Contribution to the study of the common FFT operator in Software radio context: application to channel coding

Ali Al Ghouwayel

N° d'ordre : 3729

Thèse

présentée devant
l'UNIVERSITÉ DE RENNES I

pour obtenir le grade de
Docteur de l'Université de Rennes I
Mention : *Traitement du Signal et Télécommunications*

par
Ali AL GHOUWAYEL

Équipe d'accueil : Institut d'électronique et de télécommunications de Rennes
École doctorale : Matisse
Composante universitaire : S.P.M.

Contribution to the Study of the Common FFT Operator in Software Radio Context: Application to Channel Coding

Soutenue le 27 Mai 2008 devant la commission d'examen

Composition du jury

Rapporteurs	
M. David Declercq	Professeur des Universités, Université de Cergy Pontoise - ETIS, Cergy Pontoise
M. Guy Gogniat	Maître de conférences, HDR, Université de Bretagne Sud - Lab-STICC, Lorient
Examinateurs	
M. Emmanuel Boutillon	Professeur des Universités, Université de Bretagne Sud - Lab-STICC, Lorient
M. Yves Louët	Maître de conférences, IETR/Supélec, Rennes
M. Jacques Palicot	Professeur, IETR/Supélec, Rennes
M. Olivier Sentieys	Professeur des Universités, Université de Rennes 1 - ENSSAT, Lannion

Acknowledgments

First and foremost, I would like to thank Allah, praised and exalted is He, for giving me patience and helping me finish this thesis.

Many are those who have supported, encouraged, and helped me during this thesis, and it is my pleasure to acknowledge their guidance and support. I would like to begin by expressing my gratitude to my supervisor Yves Louët for his guidance and his continuous support in this thesis. I wish to express sincere thanks to my advisor, Professor Jacques Palicot, for his support all along the past three years.

I wish to thank Supélec and in particular the research director Bernard Jouga for providing me with generous financial support. I am grateful to my colleagues in the SCEE team for their patience and encouragement.

I am thankful to Professor Olivier Sentieys for honoring me by accepting to be the chair person of my committee. I am grateful to Professor Guy Gogniat and Professor David Declercq for their careful review of my work, and to Professor Emmanuel Boutillon for accepting to participate to my committee and his valuable remarks.

I would like to thank my cousin Ali Chamas (Bachelor of engineering in Australia) for his support and his contribution for reviewing and correcting the manuscript.

I would like to thank all my friends for their friendship and in particular Abdel-Majid Mourad, Ahmad Ghareeb, Mohamad Fakih and Mohamad Mroué.

Finally, I would like to dedicate this work to my parents, sisters Rola, Oula, Madeleine and Soumaya and brothers Mohamad, Hussein and Hassan for standing with me throughout both difficult and good times, and for their patience and love.

Ali Chamas Al Ghouwayel
May 27, 2008.

Contents

Abbreviations

ADC	Analog-to-Digital Converter
ADSL	Asymmetric Digital Subscriber Line
AFE	Analog Front End
AGC	Automatic Gain Control
AGU	Address Generating Unit
ALUT	Adaptive LUT
ALM	Adaptive Logic Module
BER	Bit Error Rate
BPF	Band Pass Filter
CDMA	Code Division Multiple Access
CF	Common Function
CLAA	Carry Look Ahead Adder
CO	Common Operator
CPA	Carry Propagate Adder
CRC	Cyclic Redundancy Check
CSA	Carry Save Adder
DAB	Digital Audio Broadcasting
DAC	Digital-to-Analog Converter
DECT	Digital Enhanced Cordless Telecommunications
DFE	Digital Front End
DFT	Discrete Fourier Transform
DIF	Decimation In Frequency
DIT	Decimation In Time
DMFFT	Dual Mode FFT
DVB-T	Digital Video Broadcasting-Terrestrial
FD	Frequency Domain
FER	Frame Error Rate
FFT	Fast Fourier Transform
FFT-$\mathbb{C}$	FFT defined over complex field
FFT-GF2	FFT defined over $GF(2^m)$
FIFO	First In First Out
FIR	Finite Impulse Response
FLMS	Fast Least Mean Squares
FNT	Fermat Number Transform
FPGA	Field Programmable Gate Array
GCU	Global Control Unit
GF	Galois Field

GMSK	Gaussian Minimum Shift Keying
GPS	Global Positioning System
GSM	Global System for Mobile communication
IDEA	International Data Encryption Algorithm
IF	Intermediate Frequency
IFFT	Inverse FFT
IIR	Infinite Impulse Response
ISDB	Integrated Services Digital Broadcasting
IS	Interim Standard for US code division multiple access
LAB	Logic Array Block
LAN	Local Area Network
LE	Logic Element
LNA	Low Noise Amplifier
LO	Local Oscillator
LSB	Least Significant Bit
LUT	Look Up Table
MAC	Multiplier ACcumulator
MAP	Maximum A Posteriori probability
MLSE	Maximum Likelihood Sequence Estimation
MSPS	Million Samples Per Second
NTT	Number Theoretic Tranform
OFDM	Orthogonal Frequency Division Multiplex
PE	Processing Element
PPA	Parallel Prefix Adder
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency
RNS	Residue Number System
RPE	Reconfigurable Processing Element
RS	Reed-Solomon
SCA	Software Communication Architecture
SCU	Stage Control Unit
SDR	Software Defined Radio
SFDR	Spurious Free Dynamic Range
SIMO	Single-Input Multi-Output
SQNR	Signal-to-Noise-Quantization-Noise Ratio
SWR	SoftWare Radio
TDMA	Time Division Multiple Access
TMFFT	Triple Mode FFT
TMVFFT	Triple Mode Velcro FFT
UFLMS	Unconstrained Frequency-domain Least Mean Squares
UMTS	Universal Mobile Telecommunication System
UTRA-FDD	UMTS Terrestrial Radio Access-Frequency Division Duplexing

Résumé en français

Introduction

La multiplication actuelle des services de télécommunications (voix, données, image, vidéo) amène les acteurs du marché à repenser les architectures des émetteurs/récepteurs pour pouvoir répondre à la flexibilité croissante demandée par les utilisateurs que nous sommes, à savoir, recevoir tout, partout et avec une qualité de service acceptable. Derrière cette contrainte forte se cache d'immenses challenges technologiques visant à concevoir un terminal mobile universel supportant plusieurs standards avec lequel le réseau sera transparent à l'utilisateur. On parle alors de terminal multistandard. Le domaine de la radio logicielle a ainsi pour objectif d'apporter des réponses à cela tant sur le plan de la convergence des réseaux, l'accès à de multiples interfaces air et la flexibilité des systèmes de traitement tant logiciels que matériels.

