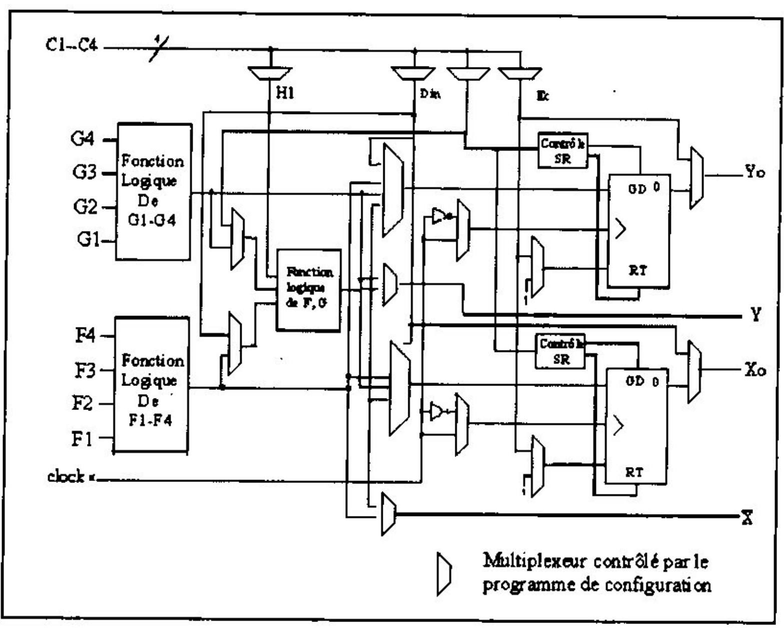


***Figure III.5** : Cellule logique (CLB).*

Les générateurs de fonctions F et G à quatre entrées indépendantes (F1...F4, G1...G4) sont générées à partir d'une table de vérité câblée inscrite dans une zone mémoire, alors il sont mis en application en tant que table; Look-Up Table (LUT) de mémoire. Une troisième fonction H est réalisée à partir des sorties F et G et d'une troisième variable d'entrée H1 sortant d'un bloc composé de quatre signaux de contrôle H1, Din, S/R, Ec. Les signaux des générateurs de fonction peuvent sortir du CLB, soit par la sortie X, pour les fonctions F et G, soit Y pour les fonctions G et H.

III-3-5-3/ Les IOBs (input output bloc)

Ces blocs permettent l'interfaçage entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisée (haute impédance).

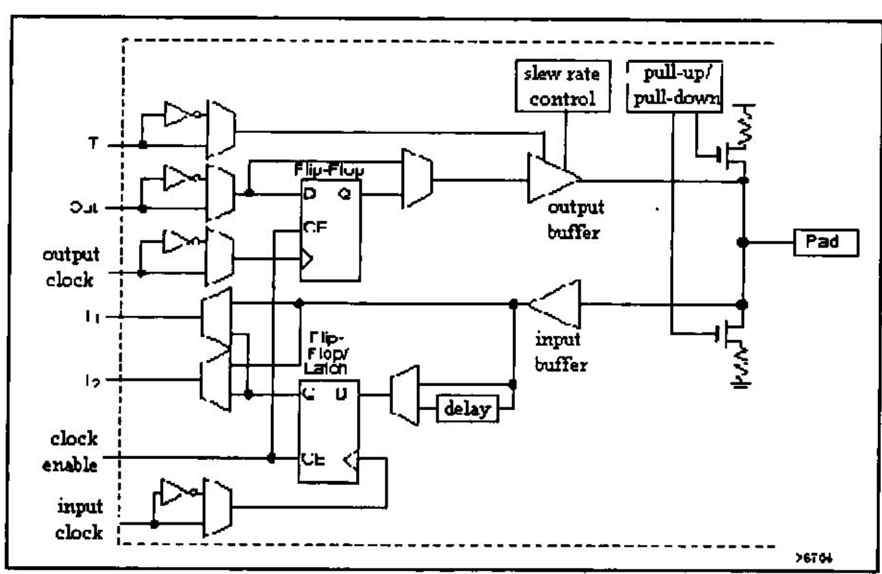


Figure III. 6: Input Output Block (IOB)

III-3-5-4/ Le système d'interconnexion

Les connexions internes dans les circuits FPGAs sont composées de segments métallisés. Parallèlement à ces lignes, nous trouvons des matrices programmables (PSM, Programmable Switch Matrix) réparties sur la totalité du circuit, horizontalement et verticalement entre les divers CLB. Elles permettent les connexions entre les diverses lignes, celles-ci sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM. Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé possible.

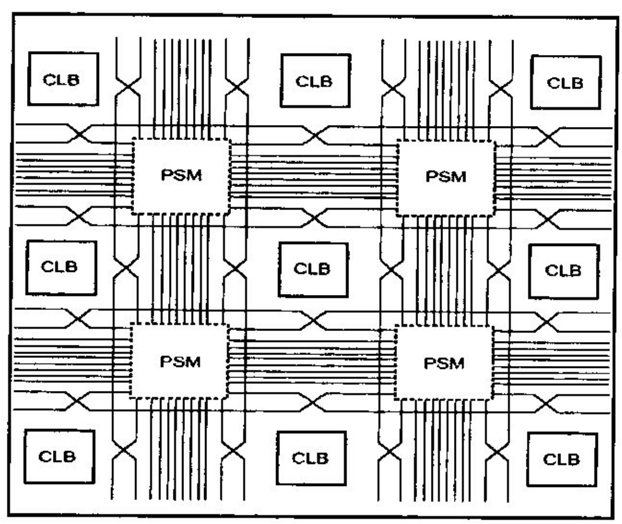
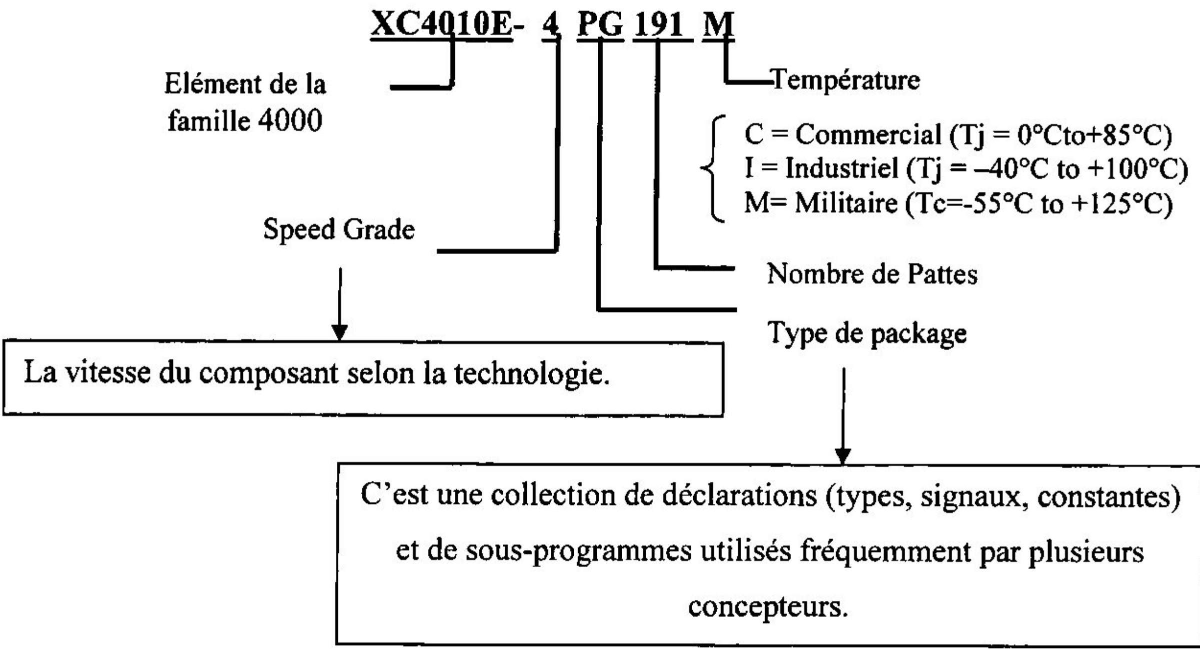


Figure III. 7 : Système d'interconnexion

III-3-6/ Nomenclature des circuits FPGAs

Les circuits FPGAs suivent la nomenclature suivante **[12]** :



III-4/ Le langage VHDL

L'évolution des circuits logiques programmables permet de répondre à un besoin de forte intégration. Des méthodes performantes sont nécessaires pour la conception de système numérique complexe afin d'utiliser au mieux la capacité des

circuits actuels tout en maîtrisant la durée du développement. La description par schémas n'est plus suffisante; il est indispensable d'utiliser des méthodes de conception moderne associées avec un langage de haut niveau. Dans cette section, nous présentons le langage de description VHDL.

III-4-1/ Présentation

Le langage **VHDL** (VHSIC (Very-High-Speed-Integrated-Circuit) **H**ardware **D**escription **L**anguage), créé à la demande du Département de la Défense aux USA (DOD), a fait l'objet d'une norme internationale en 1986 et il est en passe de s'imposer partout **[17]**. Il est proposé par la grande majorité des sociétés vendant des logiciels de développement d'ASICs (Application Specific Integrated Circuit), de FPGAs ou même de PLDs.

En 1987, une première version du langage VHDL est standardisée par l'institut IEEE (Institute of Electrical and Electronics Enginneers). Il est à noter que la norme ne définit aucune méthodologie de travail, elle ne fait que fournir un outil de description. Au début cette norme ne reconnaît que le type "bit", qui ne comprend que les états "0" et "1", ce qui pose un problème lors de la simulation. En fait, le problème est comment modéliser un bus haute impédance, ou comment réagir lorsqu'une incohérence du code provoque un court-circuit. Pour résoudre ce problème, le type "Std_Logic" est créé et normalisé par la même norme en 1993. Parallèlement, une évolution du VHDL est normalisée en 1993. Cette nouvelle version supprime quelques ambiguïtés de la version de 1987, et surtout qu'elle met à disposition de nouvelles commandes.

III-4-2/ Objectifs du langage VHDL

On peut donner une idée de la place occupée par le VHDL, en formulant une analogie entres les outils de développement pour les structures logique câblée / logique programmée

Logique câblée	Logique programmée
Schéma structurel	Langage machine
Description par équation logique	Langage assembleur
VHDL	Langage structuré

***Tableau III.1** : Analogie entre la logique câblée et la logique programmable*

Le VHDL répond à deux objectifs, dont le premier est l'obtention d'un modèle de simulation permettant de valider une solution avant de réaliser le composant, le deuxième objectif est la synthèse, en particulier pour la mise en œuvre de circuits programmables de type FPGA ou CPLD.