En 1995, Jo Mitola jeta les principes de la radio logicielle en proposant une architecture dite idéale : une antenne large bande suivie d'un convertisseur analogique/numérique à très haute fréquence d'échantillonnage permettant de traiter tout signal de façon numérique, à condition que les processeurs soient extrêmement rapides. Le terminal multistandard prend alors tout son sens si les processeurs sont dits reconfigurables (ou reprogrammables) par téléchargement de logiciels associés à la norme sur laquelle se base la télécommunication à passer.

Les obstacles technologiques sont cependant très grands, comme par exemple la réalisation de convertisseurs analogique/numérique ultra rapides, de processeurs à très forte puissance de calcul et à faible consommation ou d'amplificateurs de puissances et d'antennes très large bande.

L'aspect de la radio logicielle développé dans ce travail concerne celui de la paramétrisation. C'est un domaine nouveau dont l'objectif est de proposer des solutions pour réaliser un terminal multistandard en recherchant les traitements communs entre standards et en les mutualisant. Cette recherche aboutit à l'identification de fonctions et d'opérateurs communs, suivant le niveau de granularité voulu. Le challenge est ensuite de rendre ces traitements reconfigurables par passage de paramètres, d'où le nom de paramétrisation.

L'opérateur étudié dans cette thèse est la FFT, déjà identifiée dans [21]. Il est bien connu que de nombreux traitements peuvent être effectués de façon équivalente dans le domaine fréquentiel, justifiant ainsi l'utilisation d'une FFT commune et flexible. L'égalisation, la fonction de filtrage, l'estimation de canal ou la (dé)modulation OFDM sont des exemples de fonctions utilisant l'opérateur de FFT. L'objet de ce travail est alors d'étendre l'ensemble de ces fonctions au codage de canal (de type codes en blocs) en identifiant des

codes dont les traitements (codage et décodage) peuvent être réalisés avec les opérateurs de type FFT. Cela implique :

- l'adaptation de l'opérateur FFT pour le calcul de transformées dans des corps finis
- la réalisation et la validation d'une architecture flexible de FFT (dans le corps des complexes et dans les corps de Galois).

Les codes en blocs identifiés, permettant de conserver l'architecture de base de la FFT complexe pour ensuite l'adapter à des corps finis, sont les codes de Reed-Solomon définis sur les corps de Galois de type $CG(F_t)$ où F_t est le nombre de Fermat d'ordre t.

L'articulation du travail est alors la suivante :

Le chapitre 1 présente les principes de la radio logicielle et de la paramétrisation.

Le chapitre 2 présente les codes de Reed-Solomon définis dans $GF(F_t)$ et les différentes opérations de codage/décodage pouvant impliquer une FFT. Ce sont le calcul des syndromes et l'algorithme de Chien.

Le chapitre 3 propose une architecture de FFT réalisant à la fois des transformées dans $CG(F_t)$ et dans le corps des nombres complexes. C'est l'opérateur DMFFT pour Dual Mode FFT. Cette architecture repose sur une structure de type papillon dont tous les opérateurs ont été tous rendus reconfigurables. Elle a été implémentée sur FPGA (Stratix II) et nous avons montré qu'elle était plus optimale qu'une structure de type "Velcro" où les deux opérateurs (FFT complexe et FFT dans $CG(F_t)$) sont juxtaposés, sans souci de reconfiguration, le passage de l'un à l'autre se faisant par un simple "switch".

Enfin, le chapitre 4 propose des solutions pour réaliser un opérateur TMFFT (pour Tri Mode FFT) dont l'objectif est d'effectuer des transformées dans le corps des complexes, dans $CG(F_t)$ mais aussi dans $CG(2^n)$, intégrant ainsi les codes de Reed-Solomon classiques définis sur $CG(2^n)$.

1 La paramétrisation dans le contexte de la radio logicielle

1.1 Les architectures radio logicielles

La radio logicielle est un ensemble de techniques visant à répondre aux évolutions des radiocommunications. Suivant l'architecture de l'émetteur, on peut distinguer plusieurs déclinaisons du principe de la radio logicielle, de la structure superhétérodyne (Figure 1.a) à celle de la radio logicielle idéale (Figure 1.b) en passant par les deux versions de la radio logicielle restreinte (Figure 1.c et d).

La radio logicielle idéale est optimale car la conversion numérique analogique est directement effectuée en radio fréquence, suivie de processeurs. La radio logicielle restreinte est celle correspondant à une numérisation en fréquence intermédiaire (ou bande de base dans le cas extrême). Dans le cas idéal, une large bande de fréquence est alors directement numérisée englobant ainsi plusieurs signaux associés à différents standards. Cette

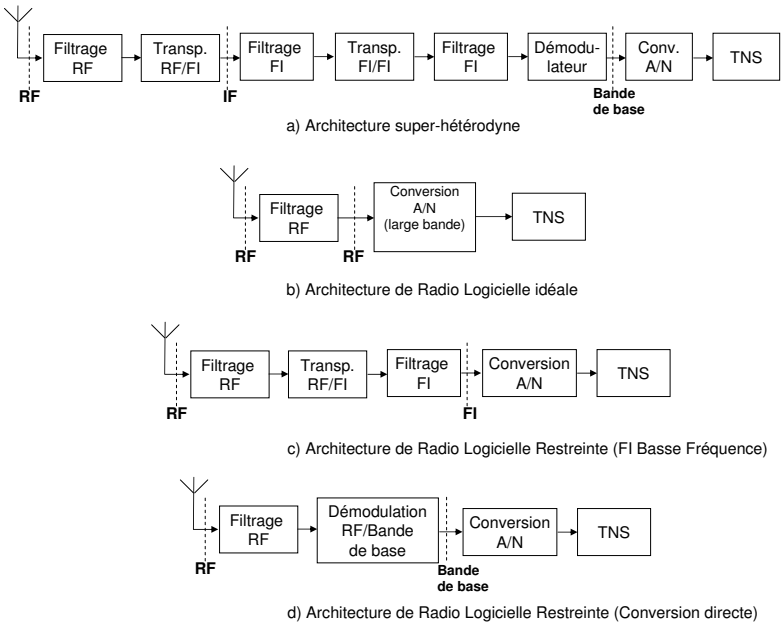

Figure 1: Les quatre principales architectures de récepteurs radio

fonctionnalité fait alors référence à un système dit multistandard, capable d'opérer selon différents modes et offrant plusieurs services. Le passage d'un mode à un autre est alors possible à condition que les processeurs de traitement du signal soient reconfigurables par téléchargement des logiciels associés aux normes des standards.

Afin de réaliser d'une façon optimale un tel système, il est nécessaire de rechercher les points de convergence entre les standards à supporter par le terminal de façon à rendre les traitements communs. Cet aspect est connu sous le nom de la paramétrisation et est développé dans la partie suivante.

1.2 Principes de la paramétrisation

Le principe de la paramétrisation est de rechercher dans un premier temps les caractères communs entre les traitements de différents standards puis d'en proposer des architectures communes et flexibles, à l'opposé d'une approche de type Velcro. La paramétrisation se décline selon deux approches : l'approche théorique et l'approche pragmatique. L'approche théorique consiste à lister de façon hiérarchique tous les appels de fonctions possibles dans un terminal (par exemple, la modulation OFDM fait appel à la FFT, qui elle même fait appel à l'opérateur papillon et ce dernier faisant appel aux opérateurs arithmétiques classiques). De cette façon, on montre en se basant sur la théorie des graphes qu'il est possible de choisir un chemin optimal (selon une fonction de coût/performance à définir) permettant de privilégier l'appel à certains opérateurs, qui deviennent alors communs.

L'autre approche, dite pragmatique, consiste dans un premier temps à identifier les traitements communs puis dans un second temps à réaliser un opérateur générique qui devra alors être reconfigurable. C'est cette approche qui a été privilégiée dans ce travail.

Ces traitements communs se déclinent en fonctions ou opérateurs communs.

1.2.1 Approche par fonctions communes

Le premier type de paramétrisation (et le plus naturel) se fait au niveau fonction. Le niveau de granularité est élevé. Bien souvent, les travaux ayant eu pour objet de proposer des fonctions communes aboutissaient à des opérateurs plus génériques que reconfigurables. Les structures proposées telles que [19], [20] et [27] sont certes communes mais prohibitives en terme de surface et surtout n'exploitent pas vraiment le principe de reconfiguration. Elles sont de plus peu évolutives car reliées à des standards prédéfinis. Cet aspect implique de diminuer le niveau de granularité. C'est l'approche opérateur commun.

1.2.2 Approche par opérateurs communs

La granularité est ici plus faible que celle des fonctions communes. L'idée est de tirer parti d'une structure de calcul existante et de la faire évoluer de façon à la rendre commune et reconfigurable. La réutilisation devient alors optimale. Par exemple, un opérateur de type filtrage a été identifié dans [29] permettant de réaliser des opérations de codage de canal ou de calcul de CRC. C'est l'opérateur MulDiv. Cette étude a été étendue pour aboutir à l'opérateur proposé dans [31].

Un autre opérateur a été identifié comme opérateur commun dans [21] : il s'agit de l'opérateur FFT. Ce dernier se retrouve en effet dans bons nombres de traitements tels que la (dé)modulation OFDM, l'égalisation, le filtrage, etc. L'objectif de cette thèse est d'étendre l'application de cet opérateur au codage de canal. Ainsi, la structure de l'opérateur de FFT a été repensée et étendue afin de réaliser aussi des transformées de Fourier dans des ensembles tels que les corps finis de Galois. L'application choisie est le codage de canal et en particulier une classe particulière des codes de Reed-Solomon (RS) dont les mots de codes sont de longueur 2^n et permettant ainsi d'exploiter la structure classique de FFT pour le codage et certaines opérations du décodage.

L'objet de la section suivante est de présenter l'intérêt de la FFT pour traiter ces codes RS.

2 La FFT et le codage de canal

2.1 Transformée de Fourier dans les corps finis

De la même façon qu'il existe des tranformées de Fourier définies sur des ensembles infinis (tel que le domaine des nombres complexes $\mathbb{C}$) utilisées classiquement en traitement du signal, il existe aussi des transformées de Fourier sur des ensembles finis (tels que les corps de Galois $CG(q)$ utilisés en particulier dans le traitement des codes cycliques.

Dans $\mathbb{C}$, la transformée de Fourier discrète d'un vecteur $v = (v_0, v_1, ..., v_{N-1})$ de nombres réels ou complexes est un vecteur $V = (V_0, V_1, ..., V_{N-1})$ défini par:

$$V_k = \sum_{i=0}^{N-1} e^{-j\frac{2\pi ik}{N}} v_i \qquad k = 0, ..., N-1 \tag{1}$$

N étant un entier représentant la longueur de la transformée et $j = \sqrt{-1}$. La base de Fourier $exp(-j2\pi/N)$ est la *Nème* racine de l'unité dans $\mathbb{C}$. Dans un corps de Galois CG(q), l'élement primitf α d'ordre N est la *Nème* racine de l'unité. Par analogie entre $exp(-j2\pi/N)$ et α, considérons un vecteur $v = (v_0, v_1, ..., v_{N-1})$ dans $CG(q)$ et α un élément d'ordre N de ce corps. Le vecteur v et sa transformée de Fourier sont reliés par les équations suivantes [44]:

$$V_j = \sum_{i=0}^{N-1} \alpha^{ij} v_i \qquad \Longleftrightarrow \qquad v_i = \frac{1}{N} \sum_{j=0}^{N-1} \alpha^{-ij} V_j, \tag{2}$$

pour $j = 0, ..., N-1$.

Le tableau 1 montre alors les simulitudes entre les opérations de FFT dans $\mathbb{C}$ et de FFT dans $CG(q)$.

Table 1: Simulitudes entre les opérations de FFT dans $\mathbb{C}$ et de FFT dans $CG(q)$.

	dans $\mathbb{C}$	dans $CG(q)$
Transformée d'un vecteur v	$V_k = \sum_{i=0}^{N-1} e^{-j\frac{2\pi ik}{N}} v_i, \quad k$=0,...,$N$-1	$V_j = \sum_{i=0}^{N-1} \alpha^{ij} v_i, \quad j$=0,...,$N$-1
Noyau de la transformée	$exp(-j2\pi/N)$	α (racine primitive du corps de Galois)

2.2 Traitement fréquentiel des codes Reed-Solomon définis sur $CG(q)$

Les transformées de Fourier définies dans $CG(q)$ ont été introduites dans l'étude des codes cycliques dans un souci de réduction de complexité des décodeurs par Gore [46] et puis par Michelson [47], Winograd [48] et Chien [49]. Plus tard, Blahut [50], afin d'optimiser l'utilisation des transformées de Fourier a traduit le processus de codage (classiquement effectué en temporel) dans le domaine fréquentiel. Il a aussi adapté les différents algorithmes de décodage de façon à être réalisés dans le domaine fréquentiel.