III-4-3/ Développement d'un projet en VHDL

La figure **(III.8)** montre le déroulement des différentes étapes de développement d'un projet sur un circuit FPGA et les différents niveaux d'utilisation d'un code VHDL **[18]**.

- Le code source VHDL peut faire appel à des composants issus d'une bibliothèque fournie par un fabricant de circuit. Si la description de ces composants est entièrement comportementale, le code obtenu est portable et peut être synthétisé par n'importe quel compilateur.

- La simulation fonctionnelle permet de tester la validité de la conception. Aucune vérification temporelle n'est ici effectuée. Elle permet de valider fonctionnellement la description (entrée syntaxique ou schématique) en s'affranchissant des étapes parfois longues de placement et routage.

- Le synthèse consiste à mettre à plat toute la schématique, à synthétiser les descriptions syntaxiques comportementales, à réduire les équations logiques et à les optimiser en fonction du composant FPGA ciblé. A ce stade de la conception, la

manière dont les ressources logiques du composant vont être utilisées est définie. La liste des interconnexions (Netlist) va décrire, en VHDL, les interconnexions entre les éléments logiques du composant. La simulation peut être partiellement temporelle mais les délais dus au routage (qui dépend lui même du placement) ne sont pas encore connus.

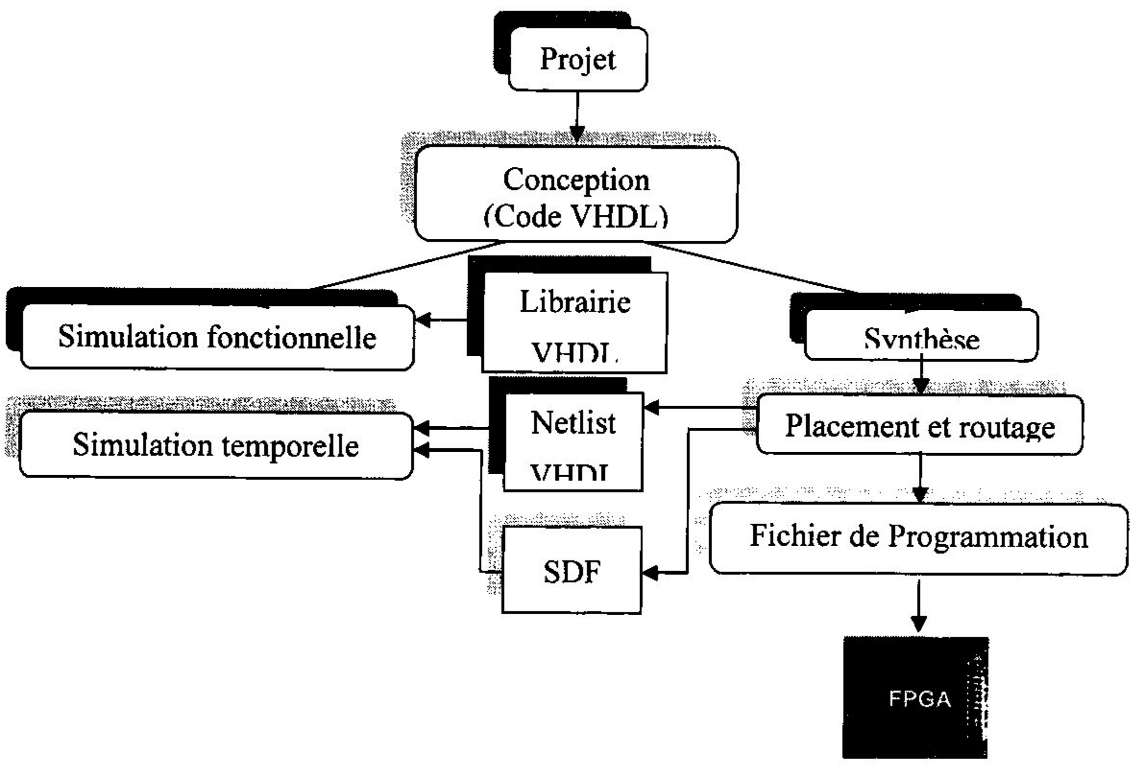


Figure III.8 *: Etapes de développement d'un projet en VHDL*

- Après placement et routage, le procédé de la vérification temporelle est envisagé. Certaines contraintes spécifiées dans un fichier permettront de vérifier la vitesse du fonctionnement globale. Les délais et les contraintes sont définis comme des paramètres et les valeurs de ces paramètres sont passées à l'outil de simulation temporelle par l'intermédiaire du fichier SDF (Standard Delay File) **[17]**. Si la simulation temporelle met en évidence un disfonctionnement dû à des délais trop grands ou si les contraintes spécifiées ne sont pas respectées, il est alors nécessaire d'optimiser soit la conception soit la phase de placement et de routage.

- La phase du fichier programmable ou appelé le "Bit File" est la dernière opération de développement du projet pour qu'il soit implanté sur le composant ciblé. C'est le fichier binaire qui peut être téléchargé dans un dispositif FPGA. Ce fichier est souvent inclus avec la phase de placement et routage.

III-5/ L'outil de conception ISE

Xilinx a mis en place des logiciels de développement des circuits FPGAs performants. Les séries de logiciels de Xilinx ont été conçues pour contribuer à l'étude des projets sur FPGAs. Le résultat final de tels projets est un fichier binaire qui peut être téléchargé dans un circuit FPGA. Le logiciel de développement de projets sur les circuits FPGAs mis sur le marché est l'ISE (Integrated Software Environement), dont nous utiliserons la série "Foundation ISE 7.1i".

III-5-1/ Les environnements de développement

L'ISE contrôle tous les aspects du déroulement de la conception. A l'aide de l'interface de gestion de projets, on peut accéder aux divers outils d'exécution de la conception. On peut également accéder aux dossiers et aux documents liés au développement du projet. L'outil ISE contient quatre environnements de développement d'un projet comme la montre la figure **(III.9)**.

III-5-1-1/ Environnement de conception

Cet environnement permet la description d'une architecture en langage VHDL, VERILOG ou ABEL, soit en utilisant l'éditeur de schémas ou bien en utilisant les diagrammes d'états.

III-5-1-2/ Environnement de simulation et de vérification

Le VHDL était en premier lieu utilisé pour décrire le comportement du système qui devra être conçu. Il y a un nouveau transfert de temps de développement; avant d'écrire et simuler le code final d'un système, on vérifie, par simulation, si le

comportement du circuit est celui souhaité. La simulation permet de vérifier si le design est opérationnel par :

- Une simulation fonctionnelle après la description en code VHDL.
- Une simulation temporelle après l'implantation de l'algorithme sur le circuit désiré.

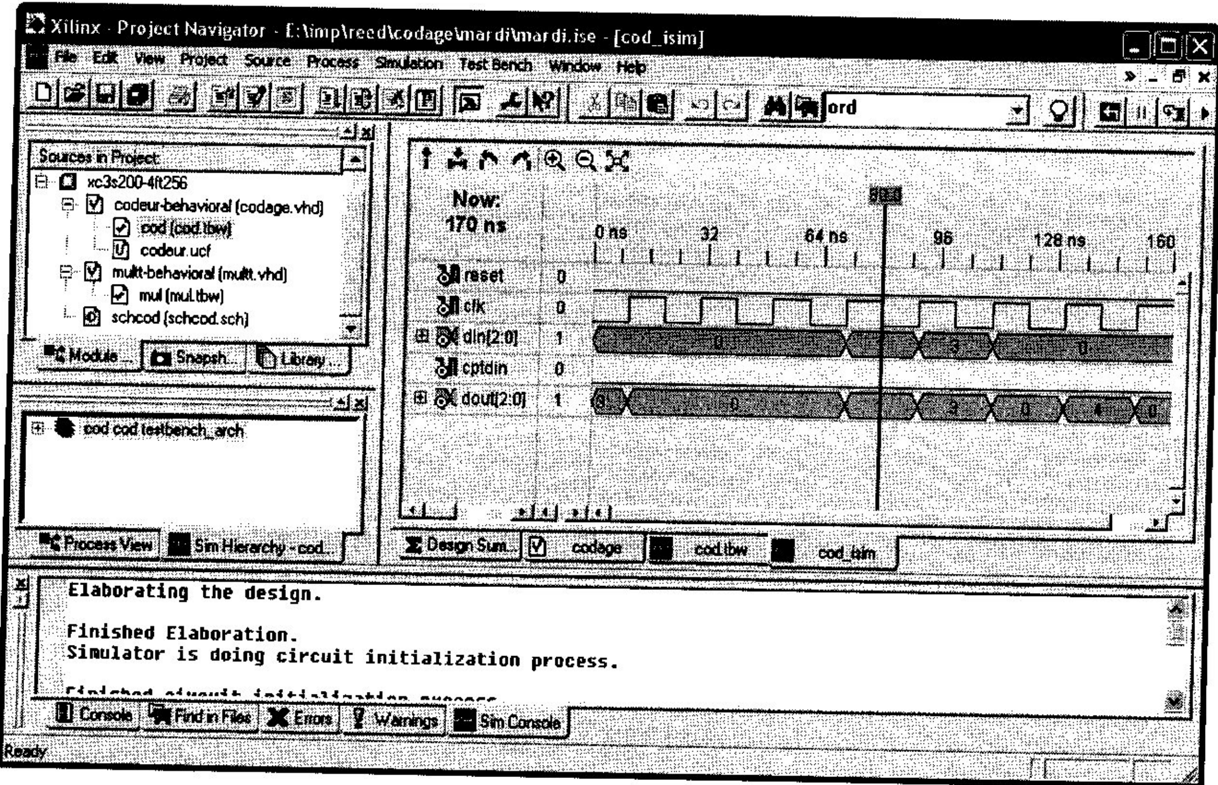


Figure III.9 : *Les environnements de l'outil de conception ISE 7.1*

III-5-1-3/ Environnement de synthèse

Cet environnement transforme la description VHDL en portes logiques et optimise l'architecture ciblée en surface et en temps.