Le principe du codage d'un code $C(n, k)$ proposé par Blahut consiste à former un mot d'information de longueur k dans le domaine fréquentiel dans lequel $2t$ composantes prédéterminées sont fixées à 0 (t : pouvoir de correction du code). Ensuite, le mot de code temporel de longueur n est obtenu à l'aide d'une transformée inverse de Fourier.

Pour le décodage, la transformée de Fourier peut être utilisée pour le calcul des deux étapes les plus longues, à savoir le calcul des syndromes et l'algorithme de Chien.

Partant de la structure de base de l'opérateur FFT définie dans $\mathbb{C}$, l'objectif de ce travail est de la faire évoluer afin qu'elle puisse aussi réaliser des transformées dans les corps finis pour le codage de canal. On parle alors d'un opérateur commun et reconfigurable. Les codes cycliques considérés dans ce travail sont les codes de Reed-Solomon (RS).

D'après ce qui a été dit précédement, la réalisation de cet opérateur commun implique alors :

- que la longueur des transformées soit une puissance de 2
- que les opérations arithmétiques requises par les deux modes utilisent les mêmes ressources.

Cependant, dans les applications actuelles, les codes RS utilisés sont définis dans $CG(2^m)$ et leurs transformées sont donc de longueur $2^m - 1$ et non 2^m. Nous avons donc été amené à considérer des codes RS dont les mots de code sont de longueur 2^m. La classe de codes RS identifiée dans ce travail est celle des codes RS définis sur $CG(F_t)$ où F_t est un nombre de Fermat de la forme $2^{2^t} + 1$. Sachant bien que la structure arithmétique des $CG(F_t)$ est plus complexe que celle des $CG(2^m)$, le choix de ces codes RS définis sur $CG(F_t)$ traités dans le domaine fréquentiel se justifie par la réutilisation d'une structure de transformée pré-existante, à savoir celle de la FFT complexe.

Il est à noter que les codes RS définis sur $CG(F_t)$ ont été recommandés par l'agence spatiale européenne (ESA) pour des télécommunications spatiales [69]. Il s'agissait d'un code RS(256,224) défini dans CG(257) concaténé avec un code convolutif.

Lorsque la transformée est effectuée sur les corps finis $CG(F_t)$, on parle de la transformée de Fermat (Fermat Number Transform (FNT)). Cette transformée présente de nombreux avantages. Sa structure est alors identique à celle de la FFT. Par conséquent, les mêmes algorithmes peuvent être utilisés pour les FFT aussi bien que pour la FNT, à condition que la FFT puisse opérer modulo(F_t).

2.3 Intérêt du traitement fréquentiel pour le décodage des codes RS sur $CG(F_t)$

Comme pour les codes RS classiques (définis sur $GF(2^m)$), le décodage des codes RS définis sur $CG(F_t)$ s'effectue selon trois phases :

- Phase 1 : calcul des syndromes
- Phase 2 : algorithme de Berlekamp
- Phase 3 : algorithmes de Chien et de Forney

Le calcul des syndromes et l'algorithme de Chien sont les phases les plus longues du décodage. L'exécution de chacune d'entre elles nécessite n cycles, avec n la longueur du mot de code. Théoriquement, en utilisant une FNT de longueur $n = 2^{2^t}$, ce temps d'exécution sera réduit à $log(n)$ cycles à condition que la FNT soit implémentée avec sa structure entière comportant $log(n)$ étages de $\frac{n}{2}$ papillons. Cependant, puisque cette approche théorique est directement liée à la réalisation pratique de l'opérateur FNT, cette réduction de temps de calcul n'est pas réaliste puisque l'implémentation de la structure entière d'une FFT consomme environ les deux tiers d'un FPGA (dans le cas de $n = 16$ sur un composant Stratix II).

L'intérêt des codes RS définis sur $CG(F_t)$ a été détaillé. Il faut maintenant établir leurs performances en terme de taux d'erreur binaire. C'est ce qui fait l'objet de la section suivante.

2.4 Performances des codes RS définis sur $CG(F_t)$

Les performances en terme de taux d'erreur binaire sont présentées pour les codes RS définis sur $CG(F_t)$ et sur $CG(2^m)$ et ceci pour des capacités de correction identiques. Elles sont établies sur des canaux gaussiens. La figure 2 donnent les performances pour quatre types de codes :

- (i) système non codé
- (ii) RS(16,12) défini sur $CG(17)$ avec des codages et décodages fréquentiels. Tous les symboles des mots de codes sont représentés sur 5 bits.
- (iii) cas identique au précédent mais avec codage temporel, décodage fréquentiel. Les symboles de parité et les symboles d'information sont représentés sur 4 bits (4-4).
- (iv) cas identique au précédent (iii) mais seuls les symboles de parité sont représentés sur 5 bits et les symboles, d'information sur 4 bits (5-4), décodage fréquentiel.
- (v) RS(15,11) défini sur $CG(16)$, décodage fréquentiel

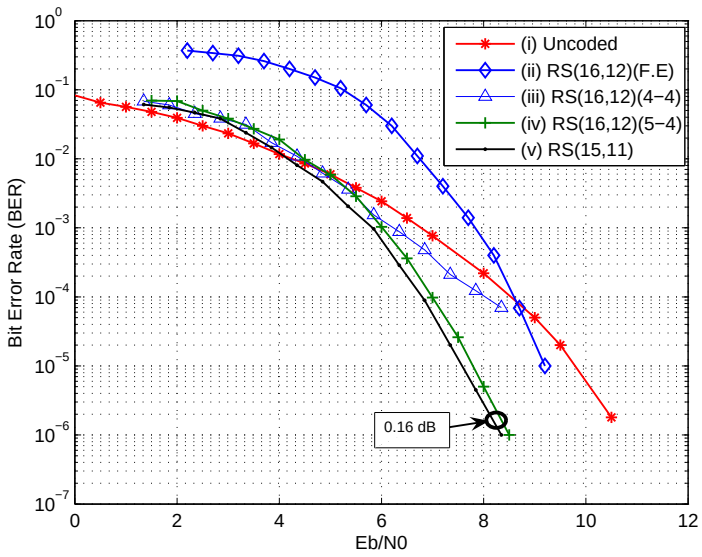

Figure 2: Performances des différents codes RS sur $CG(17)$ et $CG(16)$.