III-5-1-4/ Environnement de l'implantation physique

La spécification VHDL est directement émulée sur un support matériel (circuit FPGA) en précisant la famille utilisée pour une implantation physique du circuit. La compilation du code VHDL en fichier binaire permet de générer le schéma

correspondant et une netlist constituée d'une liste d'équations booléennes et d'informations portant sur les entrées/sorties du circuit.

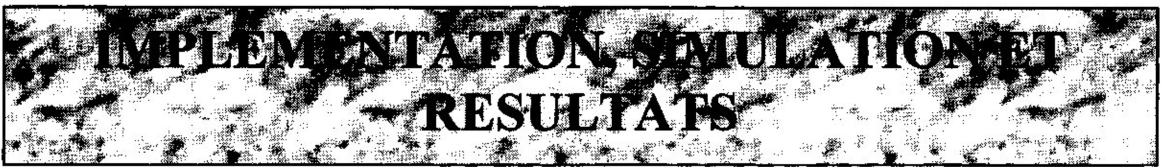
III-6/ Conclusion

Dans ce chapitre nous avons donné un aperçu sur l'évolution des circuits programmables et nous avons fait une description particulière des circuits FPGAs. Ces derniers permettent, aujourd'hui, un degré d'intégration très élevé (plus d'un million de portes). Ils amènent toute la souplesse de la logique programmable pour l'utilisateur.

Par la suite, nous avons donné une brève présentation du langage de conception VHDL et cité les étapes de développement d'un projet en VHDL. Si ce langage supporte un certain niveau d'abstraction, le résultat final après compilation consiste en une réalisation physique exploitant plus ou moins bien l'architecture interne d'un circuit logique intégrable dans un FPGA.

Nous avons terminé ce chapitre par une présentation de l'outil "ISE Foudation 7.1i" de Xilinx, utilisé pour développement d'une implantation sur FPGA.

CHAPITRE IV



Dans ce chapitre :

- Introduction
- Construction d'un GF(2^3)
- Construction d'un champ de GF(2^3) sous Matlab
- Exemple du nombre de symboles corrigibles
- Codage
- Décodage
- Implémentation Hardware

IV -1/ Introduction

Dans le chapitre précédent nous avons donnée une description générale sur les circuits logiques programmables, en particulier les FPGAs. Ces derniers peuvent être adaptés aux systèmes à concevoir, avec un degré d'intégration très élevé, tout en permettant de réaliser des descriptions de haut niveau ou encore des cœurs logiques (Logic core)

Dans ce chapitre nous allons décrire notre travail qui consiste à implémenter un code correcteur d'erreurs de type Reed-Solomon sur FPGA.Dans un premier temps, nous allons donner les étapes du codage et du décodage des informations avec les codes détecteurs /correcteurs de Reed-Solomon, avec un exemple pratique RS(7,5) utilisant les champs de Galois dans la construction du code.

IV -2/ Construction d'un $GF(2^3)$

On a un code de Reed-Solomon RS (7,5) **(II-2)** avec 5 symboles d'informations et 2 symboles de contrôles (7-5=2). Donc chaque symbole est représenté sur 3 bits.

On va construire tous les éléments du $GF(2^3)$ à partir du polynôme primitif : $p(x) = x^3 + x+1$ Avec $m=3$ (nombre de bits).

Ce polynôme permet de construire le « champ de Galois » souhaité. Tous les éléments non nuls du champ peuvent être construits en utilisant l'élément α comme racine du polynôme primitif.

On peut en déduire:

$$p(x) = x^3 + x+1$$
$$p(\alpha) = \alpha^3 + \alpha +1$$
$$0 = \alpha^3 + \alpha +1$$
$$\alpha^3 = \alpha +1$$

Maintenant,il suffit de multiplier l'élément α^3 par α à chaque étape et réduire par rapport à $\alpha^3 = \alpha + 1$ pour obtenir le champ complet. On aura besoin de 5 multiplications pour compléter le champ.

Les éléments d'un « champ de Galois » de $GF(2^3)$ sont :

Elément	Forme polynomia	Forme binaire	Forme décimales
0	0	000	0
1	1	001	1
α	α	010	2
α^2	α^2	100	4
α^3	$\alpha+1$	011	3
α^4	$\alpha^2+\alpha$	110	6
α^5	$\alpha^2+\alpha+1$	111	7
α^6	α^2+1	101	5

***Tableau IV.1**: éléments de GF (2^3)*

IV -3/ Construction d'un champ de $GF(2^3)$ sous Matlab

Les éléments d'un « champ de Galois » peuvent être aussi calculés avec Matlab selon les instructions suivantes:

```
p=2;                                       % Nombre base du champ
m=3;                                       % Eléments
Champ = gftuple ([-1:p^m-2]', m, p);       %Calcul du champ en binaire
```

***Code IV.1** : éléments dans GF (2^3) sous Matlab*

IV -4/ Exemple du nombre de symboles corrigibles

On va maintenant s'introduire dans le principe pratique de Reed-Solomon avec un petit aperçu sur le nombre de symboles corrigibles dans l'exemple du code de Reed-Solomon RS(7,5) ; que l'on utilisera par la suite pour tous les autres exemples. L'objectif est de découvrir combien de bits sont utilisés pour chaque symbole et combien d'erreurs peut-on corriger.

RS(n,k) = RS(7,5) sachant que n indiquant la longueur totale d'un bloc de Reed – Solomon ; 7 symboles dans ce cas et k indique la longueur du bloc d'information, 5 symboles dans cet exemple.

La capacité de correction des erreurs du système est:

$$2t = n - k = 7 - 5 = 2$$

Donc :

$$t = \frac{n-k}{2} = 1$$

Ce code permettra de corriger 1 symbole. Le nombre de bits m par symbole est :

$$n=2^m-1$$

$$m = \frac{\ln(n+1)}{\ln(2)} = \frac{\ln(8)}{\ln(2)} = 3$$

Le nombre de bits utilisés pour coder les symboles est donc de 3. Ce qui nous amène à Utiliser un « Champ de Galois » de $GF(2^3)$

IV -5/Codage

On procède par plusieurs étapes pour coder l'information à transmettre :

IV -5-1/ Calcul des coefficients du polynôme générateur pour RS(7,5)

On veut calculer les coefficients du polynôme générateur pour le calcul des symboles de contrôle d'un code de RS (7,5) qui puisse corriger 1 erreur, comme vu dans l'exemple précédent.

La forme générale du polynôme générateur est :

$$g(x)=(x-\alpha^1)(x-\alpha^2)\ldots\ldots(x-\alpha^{2t})$$

En développant cette équation, on trouve :

$$g(x) = (x - \alpha)(x - \alpha^2) \quad \text{avec } 2t = 2$$
$$= x^2 - \alpha^2 x - \alpha x + \alpha^3$$

$$= x^2 - (\alpha^2 + \alpha)x + \alpha^3$$
$$= x^2 + (\alpha^2 + \alpha)\, x + \alpha^3$$

En prenant l'équivalence en décimal, on obtient :

$$g(x) = x^2 + 6x + 3x$$

IV -5-2/ Implémentation Hardware du codeur

Le codage est systématique, on doit effectuer une opération de décalage pour placer les informations dans le degré élevé du mot-code de sortie.

L'implémentation du codeur demandera deux opérations : un décalage et une division. Ces deux opérations peuvent être effectuées grâce à des registres à décalage et à des multiplexeurs. Pour *GF(7,5)* ;on aura le schéma suivant :

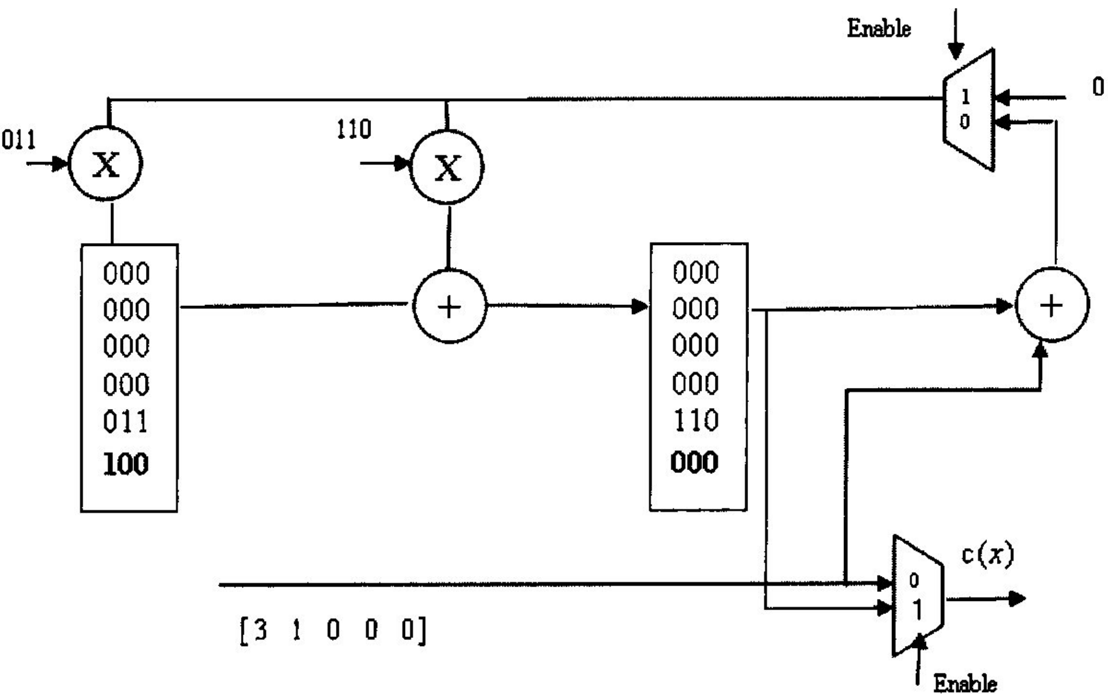


***Figure IV.1** : Schéma du codage du RS (7,5)*

Dans la figure **(IV.1)** du codage, quand les symboles à coder arrivent le premier signal Enable de *c(x)* sera activé et reste actif jusqu'à ce que le dernier symbole d'information soit sorti du codeur. En parallèle le deuxième Enable du polynome générateur est désactivé en même temps. Après cela le signal Enable de c(x) sera désactivé pour recevoir les symboles à codés ;en parallèle l'autre signal Enable du

polynome générateur sera activer pour permettre le calcul des symboles de contrôles . Dans notre travail on a introduit une information *i(x)* = [0 0 0 1 3] dans le codeur ;la même information sortira dans *c(x)* pendant les 5 premier coups d'horloge ou rien ne ce passe en parallèle dans le polynôme générateur .Les symboles de contrôles serons calculer pendant les deux autres coups d'horloges.