Les conclusions liées à ces simulations sont les suivantes. D'une part, le codage fréquentiel dégrade les performances des codes. Il est donc préférable d'utiliser un codage temporel. La raison vient du fait que le mot de code n'est pas systématique et qu'une erreur

sur un symbole à la réception véhicule, du fait de la transformée, un nombre important d'erreurs.

D'autre part, les performances des codes RS sur $CG(F_t)$ sont semblables à celles des codes RS sur $CG(2^m)$ à condition que les symboles de parité soient émis sur $m+1$ bit. Ceci est un argument supplémentaire pour justifier leur utilisation dans un futur terminal reconfigurable utilisant des FFT comme opérateur commun.

Nous allons maintenant aborder la réalisation de l'opérateur FFT reconfigurable permettant d'effectuer des transformées dans $\mathbb{C}$ et $CG(F_t)$. Cet opérateur est le DMFFT pour Dual Mode FFT.

3 Architecture de l'opérateur DMFFT

Le chapitre 3 se focalise sur l'implémentation de l'opérateur DMFFT. Cet opérateur dual mode est un opérateur reconfigurable capable d'effectuer deux types de transformées : FFT et FNT. Le passage d'un mode à un autre est assuré par la mise à jour de certains paramètres qui entraîne la reconfiguration des interconnections et des opérateurs arithmétiques constituant le DMFFT.

L'implémentation du DMFFT est basée sur celle de la FFT classique dans $\mathbb{C}$. L'idée de base est d'exploiter la structure de l'opérateur FFT complexe en termes de bus de routage des flux de données, de mémorisation des calculs intermédiaires et des racines de Fourier ainsi que des ressources arithmétiques disponibles dans la structure.

En général, il y a plusieurs façons d'implémenter les algorithmes de FFT dans $\mathbb{C}$. La plupart des travaux dans ce domaine ont été réalisés en utilisant des DSP [71] [72] et des processeurs dédiés [73]. Cependant, grâce à leur grande capacité de calcul et leur prix économique, les composants FPGA représentent aujourd'hui une solution efficace pour implémenter des agorithmes nécessitant des calculs intensifs.

Le calcul de la transformée de Fourier rapide fut initié avec la publication par Cooley et Tuckey d'un algorithme permettant la réduction du complexité de calcul de $O(N^2)$ à $O(N\ log\ N)$, N représentant la longueur de la transformée. Cet algorithme peut être implémenté avec différentes racines : radix-2, radix-4 ou radix-16. Un ordre élevé de radix permet de réduire le nombre de multiplications mais la structure ne sera plus régulière. Pour réduire la complexité de calcul, des algorithmes utilisant deux racines différentes ont été développés sous le nom split-radix. Pour plus de détails sur les différents algorithmes, le lecteur pourra se référer à [79].

Dans cette étude nous avons choisi d'implémenter l'algorithme radix-2 puisqu'il offre certaines avantages en terme de régularité de la structure et la simplicié des calculs dans l'opération *papillon*. L'objectif est de valider l'architecture de l'opérateur reconfigurable que nous proposons et d'évaluer sa complexité et ses performances en terme de temps de calcul.

3.1 L'architecture du papillon type-FFT

Le coeur de calcul de l'opérateur FFT classique est connu sous le nom *papillon*. Le schéma de ce papillon classique est illustré sur la figure 3. Il est composé de trois opérateurs arithmétiques : multiplieur, additionneur et soustracteur. Dans le mode calcul de FFT

dans $\mathbb{C}$, ces opérateurs réalisent les opérations binaires sur des nombres complexes (ou réels). Etant donné que l'opérateur FFT devra réaliser des transformées dans $CG(F_t)$, l'architecture du papillon doit être redéfinie au niveau des opérateurs de façon à pouvoir opérer dans $CG(F_t)$. Dans ce dernier mode de fonctionnement, la racine de Fourier est remplacée par la puissance correspondante de l'élément primitf α comme le montre la figure 3. Chacun des opérateurs doit être conçu d'une façon reconfigurable lui permettant d'exécuter des opérations dans $\mathbb{C}$ ainsi que dans $CG(F_t)$. Dans les paragraphes suivants nous allons présenter les architectures reconfigurables de ces trois opérateurs. Leur conception s'est basée sur l'architecture des opérateurs arithmétiques pré-implémentés dans le papillon complexe puisque l'idée de base est d'exploiter une structure FFT déjà existante.

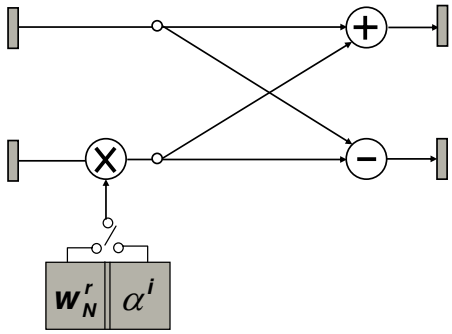

Figure 3: L'architecture du papillon FFT/FNT

3.2 L'additionneur reconfigurable proposé

Il existe différentes architectures des additionneurs binaires dont les prinicipales sont les additionneurs à propagation de retenue (CPA), les additionneurs à retenue conservée (CSA) et les additionneurs à retenue anticipée (CLA).

Les additionneurs modulo F_t sont dérivés des architectures des aditionneurs binaires normaux.

Considérons un nombre $A = a_{n-1}a_{n-2}...a_0$ écrit sous la forme:

$$A = \sum_{i=0}^{n-1} 2^i a_i.$$

La réduction modulo $2^n + 1$ de A peut être réalisée soit par une division et le reste de la division constitue le résultat, soit par une soustraction itérative du modulo jusqu'au $A < 2^n + 1$. Ayant

$$2^n \ mod \ (2^n + 1) = 2^n - (2^n + 1) = -1,$$

la réduction modulo $2^n + 1$ peut être reformulée de la façon suivante:

$$A \ mod \ (2^n + 1) = (A \ mod \ 2^n - A \ div \ 2^n) \ mod \ (2^n + 1), \tag{3}$$

où $A \ mod \ 2^n$ et $A \ div \ 2^n$ correspondent respectivement aux mots binaires de taille n bits de poids le moins significatif et de poids le plus significatif. Le terme modulo $2^n + 1$ à

droite de l'équation 3 est utilisé pour la correction finale dans le cas où un résultat négatif est obtenu.

Dans la littérature, les travaux les plus importantes dédiés à la réalistation des additionneurs modulo $2^n \pm 1$ sont basés sur les architectures des additionneurs CLA en utilisant la représentation diminuée de 1 des nombres traités [85][86][87][88].

Nous considérons ici les travaux développés par Zimmermann [85]. En se basant sur une architecture PPA (Parallel Prefix Adder) et en utilisant la représentation diminuée de 1, Zimmermann a proposé un multiplieur modulo $2^n + 1$ dont le principe est décrit ci après.

Soient A et B deux nombres quelconques. Avec une représentation diminuée de 1, ils peuvent s'écrire sous la forme $A' = A - 1$ et $B' = B - 1$. Leur somme diminuée de 1 ($S' = S - 1$) peut s'écrire

$$S' = A' + B' + 1,$$

et l'addition modulo $(2^n + 1)$ peut être reformulée avec l'équation suivante:

$$\begin{aligned} (A' + B' + 1)\ mod\ (2^n + 1) &= \begin{cases} (A' + B')\ mod\ 2^n & \text{if } A' + B' \geq 2^n \\ (A' + B' + 1)\ mod\ 2^n & \text{if } A' + B' < 2^n \end{cases} \\ &= (A' + B' + \overline{c}_{out})\ mod\ 2^n. \end{aligned} \tag{4}$$

L'architecture matérielle de l'équation 4 proposée par Zimmermann [85] consiste à réinjecter la retenue finale de l'addition binaire normale dans le dernier bloc de l'additionneur PPA. Pour éviter les effets des boucles combinatoires, Zimmerman a proposé d'insérer un étage de calcul supplémentaire. Pour des opérandes de petites tailles et moyennes, l'architecture proposée par Zimmermann constitue un bon compromis complexité-temps de calcul. Pour des tailles d'opérandes plus grandes, d'autres architectures ont été proposées [86][87][88]. Ces architectures traitent les nombres avec leurs représentations diminuées de 1. Cependant cette représentation souffre du problème de multiples interprétations des sorties égales à zéro. Une sortie égale à zéro peut représenter un vrai résultat nul ou bien erroné. Pour remédier à ce problème, il est nécesaire d'utiliser un circuit pouvant détecter le résultat correct [87].

En plus, la représentation diminuée de 1 nécessite des circuits de conversion (représentation binaire normale vers représentation diminuée de 1 et l'inverse) en utilisant des incrémentation/décrémentation qui pourront être dans certains cas plus coûteuses comparées aux avantages que cette représentation présente.

Tenant compte de ces deux inconvénients et du fait que l'additionneur modulo $2^n + 1$ que l'on compte réaliser sera implémenté dans la structure de l'additionneur complexe pré-implémentée dans le papillon, nous avons choisi de réaliser l'additionneur modulo $2^n + 1$ suivant une autre technique utilisant deux additionneurs [90]. Cette technique permet d'une part d'exploiter les additioneurs disponibles dans le papillon et d'autre part de profiter des performances de ces additionneurs cablés et pré-implémentés dans les composants FPGA. Quant au temps de calcul, le chemin critique pourra être réduit en utilisant un étage de pipeline. Tous ces paramètres seront évalués lors de l'implémentation FPGA.

Dans [90], l'auteur propose quelques architectures pour réaliser un additionneur modulo $2^n + 1$ basée sur l'idée d'utiliser deux additionneurs, le premier pour réaliser l'addition

binaire normale et le deuxième pour réaliser la réduction modulo $2^n + 1$. La meilleure architecture proposée dans [90] qui traite des nombres avec leur représentation binaire normale sur $n + 1$ bits est constituée de deux additioneurs et d'une porte NOR et elle fournit la somme incrémentée de 1.

Nous avons proposé une architecture d'un additionneur modulo $2^n + 1$ basée sur le même principe présenté dans [90] et qui fournit la somme correcte. L'addition modulo $2^n + 1$ est décrite par l'équation suivante:

$$(x+y)\ mod\ (2^n+1) = \begin{cases} (x+y)\ mod\ 2^n & \text{si } 0 \leq x+y < 2^n \\ (x+y)\ mod\ 2^n + 2^n - 1 & \text{si } 2^n < x+y \leq 2^{n+1} \\ 2^n & \text{si } (x = 2^n \text{ et } y = 0) \\ & \text{ou } (x{=}0 \text{ et } y = 2^n). \end{cases} \quad (5)$$

En d'autres termes,

$$(x+y)\ mod\ (2^n+1) = \overline{S_n^2} S_2 + S_n^2 2^n, \quad (6)$$

où S^2 représente la somme du deuxième additionneur:

$$S^2 = [S_{n+1}^2 S_n^2 ... S_0^2] = [S_{n-1}^1 ... S_0^1] + (2^n - 1)(S_{n+1}^1 \vee S_n^1),$$

et S^1 la somme du premier additionneur:

$$S^1 = [S_{n+1}^1 S_n^1 ... S_0^1] = x + y.$$

La démonstration de l'équation 6 est expliquée dans le chapitre 3. Sa réalisation matérielle est illustrée sur la figure 4.a. L'implémentation de cet additionneur modulo $2^n{+}1$ dans l'architecture de l'additioneur complexe mène à l'architecture reconfigurable de la figure 4.b. L'opérateur résultant est un additionneur ayant deux modes de fonctionnement, dans $CG(F_t)$ et dans $\mathbb{C}$. Le passage d'un mode à un autre est assuré par un signal de contrôle DM.

3.3 Le soustracteur reconfigurable proposé

L'architecture classique d'un soustracteur binaire est basée sur l'architecture de l'additionneur. Pour réaliser une soustraction en complément à 2 à l'aide d'un additionneur, il suffit d'inverser l'opérande à soustraire et de mettre la retenue d'entrée à 1. Pour la soustraction modulo $2^n + 1$, une légère modification de l'architecture de l'additionneur conduit à l'opérateur représenté sur la figure 5.a dont le principe de fonctionnement est décrit par l'équation suivante:

$$(x-y)\ mod\ (2^n+1) = \begin{cases} 2^n & \text{si } (x = 2^n \text{ and } y = 0) \\ (x + \overline{y} + 1 + S_n)\ mod\ 2^n & \text{sinon.} \end{cases} \quad (7)$$

La figure 5.b représente le soustracteur reconfigurable contrôlé par le signal DM permettant de choisir le mode de fonctionnement dans $CG(F_t)$ ou bien dans $\mathbb{C}$.