Après les 7 coups d'horloges ; on aura en sortie le mot-code reconstitué composé des symboles de redondances en plus des symboles d'informations envoyer.

Donc *c(x)*= [0 0 0 1 3 / 0 4]

IV -5-2-1/Addition

L'addition des deux symboles définis dans le « champ de Galois » de $GF(2^3)$.

$$A(x) = a_2x^2 + a_1 x + a_0$$

$$B(x) = b_2 x^2 + b_1 x + b_0$$

Sachant que : $a_2 ; a_1 ; a_0 ; b_2 ; b_1 ; b_0$: appartiennent à $GF(2)$.

L'addition dans un « champ de Galois » de $GF(2^3)$ se calcule ainsi:

$$Out = A(x) + B(x) = x^2(a_2 + b_2) + x(a_1 + b_1) + a_0 + b_0$$

$$Out(2) = (a_2 + b_2)$$

$$Out(1) = (a_1 + b_1)$$

$$Out(0) = a_0 + b_0$$

La figure **(IV .2)** montre le schéma de l'addition

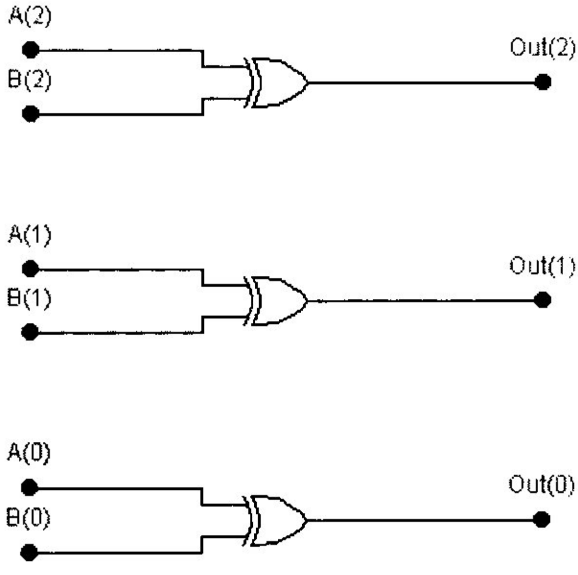


***Figure IV.2**: schéma de l'addition en GF (2^3).*

IV -5-2-2/Multiplication

Les multiplications utilisées dans les codes de Reed – Solomon sont des multiplications dans le « champ de Galois » $GF(2^m)$

La multiplication de deux éléments dans le « champ de Galois » de $GF(2^3)$.

$$A(x) = a_2 x^2 + a_1 x + a_0$$
$$B(x) = b_2 x^2 + b_1 x + b_0$$

Sachant que : $a_2 ; a_1 ; a_0 ; b_2 ; b_1 ; b_0$ appartiennent à $GF(2)$.

La multiplication dans ce cas doit être réduite selon le polynôme primitif :

$$P(x) = x^3 + x + 1$$

Donc la réduction est en :

$$x^3 = x + 1$$

La multiplication donne :

$$A(x)B(x) = (a_2 x^2 + a_1 x + a_0)(b_2 x^2 + b_1 x + b_0)$$
$$= (a_2 b_2 x^4 + (a_2 b_1 + a_1 b_2)x^3 + (a_2 b_0 + a_0 b_2 + a_1 b_1)x^2 + (a_1 b_0 + a_0 b_1)x + a_0 b_0)$$

Réduction en x^3:

$$x^3 = x+1$$
$$x^4 = x^3 x = x(x+1)$$

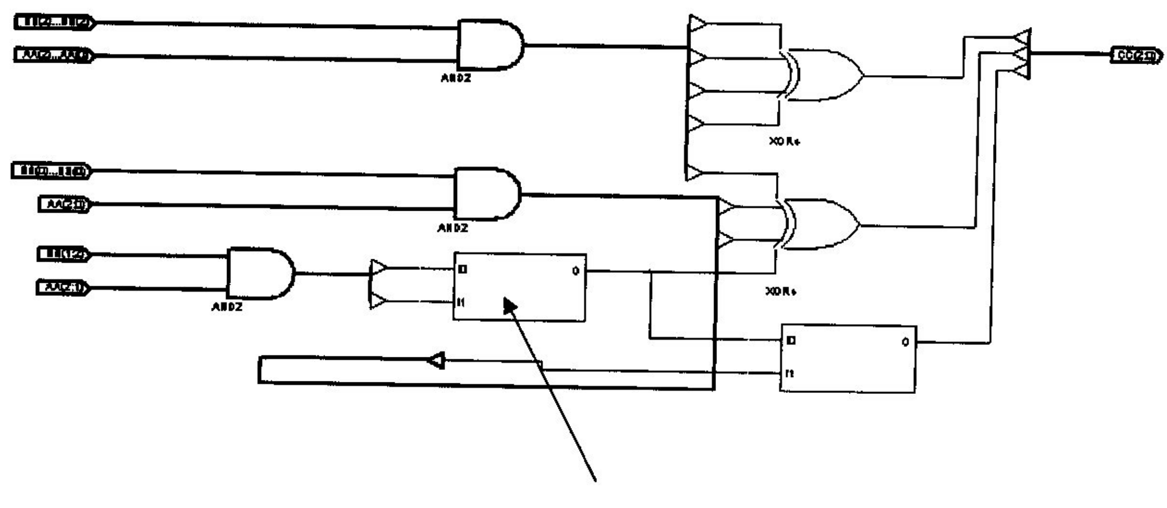
En réduisant l'expression ci-dessus, on obtient :

$$A(x)B(x) = (a_2 x^2 + a_1 x + a_0)(b_2 x^2 + b_1 x + b_0)$$
$$= (a_2 b_2 x^4 + (a_2 b_1 + a_1 b_2)x^3 + +(a_2 b_0 + a_0 b_2 + a_1 b_1)x^2 + (a_1 b_0 + a_0 b_1)x + a_0 b_0)$$
$$= (a_2 b_2 x(x+1) + (a_2 b_1 + a_1 b_2)(x+1) + +(a_2 b_0 + a_0 b_2 + a_1 b_1)x^2 + (a_1 b_0 + a_0 b_1)x + a_0 b_0)$$
$$= (a_2 b_0 + a_0 b_2 + a_1 b_1 + a_2 b_2)x^2 + (a_1 b_0 + a_0 b_1 + a_2 b_1 + a_1 b_2 + a_2 b_2)x + a_0 b_0 + a_2 b_1 + a_1 b_2)$$

La multiplication entre chaque terme est une opération logique « AND » et la somme est une addition modulo 2, donc une opération logique « XOR ».

La figure **(IV.3)** montre la multiplication en $GF(2^3)$:

Si on double clic sur l'emplacement du curseur (flèche) sur le schéma suivant reçu à partir du schématic du logiciel Xilinx 7,1 i on aura le schéma qui le suit ;de la même façon on aura le troisième schéma le plus siumpifier (schéma avec composants de base)



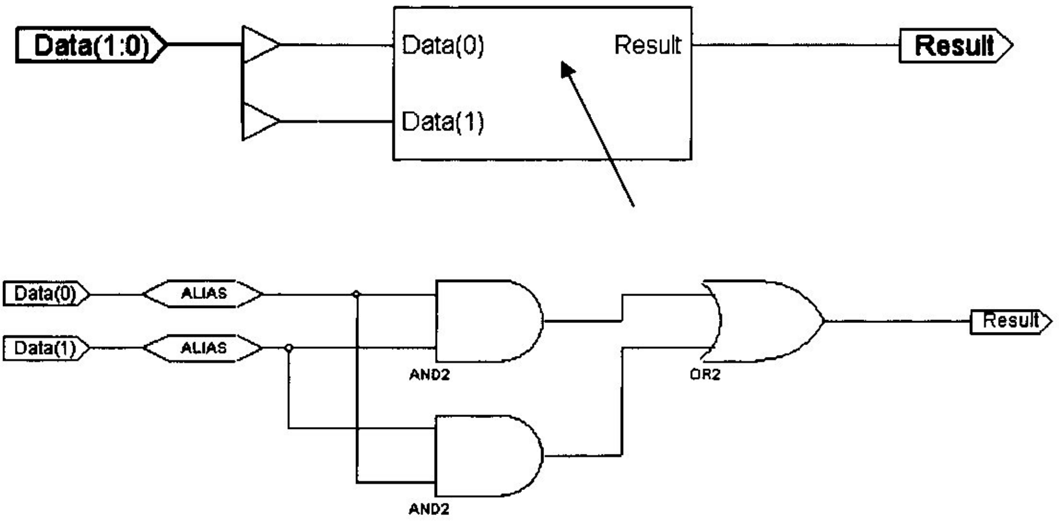


***Figure IV.3** : schéma de multiplication en GF (2^3).*

En ce qui concerne la programmation des codes Reed-Solomon en utilisera le langage VHDL, qui permet la description des aspects les plus importants d'un système matériel (*hardware system*), à savoir son comportement, sa structure et ses caractéristiques temporelles. Par système matériel, on entend un système électronique arbitrairement complexe réalisé sous la forme d'un circuit intégré ou d'un ensemble de cartes.c'est un langage de description de haut niveau.

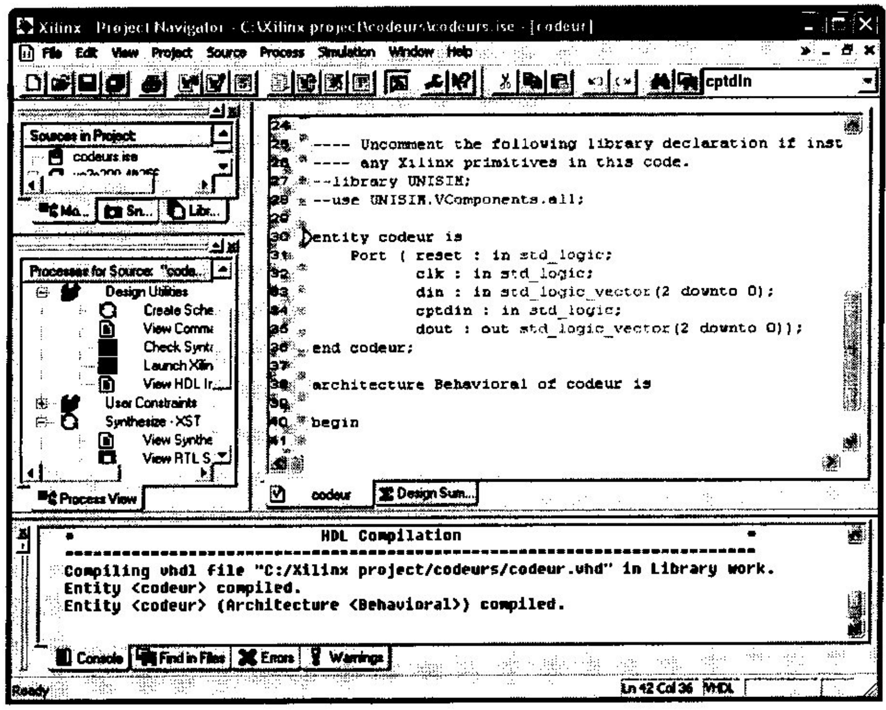


***Figure IV.4** : L'outil de programmation VHDL*

Pour la simulation du programme du codage ; on utilise "ISE Simulator "

***Figure IV.5** : L'outil de simulation* "ISE Simulator "

IV -6/Décodage

L'idée de base du décodeur de Reed – Solomon est de détecter une séquence erronée avec peu de termes, qui sommée aux données reçues, donne lieu à un mot-code valable.

Plusieurs étapes sont nécessaires dans notre travail pour le décodage de ces codes :

- Calcul du syndrome
- Calcul des polynômes de localisation des erreurs et de d'amplitude
- Calcul des racines et évaluation des deux polynômes
- Sommation du polynôme constitué et du polynôme reçu pour reconstituer l'information de départ sans erreur.

IV-6-1/Calcul du syndrome

Dans notre travail ; le calcul du syndrome a été effectué par un processus itératif. Avant de pouvoir calculer le polynôme du syndrome, on doit attendre que l'on ait reçu tous les éléments du polynôme $r(x)$.

On a un code de RS(7,5) **(II-2)**; donc 7− 5 = 2 symboles de contrôle ($2t = 2$).

On envoie le message suivant :

$$c(x)=x^3+\alpha^3x^2+\alpha^2$$

On prend deux cas ; le premier sans erreurs et dans le deuxième on introduit une seule erreur

IV -6-1-1/ 1er cas

On reçoit le polynôme suivant $r(x)$ et on veut calculer le syndrome S

$$r(x)=x^3+\alpha^3x^2+\alpha^2$$

$$\begin{aligned} S_1 = r(x=\alpha) &= (\alpha^3) + \alpha^3(\alpha^2) + \alpha^2 \\ &= \alpha^3 + \alpha^5 + \alpha^2 \\ &= \alpha + 1 + \alpha^2 + \alpha + 1 + \alpha^2 \\ &= 0 \end{aligned}$$

$$\begin{aligned} S_2 &= r(x = \alpha^2) \\ &= ((\alpha^2)^3) + \alpha^3((\alpha^2)^2) + \alpha^2 \\ &= \alpha^6 + \alpha^7 + \alpha^2 \\ &= \alpha^2 + 1 + 1 + \alpha^2 \\ &= 0 \end{aligned}$$

Donc le syndrome est:

$$S(x) = 0.$$

Puisque le syndrome est nul ; alors on n'aura pas besoin de poursuivre les autres étapes de correction des erreurs (Le message reçu est celui qu'on l'a envoyé).

IV -6-1-2/ 2er cas

On reçoit le polynôme suivant $r(x)$ et on veut calculer le syndrome S

$$r(x)=\alpha x^3+\alpha^3x^2+\alpha^2$$

$S_1 = r(x=\alpha)$

$= \alpha\,(\alpha^3) + \alpha^3(\alpha^2) + \alpha^2$

$= \alpha^4 + \alpha^5 + \alpha^2$

$= \alpha^2 + \alpha + \alpha^2 + \alpha + 1 + \alpha^2$

$= 1 + \alpha^2$

$= \alpha^6$

$S_2 = r(x = \alpha^2)$

$= \alpha\,((\alpha^2)^3) + \alpha^3((\alpha^2)^2) + \alpha^2$

$= \alpha^2$

Donc le syndrome est :

$$S(x) = \alpha^2 x + \alpha^6$$

Alors on poursuivra toutes les étapes de corrections des erreurs après le calcul du syndrome comme suit :

On aura besoins de *2t* schémas comme celui de la figure **(IV.6)**, pour avoir le syndrome complet.

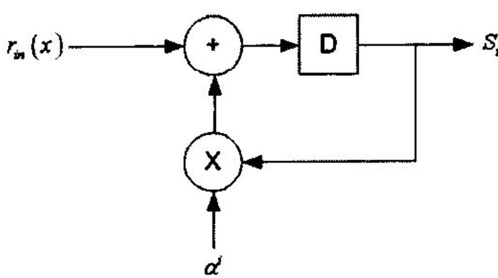


***Figure* IV.6**: *schéma pour le calcul du syndrome*

Depuis la figure **(IV.7)**, on peut calculer la valeur du premier coefficient du polynôme S_1 selon le polynôme reçu : $r(x)=\alpha x^3+\alpha^3x^2+\alpha^2$

I	r_{in}	D_1	$S_{1\,out}$	$Multi$
0	0	0	0	-
1	0	0	0	0
2	0	0	0	0
3	α	α	0	0
4	α^3	α^5	α	α^2
5	0	α^6	α^5	α^6
6	α^2	α^6	α^6	1
7	-	-	α^6	1

Tableau IV.2: *tableau du calcul du syndrome S_1*

Schéma

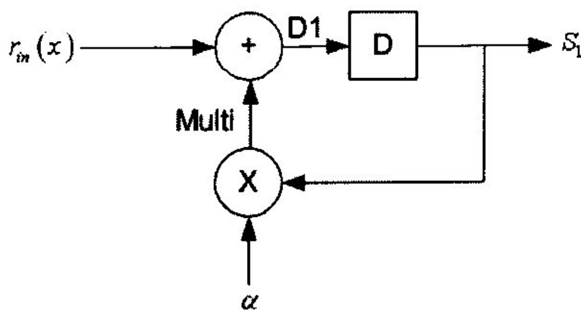


Figure IV.7 *: schéma avec signaux détaillés pour le calcul du syndrome S_1*

Et on peut calculer aussi le coefficient du polynôme S_2 depuis la figure ***(IV.8)***, selon le polynôme reçu $r(x)$; Avec :

$$r(x)=\alpha x^3+\alpha^3x^2+\alpha^2$$

I	r_{in}	D_1	$S_{1\,out}$	Multi
0	0	0	0	-
1	0	0	0	0
2	0	0	0	0
3	α	α	0	0
4	α^3	0	α	α^3
5	0	0	0	0
6	α^2	α^2	0	0
7	-	-	α^2	α^4

***Tableau VI.3**: tableau du calcul du syndrome S_2*

Schéma

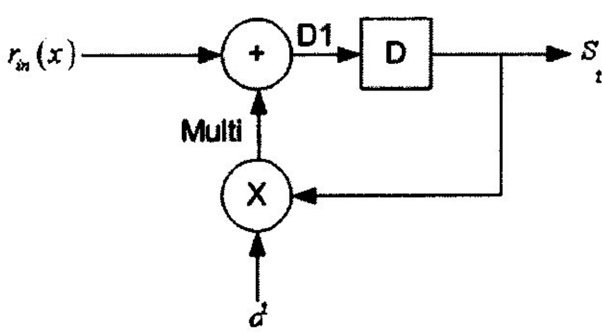


***Figure IV.8** : schéma avec signaux détaillés pour le calcul du syndrome S_2*

IV -6-2/Correction d'erreurs avec Euclide :

On cherche à trouver avec L'algorithme d'Euclide le plus grand diviseur commun de deux polynômes $r_0(x)$ et $r_1(x)$ dans le « champ de Galois » $GF(q)$.

Considérons le code RS(7,5), comme on l'a calculé au chapitre **(IV-4)**, avec un mot-code $c(x) = 0$, un mot-code reçu $r(x) = \alpha x^3 + \alpha^3 x^2 + \alpha^2$ et le syndrome calculé au chapitre **(IV-6-1-2)**. On cherche à calculer le polynôme de localisation des erreurs et le polynôme d'amplitude en utilisant l'algorithme d'Euclide.

On prend $r_0 = x^{2t} = x^2$ et $r_1(x) = \alpha^2 x + \alpha^6 = S(x)$

On commence à diviser $r_0(x)$ par $r_1(x)$:

$x^2 + 0x + 0$ $x^2 + \alpha^4 x$ ——————— $\alpha^4 x + 0$ $\alpha^4 x + \alpha$ ——————— α	$r_1(x) = \alpha^2 x + \alpha^6$ ——————— $\alpha^5 x + \alpha^2$

Comme $deg\ (r_2(x)) < t$, $deg\ (r_2(x)) < 1$ on arrête l'algorithme. Le dernier reste de la division est le polynôme d'amplitude cherché :

$$\omega\ (x) = \alpha$$

Le polynôme de localisation des erreurs cherché est :

$$\begin{aligned}\sigma_2 &= \sigma_0(x) + \sigma_1(x)\ Q_2(x)\\ &= 0 + 1(\alpha^5 x + \alpha^2)\\ &= \alpha^5 x + \alpha^2\end{aligned}$$

IV -6-3/ Chien search

L'évaluation des racines est effectuée avec l'algorithme appelé « Chien Search », qu'il évalue toutes les possibilités. Dans notre exemple RS(7,5), on évalue le polynôme de localisation des erreurs et sa dérivée pour tous les éléments du « champ de Galois » $GF(2^3)$, sauf pour l'élément nul.