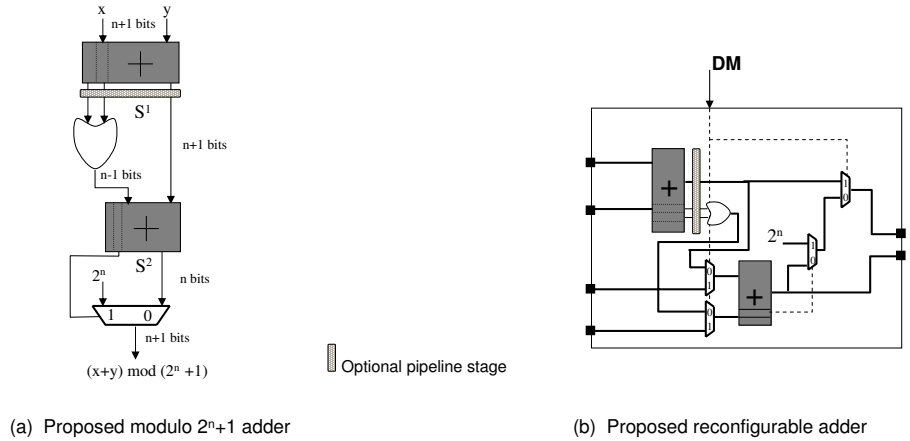

Figure 4: L'additionneur modulo $2^n + 1$ et l'additionneur reconfigurable

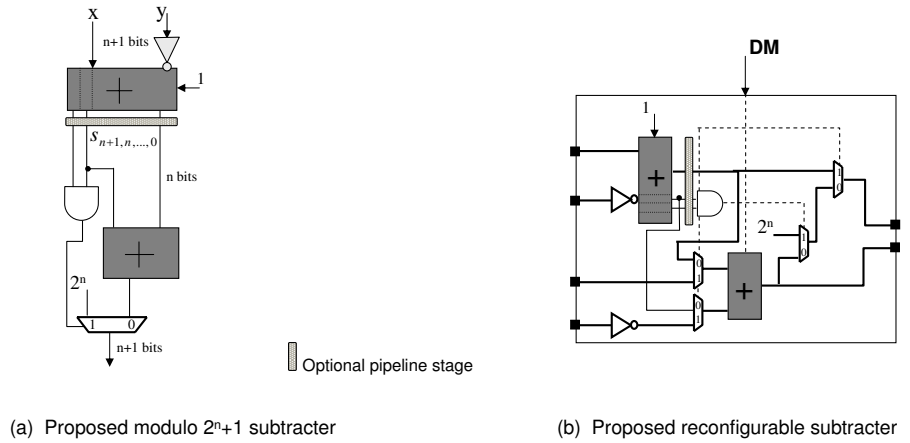

Figure 5: Le soustracteur modulo $2^n + 1$ et le soustracteur reconfigurable

3.4 Le multiplieur reconfigurable proposé

La multiplication modulo $2^n + 1$ est largement utilisée dans les systèmes des nombres résiduels (RNS) [81] et dans la cryptographie [91]. Plusieurs algorithmes ont été développés pour réaliser des multiplieurs modulo $2^n + 1$. Parmi ces algorithmes, certains sont basés sur des éléments de base de bas niveau tel que le *full-adder* et les portes logiques (*NAND/XOR*), et d'autres sont dédiés à des implémentations sur des composants FPGA. Ce dernier type d'algorithme permet de bénificier des avantages qu'offrent les composants FPGA récents, tels que l'intégration des ressources arithmétiques comme les multiplieurs binaires.

Les techniques de multiplication modulo $2^n + 1$ peuvent être divisées en trois classes:

1. multiplication à l'aide des Look Up Tables (LUT).

2. multiplication avec un multiplieur $(n+1)*(n+1)$ bits.

3. multiplication basée sur des additionneurs *Carry Save*.

Dans la première méthode, les produits de multiplication modulo 2^n+1 sont sauvegardés dans des tables et le résultat souhaité est indexé par une adresse formée par les deux opérandes. La taille des tables croît exponentiellement avec la longueur du mot et pour des valeurs de $n \geq 8$, la taille de mémoire nécessaire devient très grande ce qui rend cette méthode peu intéressante [92]. Pour réduire la taille des tables, des méthodes ont été proposées dans [94] [95]. Ces méthodes permettent de réduire la taille des mémoires de $O(2^{2n} \times n)$ à $O(2^n \times n)$. Malgré cette réduction, cette méthode basée sur des LUT ne pourra pas être une alternative pour réaliser des multiplications modulo 2^n+1 pour des grandes valeurs de n.

Dans la deuxième méthode utilisant un multiplieur $(n+1)*(n+1)$ bits, les algorithmes développés traitent les nombres dans l'ensemble $\mathbb{Z}^*_{2^n+1} = \{a \mid 1 \leq a \leq 2^n\}$, où le nombre 2^n est représenté par le nombre 0. L'algorithme de base est l'algorithme *Low-High* décrit par l'équation suivante [93]:

$$x \odot y = (c_L - c_H)\ mod\ (2^n+1) = \begin{cases} (c_L - c_H)\ mod\ 2^n & \text{si } c_H \leq c_L \\ (c_L - c_H + 1)\ mod\ 2^n & \text{si } c_H > c_L, \end{cases} \tag{8}$$

où $\odot$ représente la multiplication modulo F_t et où

$$c_L = \sum_{i=0}^{n-1} p_i 2^i \quad et \quad c_H = \sum_{i=0}^{n-1} p_{n+i} 2^i,$$

représentent respectivement les mots de n-bit de poids plus faible et plus fort.

L'implémentation directe de l'équation 8 exige un multiplieur $n \times n$-bits, trois soustracteurs, un additionneur, un comparateur et un multiplexeur. Des reformulations de l'équation 8 ont été proposées permettant l'amélioration de l'architecture matérielle du multiplieur. Nous citons ici la reformulation introduite par Beuchat [93]. L'auteur a suggéré d'utiliser un multipieur $(n+1) \times (n+1)$-bits en décomposant le produit

$$P = P_{2n} 2^{2n} + 2^n \sum_{i=0}^{n-1} p_{n+i} 2^i + \sum_{i=0}^{n-1} p_i 2^i,$$

et d'écrire la multiplication modulo 2^n+1 sous la forme

$$x \ \odot \ y = \begin{cases} (c'_L + \overline{c'_H} + 2)\ mod\ 2^n & \text{si } P_{2n} = 1 \text{ or } (P_{2n} = 0 \text{ et } c'_L + \overline{c'_H} + 1 < 2^n), \\ (c'_L + \overline{c'_H} + 1)\ mod\ 2^n & \text{ailleurs,} \end{cases} \tag{9}$$

avec $c'_L = \sum_{i=0}^{n-1} p_i 2^i$ et $c'_H = \sum_{i=0}^{n-1} p_{n+i} 2^i$. L'architecture matérielle réalisant l'équation 9 est illustrée sur la figure 6.