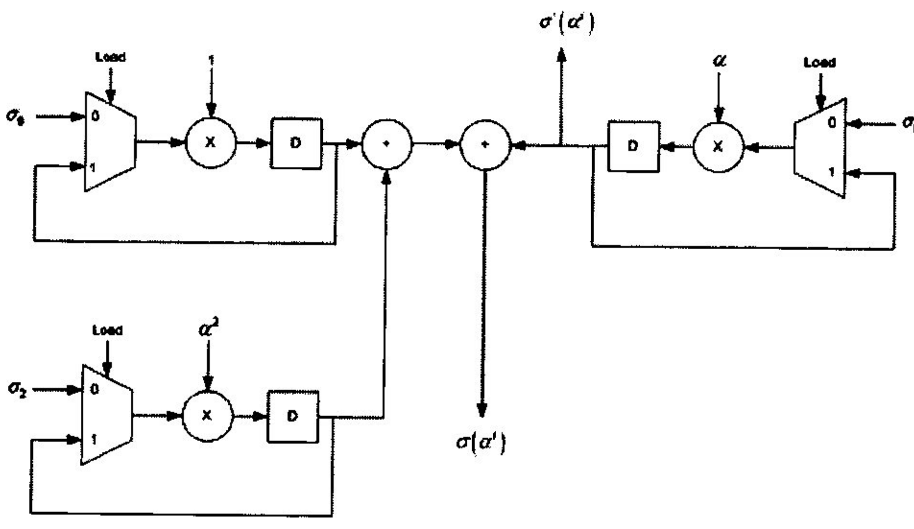


Figure IV.9 *: schéma du bloc Chien Search*

Le calcul des racines du polynôme de localisation des erreurs est établi au chapitre **(IV-6-2)** selon « Chien Search ». Premièrement, on calculera les racines en substituant x pour les différents éléments de $GF(2^3)$, car on utilise un code RS(7,5). Ensuite, on testera le schéma pour comprendre son fonctionnement.

Soit le polynôme de localisation des erreurs suivant :

$$\sigma(x) = \alpha^5 x + \alpha^2$$

On doit essayer tous les éléments possibles de $GF(2^3)$.

$$\sigma(\alpha^i) = \alpha^5(\alpha^i) + \alpha^2$$

$$\begin{aligned}\sigma(\alpha) &= \alpha^5(\alpha) + \alpha^2 \\ &= \alpha^6 + \alpha^2 \\ &= \alpha^2 + 1 + \alpha^2 \\ &= 1\end{aligned}$$

$$\begin{aligned}\sigma(\alpha^2) &= \alpha^5(\alpha^2) + \alpha^2 \\ &= \alpha^7 + \alpha^2 \\ &= 1 + \alpha^2 \\ &= \alpha^6\end{aligned}$$

$\sigma(\alpha^3) = \alpha^5(\alpha^3) + \alpha^2$

$= \alpha^8 + \alpha^2$

$= \alpha + \alpha^2$

$= \alpha^4$

$\sigma(\alpha^4) = \alpha^5(\alpha^4) + \alpha^2$

$= \alpha^9 + \alpha^2$

$= \alpha^2 + \alpha^2$

$= 0$

$\sigma(\alpha^5) = \alpha^5(\alpha^5) + \alpha^2$

$= \alpha^{10} + \alpha^2$

$= \alpha^3 + \alpha^2$

$= \alpha + 1 + \alpha^2$

$= \alpha^5$

$\sigma(\alpha^6) = \alpha^5(\alpha^6) + \alpha^2$

$= \alpha^{11} + \alpha^2$

$= \alpha^4 + \alpha^2$

$= \alpha^2 + \alpha + \alpha^2$

$= \alpha$

$\sigma(\alpha^7{=}1) = \alpha^5(1) + \alpha^2$

$= \alpha^5 + \alpha^2$

$= \alpha^2 + \alpha + 1 + \alpha^2$

$= \alpha + 1$

$= \alpha^3$

Résumé des calculs :

Elément	Résultat
α	*1*
α^2	α^6
α^3	α^4
α^4	*0*
α^5	α^5
α^6	α
1	α^3

***Tableau VI.4**: racine du polynôme de localisation des erreurs*

Les racines peuvent être calculées selon le schéma de la figure **(IV.10)**.

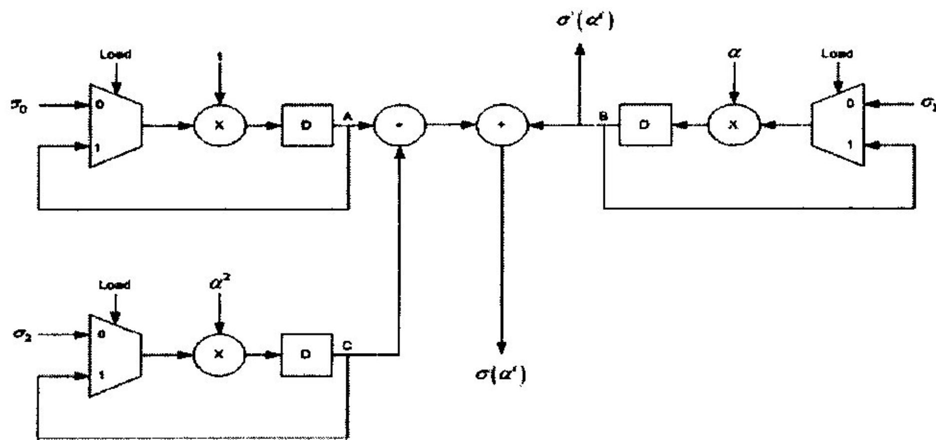


***Figure IV.10** : schéma avec signaux détaillés pour l'évaluation du polynôme de localisation et sa dérivée*

On résume les résultats dans le tableau ci-dessous ; selon le schéma précédent :

i	A	B	C	$A+C$	B	$\sigma'(a)$	$\sigma(\alpha^i)$
0	0	0	0	0	0	0	0
α	α^2	α^6	0	α^2	α^6	α^6	1
α^2	α^2	1	0	α^2	1	1	α^6
α^3	α^2	α	0	α^2	α	α	α^4
α^4	α^2	α^2	0	α^2	α^2	α^2	0
α^5	α^2	α^3	0	α^2	α^3	α^3	α^5
α^6	α^2	α^4	0	α^2	α^4	α^4	α
1	α^2	α^5	0	α^2	α^5	α^5	α^3

Tableau IV.5: *Calcul des racines du polynôme de localisation des erreurs*

IV -6-4/ Algorithme de Forney

IV -6-4-1/Schéma de l'algorithme de Forney et de la correction des erreurs :

IV -6-4-1-1/ Evaluation du polynôme d'amplitude

On prend comme exemple le schéma de la figure **(II-15)** pour un code RS(7,5). Le but étant d'évaluer le polynôme d'amplitude :

$$\omega(x) = \alpha$$

On évalue le polynôme d'amplitude selon le schéma de la figure **(IV-9)**.

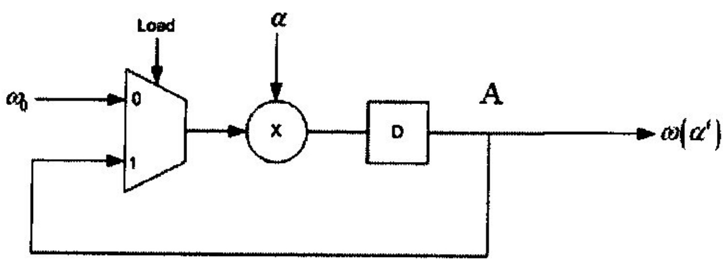


Figure IV.11: *schéma avec signaux détaillés pour l'évaluation du polynôme d'amplitude*

Résumé des calculs :

A	*ω (α^i)*
0	0
α^2	α^2
α^3	α^3
α^4	α^4
α^5	α^5
α^6	α^6
1	1
α	α

Tableau IV.6:*Evaluation du polynôme d'amplitude*

IV -6-4-1-2/ Inversion avec ROM

La division peut être calculée en faisant une multiplication par l'élément inverse du dénominateur. On crée un ROM avec tous les éléments inverses de $GF(2^3)$ de manière à pouvoir effectuer la division avec une multiplication par un symbole inverse.

Dans le cadre de notre travail ; on va calculer l'inverse pour $GF(2^3)$ .En sachant que les éléments non nuls dans $GF(2^3)$ sont au nombre de 7, on peut calculer tous les éléments inverses.

Le calcul des inverses est effectué selon l'équation **(II-11)**:

$\alpha^{-1}= \alpha^{7-1}=\alpha^6$

$\alpha^{-2}= \alpha^{7-2}=\alpha^5$

…

Résumé des calculs :

Elément	*Inverse*
0	0
1	1
α	α^6
α^2	α^5
α^3	α^4
α^4	α^3
α^5	α^2
α^6	α

***Tableau IV.7**: tableau d'inversion pour $GF(2^3)$*

IV -6-4-2/Calcul du polynôme de correction selon le schéma de Forney

On veut calculer le polynôme de correction selon le schéma de Forney pour le polynôme de localisation des erreurs et pour le polynôme d'amplitude des erreurs du chapitre **(IV-6-2)**. Pour ce faire, on a fixé des points sur le schéma, de manière à mieux repérer les signaux en question.

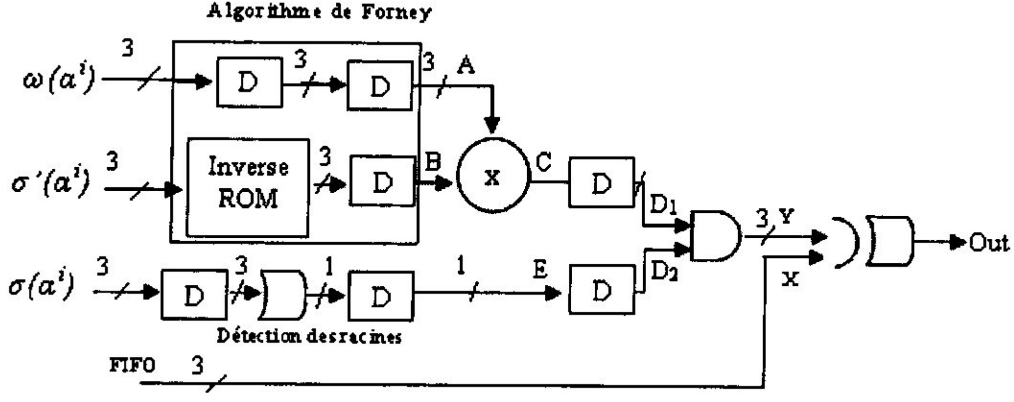


***Figure IV.12**: schéma de l'algorithme de Forney détaillé*

Les valeurs de $\omega(\alpha^i)$, $\sigma(\alpha^i)$, $\sigma'(\alpha^i)$ sont les valeurs calculées dans les exemples Précédents, paragraphes **(IV-6-4-1-1)**, **(IV-6-3)**.

Le tableau ci-dessous montre les différentes étapes :

i	*A*	*B*	*C*	*E*	*D1*	*D2*	*X*	*Y*	*Out*
0	α^2	α	α^3	0	0	0	-	0	-
1	α^3	1	α^3	0	α^3	0	0	0	0
2	α^4	α^6	α^3	0	α^3	0	0	0	0
3	α^5	α^5	α^3	1	α^3	0	0	0	0
4	α^6	α^4	α^3	0	α^3	1	α	α^3	
5	1	α^3	α^3	0	α^3	0	α^3	0	α^3
6	α	α^2	α^3	0	α^3	0	0	0	0
7	-	-	-	-	α^3	0	α^2	0	α^2

***Tableau IV.8**: calcul de l'algorithme de Forney selon schéma*

D'après le tableau ci-dessus et après tout les étapes de décodages ; on remarque que la correction des erreurs apparaissent dans la case finale (*Out*) .Dans notre exemple on a une seule erreur ; qu'on a corrigé par les multiples étapes ; qui se situe à la case de la position n°4 (x^3) .D'ou on aura à la fin notre message initiale $c(x)$.

IV -7/Implémentation Hardware

IV -7-1/ Introduction

Dans les sous-chapitres qui suivent, on discutera et expliquera l'implémentation hardware choisie. On implémentera un code RS(7,5). Ce code sera un simple correcteur d'erreurs,

IV -7-2/ Flux de conception hardware

Les principales étapes pour la conception hardware d'un circuit logique sur FPGA ou CPLD sont :

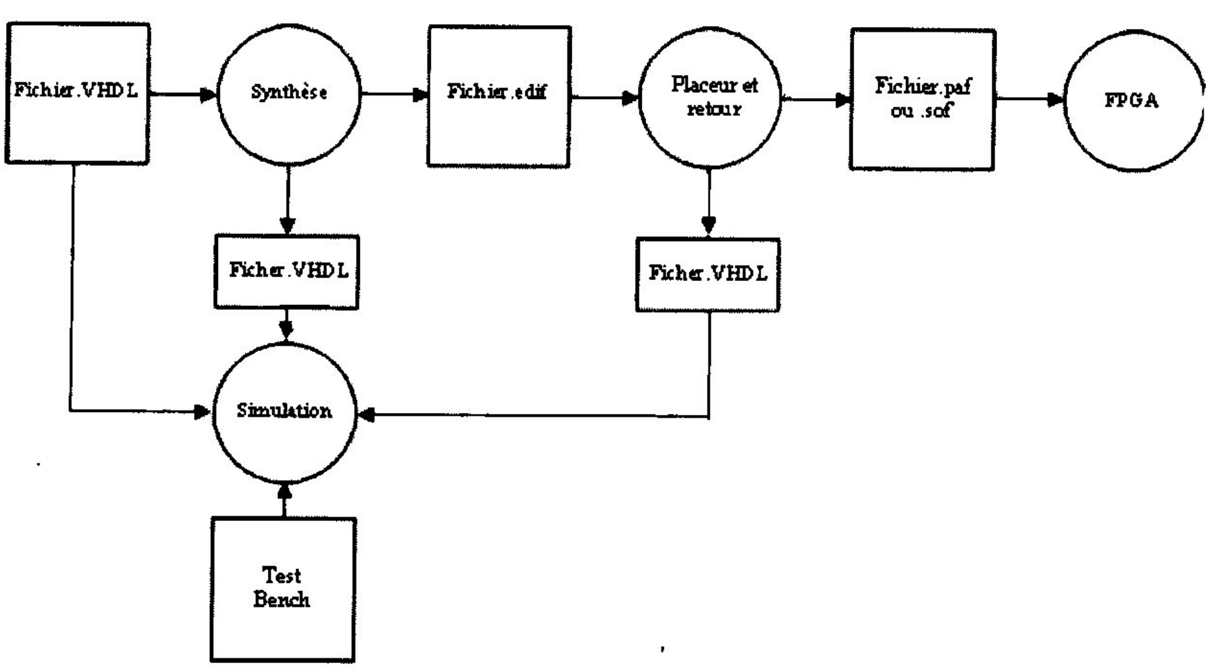


***Figure IV.13**: étape d'implémentation d'un circuit logique sur FPGA ou CPLD*

1. Le fichier .vhdl contient la description du circuit traité
2. La simulation permet d'essayer le fichier décrit en .vhdl, afin de contrôler le résultat temporellement
3. La synthèse permet de faire la traduction du fichier .vhdl en fichier logic. Le fichier crée sera un fichier .edif
4. Le placement/routage permet de placer les interconnexions créées avec le fichier edif de synthèse pour une technologie donnée. Chaque fabricant fournit son propre logiciel. Le placement et routage fournira selon la technologie choisie un fichier .paf ou sof
5. Le programmeur permettra de programmer un FPGA ou un CPLD avec le fichier .paf ou .sof

IV -7-3/ Codeur

IV -7-3-1/ Eléments du codeur

Le schéma du codeur est le suivant :

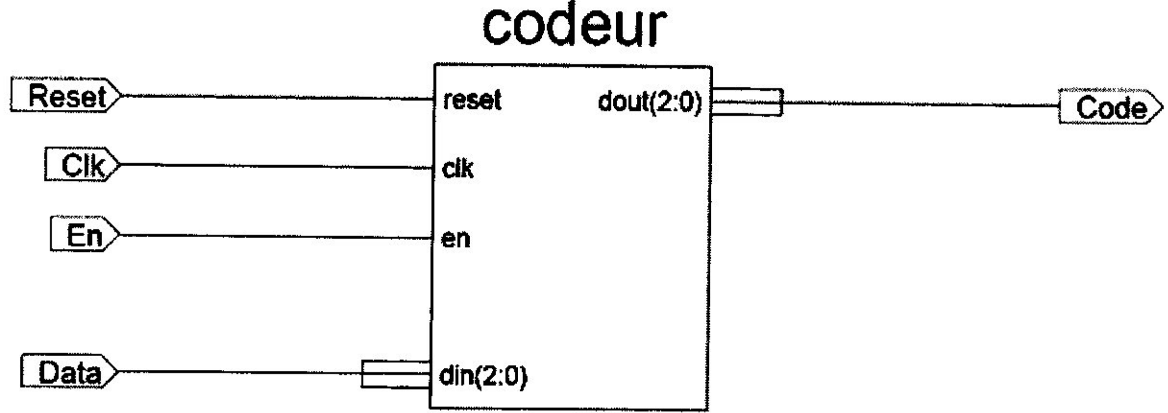


***Figure IV.14**: Schéma du codeur*

Le codeur est composé à la base d'éléments simples comme expliqué au chapitre **(II-3)** Les éléments base utilisés pour le codage sont:

- Additionneurs
- Bascules D
- Multiplicateurs en $GF(2^3)$
- Multiplexeurs

IV -7-3-2/ Signaux de commande et d'entrée/sortie du codeur

Les signaux utilisés par le codeur se divisent en deux catégories différentes :

- Les signaux de commandes
- Les signaux d'entrée/sortie

Les signaux de commande représentent tous les signaux permettant au codeur de fonctionner correctement. Ces signaux sont :

- clk : signal d'horloge (std_logic)

- Enable (En): signal qui permet d'activer le codeur. Ce signal est activé quand les symboles à coder arrivent et reste actif jusqu'à ce que le dernier symbole codé soit sorti du codeur (std_logic)
- Reset : signal permettant la remise à zéro du codeur (std_logic)

Les signaux d'entrée/sortie constituent tous les signaux que le système doit traiter. Ces signaux sont :

- Data : signal d'entrée des symboles à coder. Cette entrée est sur 3 bits.
  (std_logic_vector (2 downto 0))
- Code : signal de sortie des symboles codés. Cette sortie est sur 3 bits
  (std_logic_vector (2 downto 0))

IV -7-3-3/ Test Bench du codeur

On effectue un test bench du codeur, Les coefficients de référence sont calculés à l'aide du logiciel « Matlab » selon la routine rsenc(msg, n, k).

Les coefficients de référence pour le test bench:

$n = 7$

$k = 5$

$$\text{msg} = \begin{bmatrix} 0 & 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

La fonction de « Matlab » nous retourne le message codé suivant :

$$\text{Msg} = \begin{bmatrix} 0 & 0 & 0 & 1 & 3 & 0 & 4 \\ 0 & 0 & 0 & 0 & 2 & 7 & 6 \end{bmatrix}$$

IV-7-3-4/ performance du codage

IV-7-3-4-1/ Ressource hardware

Les ressources hardware utilisées pour un code de RS(7,5) après placement et routage à l'aide du logiciel XILINX 7.1i sur une carte XC95108-7-PC84 sont :

Macrocells Used 47 /108 (43%)

Pins Used 27 /69 (39%)

PIN RESOURCES:

Signal Type	Required	Mapped	Pin Type	Used	Remaining
Input	4	4	I/O	25	38
Output	21	21	GCK/IO	1	2
Bidirectional	0	0	GTS/IO	0	2
GCK	1	1	GSR/IO	1	0
GTS	0	0			
GSR	1	1			

Total	27	27			

IV-7-3-4-2/ débit binaire

Le débit binaire est calculé pour le code RS(7,5) avec FPGA XC95108-7-PC84. Les temps de propagation sont calculés à l'aide de logiciel Xilinx 7.1i.

L'horloge interne de la carte XC95108-7-PC84 est de fréquence f=39.2 [MHz], ce qui donne une période d'environ p ≅ 25.5[ns]. Le codage requiert 7 coups d'horloge, pendant cette période, 5 symboles d'information suivis par 2 symboles de contrôle seront sortis du codeur.

Comme on est dans le cas d'un $GF(2^3)$, on travaille sur 3 bits, donc on entrera au total pour le codage

b= nbr_symboles * nbr

b=5 * 3=15[bits]

Avec :

b : nombre total de bits rentrés dans le codeur

nbr_ symboles : nombre total de symboles rentrés dans le codeur

nbr : nombre de bits utilisés

On a un temps total de latence de :

t= nbr_hor * p

$t=7 * 25.5 * 10^{-9}=178.5$ [ns]

Avec :

t : temps de codage

nbr_ hor : nombre de coups d'horloge nécessaires pour le codage

p : période de l'horloge

Le débit binaire est défini comme :

$$d=\frac{b}{t}[bps] \qquad \textbf{(IV-1)}$$

Avec :

d : débit binaire en bits par seconde

b : nombre de bits

t : temps de codage

En se basant sur la relation **(IV-1)** on peut déduire le débit du codage :

$$d=\frac{b}{t}=\frac{15}{178,5*10^{-9}}=84033600 \simeq 84[Mbps]$$

IV-7-3-5/conclusion du codage

Le codeur fonctionne correctement selon la théorie exposée au chapitre **(II-3)** Le codage choisi est un code de RS(7,5) permettant de comparer les résultats théoriques avec les résultats pratiques.

La simulation temporelle confirme que la démarche suivie pour constituer le codeur est correcte.

Les simulations des différents blocs du codeur sont effectuées avec une fréquence d'horloge de f=39.2 [MHz], L'hardware n'a aucune influence sur la fréquence d'horloge choisie pour les simulations.

Le codage ne présente pas de vraies possibilités d'amélioration du débit binaire en changeant le type de codage ou les algorithmes, car il existe une seule façon de calculer les symboles de parité.

Pour améliorer sensiblement le débit binaire, on devrait changer le type de codage en prenant un codage plus performant, par exemple RS(255,223) ou des FPGA plus performantes. Plus la carte choisie est performante, plus les coûts seront élevés.

IV -7-4/ Décodeur

Le décodeur se compose de plusieurs blocs comme vus dans la figure **(II.4)**.

IV -7-4-1/ Syndrome_bloc

Le bloc qui calcule les symboles du polynôme du syndrome est :

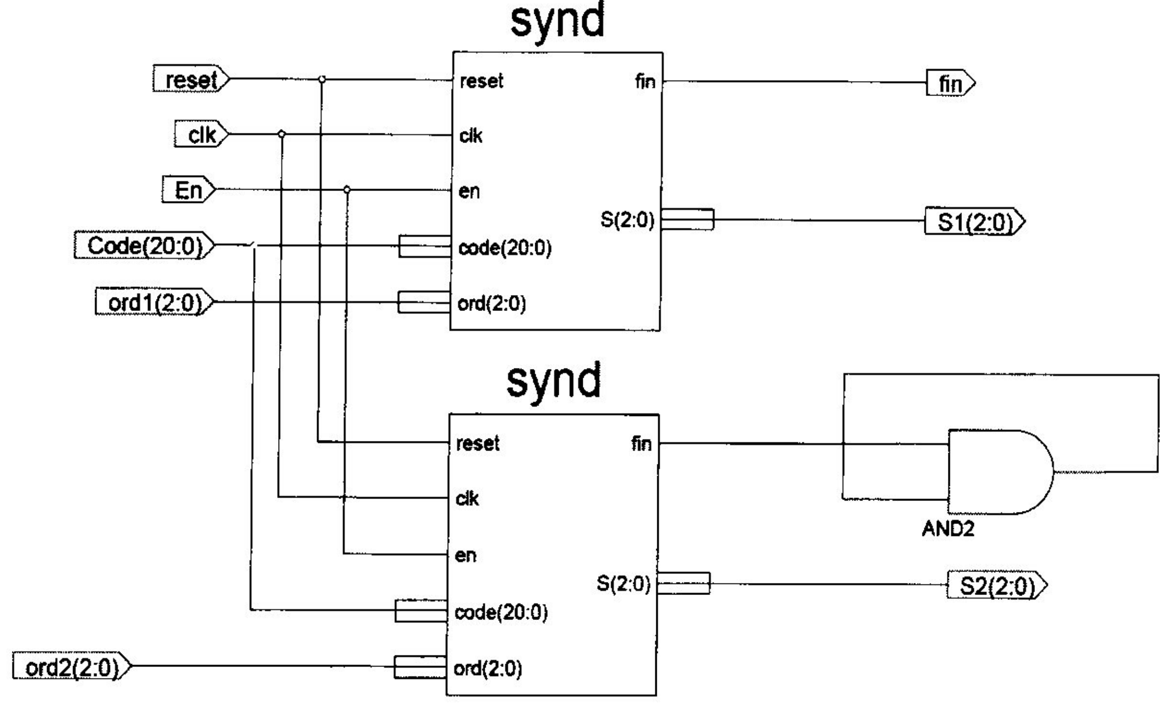


Figure IV.15:Schéma du syndrome_bloc

IV -7-4-2/ Signaux de commande et d'entrée/sortie syndrome _bloc

Les signaux de commande du syndrome_bloc sont:

- clk : signal d'horloge (std_logic)
- Enable (En) : signal qui permet d'activer le calcul des symboles du syndrome. (std_logic)
- reset : signal permettant la remise à zéro du blocs syndrome_bloc (std_logic)

Les signaux d'entrée/sortie du syndrome_bloc sont :

- Code : entrée des symboles du mot-code reçu. Cette entrée est sur 3 bits (std_logic_vector(2 downto 0))
- S1,S2: sortie des symboles du syndrome calculé. Cette sortie est sur 3 bits

(std_logic_vector(2 downto 0))

Ord :Entrée des racines des symboles du syndrome(std_logic_vector (2 downto 0))

fin : Sortie des symboles du syndrome(std_logic)

IV -7-4-3/ Test Bench

Le test bench effectué sur le syndrome_bloc est calculé avec «ISE SIMULATOR»

IV -7-4-4/ Conclusion du décodage

Dans le bloc de décodage, La majeure partie des opérations de ce bloc requiert beaucoup plus de ressources hardware que n'importe quel autre bloc traité ici.

Dans le cadre de notre travail, On n'a pas pu rendre fonctionnelle la partie de décodage en « VHDL » sauf le syndrome car on n'a pas eu le temps suffisant et à cause du langage qui est nouveau pour nous.

CONCLUSION GENERALE

CONCLUSION GENERALE

Les codes de Reed–Solomon sont aujourd'hui dans toute application nécessitant une correction d'erreurs, ceci est le cas de la communication aéronautique et spatiale.

Ce modeste travail nous a permis de se familiariser avec le codage Reed-Solomon ainsi que sa programmation sous VHDL.

La maîtrise du logiciel Xilinx nous a permis à son tour d'avoir une idée détaillée sur les étapes correspondantes à la réalisation d'une carte FPGA.

Finalement nous souhaitons que ce travail sera considéré comme un plus à notre département.

BIBLIOGRAPHIE

[1] Source: Cryptography and Network Security, Third Edition, William Stalling, Prentice Hall

Source: Error Control Coding, Second Edition, S. Lin et D.J. Costello, Prentice Hall

[2] Source: Cryptography and Network Security, Third Edition, William Stalling, Prentice Hall

Source: Error Control Coding, Second Edition, S. Lin et D.J. Costello, Prentice Hall

[3] Source: Reed – Solomon error correction, C. K. P Clarke, British Broadcast Corporation

[4] Source: Errror control coding, Modin

[5] Source: PGZ Decoder, John Gill, Stanford University

[6] Source: Self-correcting codes conquer noise, Reed – Solomon codecs, S. S. Shab, S. Yaqub, F. Suleman,designfeature

[7] Source : Self-correcting codes conquer noise, Reed – Solomon codecs, S. S. Shab, S. Yaqub, F. Suleman,designfeature

[8] Source: Error Control Coding, Second Edition, S. Lin et D.J. Costello, Prentice Hall

[9] Source: Error-Control Block Codes, L.H. Charles LEE, Artech House Publishers

[10] Source: Error Control Coding, Second Edition, S. Lin et D.J. Costello, Prentice Hall
Source: Error-Control Block Codes, L.H. Charles LEE, Artech House Publishers

[11] Source: Reed – Solomon decoder for HDTV application, E. Aleman, E. Galema, M. Kettledon

[12] D.Rabaste “ASIC et composants à réseaux logiques programmables : PAL, PLD, CPLD, FPGA ", IUFM d’AIX-Marseille, Mars 2002

[13] A.Nketsa “Circuits logique programmables “, ELLIPSES, 1998.

[14] C.Guex & E.Messerli”Circuits programmables et langages de conception, une évolution en parallèle”.Ecole d’ingénieur du canton de Vaud, Suisse, 1998

[15] M.Laurent "Applications des mémoires active programmables" Ecole polytechnique France, 1997.

[16] C.Guex & E.Messerli"Circuits programmables et langages de conception, une évolution en parallèle".Ecole d'ingénieur du canton de Vaud, Suisse, 1998

[17] D .Houzet. "Conception des circuits en VHDL, Principe et méthodologie",Paduès-Editions,Avril2000

[18] D .Houzet. "Conception des circuits en VHDL, Principe et méthodologie",Paduès-Editions,Avril2000

SITES INTERNET

http://www.stanford.edu/class/ee387/handouts/galois.pdf
http://www.bbc.co.uk/rd/pubs/whp/whp-pdf-files/WHP031.pdf
http://www.stanford.edu/class/ee387/handouts/lect24.pdf
http://www.4i2i.com/reed_solomon_codes.htm
http://www.stanford.edu/class/ee387/handouts/lect26.pdf
http://www.stanford.edu/class/ee387/handouts/chien.pdf
http://www.stanford.edu/class/ee387/handouts/lect23.pdf
http://direct.xilinx.com/bvdocs/whitepapers/wp110.pdf
http://www.ece.iit.edu/~niliev/gf_mult_2003_conf.pdf
http://www.edn.com/contents/images/315013.pdf
http://www.elektrobit.co.uk/pdf/reedsolomon.pdf
http://www.ee.nctu.edu.tw/course/channel_coding/chap6.pdf
http://www.math.msu.edu/~jhall/classes/codenotes/GRS.pdf
http://epubl.luth.se/1402-1617/2002/289/LTU-EX-02289-SE.pdf
http://www.dietler _3.pdf
http://www-rohan.sdsu.edu/~mosulliv/Courses/Coding02/RScodes.pdf
http://members.aol.com/mnecctek/faqs.htm
http://ieeexplore.ieee.org/iel3/30/13954/00642367.pdf?arnumber=642367
http://ieeexplore.ieee.org/iel4/5666/15173/00694898.pdf?arnumber=694898
http://ieeexplore.ieee.org/iel6/8361/26343/01168233.pdf?arnumber=1168233
http://portal.acm.org/citation.cfm?id=505522
http://www.imec.be/esscirc/ESSCIRC2002/PDFs/C29.01.pdf
http://soc.inha.ac.kr/mypapers/ITCAS2005_RSdec.pdf
http://soc.inha.ac.kr/mypapers/Eletter03_LEE.pdf
http://soc.inha.ac.kr/mypapers/itvlsi03_lee.pdf
http://soc.inha.ac.kr/mypapers/Eletter01.pdf
http://www.seasolve.com/products/reed-solomon/productinfo/1002.html