Avec cette architecture, la multiplication xy est réalisée avec un multiplieur $n \times n$-bits et ensuite la réduction modulo 2^n+1 est faite à l'aide de deux additionneurs.

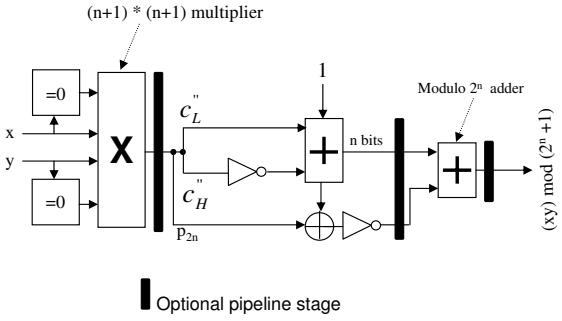

Figure 6: Le multiplieur modulo $2^n + 1$ [93]

Dans la troisième méthode basée sur les additionneurs *Carry Save*, la réduction modulo 2^n+1 est exécutée au niveau de chaque produit partiel $x_i y$. Plusieurs architectures ont été proposées [85] [89] [92]. Le circuit correspondant est constitué d'un générateur des produits partiels, de deux additionneurs *Carry Save* modulo $2^n + 1$, d'une unité de correction et d'un multiplexeur.

Considérons maintenant le choix entre les deux dernières méthodes. Dans [92], les auteurs ont comparé l'architecture basée sur le multiplieur $(n + 1) * (n + 1)$-bit à celle basée sur les additionneurs *Carry Save* et ont montré que la première architecture offre un excellent compromis complexité-temps de calcul. Dans [93], l'auteur a montré, en implémentant les deux mêmes classes de multiplieurs sur des composants FPGA Virtex-II, que le multiplieur modulo 2^n+1 basé sur le multiplieur $(n+1)*(n+1)$-bit offre un gain en termes de complexité et temps de calcul par rapport à un multiplieur modulo 2^n+1 basé sur des additionneurs *Carry Save*. Ce gain diminue et la deuxième méthode devient plus avantageuse quand les multiplieurs sont implémentés sur des composants Virtex-E. Ceci est dû au fait que les Virtex-II contiennent des multiplieurs pré-implémentés dont l'utilisation présente des avantages comparés à une implémentation basée sur des additionneurs, tandis que les Virtex-E ne contiennent pas des multiplieurs cablés et avec ce type de composant la deuxième méthode est plus avantageuse.

En tenant compte des résultats de comparaison dans [92] [93] et du fait que notre multiplieur modulo 2^n+1 sera réalisé en exploitant les ressoures disponibles dans le multiplieur complexe, nous avons choisi d'adopter la première méthode basée sur le multiplieur $(n+1)*(n+1)$-bit. Cependant, les multiplieurs réalisés dans [92] [93] selon cette méthode traitent les nombres dans l'ensemble $\mathbb{Z}^*_{2^n+1} = \{a \mid 1 \leq a \leq 2^n\}$. Pour notre application, le multiplieur sera sollicité pour les calculs de la FNT et doit donc traiter tous les nombres de l'ensemble $\mathbb{Z}_{2^n+1} = \{0,\ 1,\ ...,\ 2^n\}$. Pour cela, nous avons modifié l'architecture présentée dans [93] et nous avons proposé l'architecture illustrée sur la figure 7 qui, implémentée dans le multiplieur complexe conduit à un multiplieur reconfigurable dont l'architecture est représentée sur la figure 8.

Ce multiplieur est capable d'opérer dans $\mathbb{C}$ ainsi que dans $CG(F_t)$. Le passage d'un mode à un autre est controlé par le signal *DM*. Les valeurs 1, 0 de *DM* indiquent repectivement l'exécution des multiplications complexes et modulaires. Sur la figure 8, n_c représente le nombre de bits utilisés pour les nombres complexes dont les $2^t + 1$ bits de

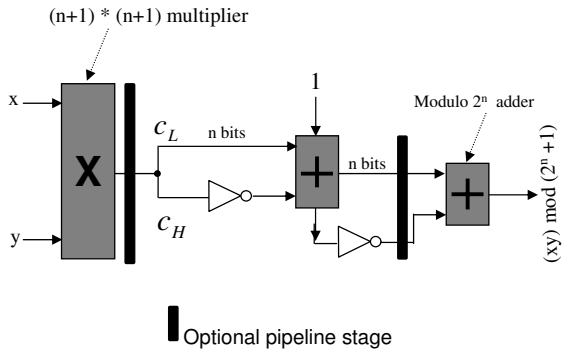

Figure 7: Le multiplieur modulo $(2^n + 1)$ proposé

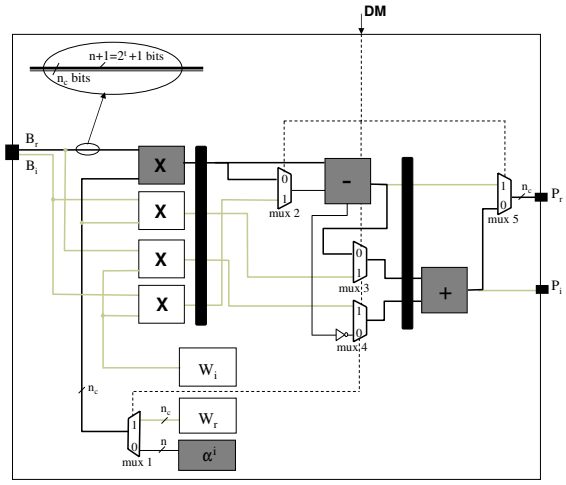

Figure 8: Le multiplieur reconfigurable proposé

poids faible seront utilisés pour les nombres dans $CG(F_t)$. Le principe de fonctionnement de ce multiplieur reconfigurable est exprimé par l'équation suivante:

$$\begin{aligned} P_r &= \begin{cases} (B_r\ W_r - B_i\ W_i) & \text{si DM=1} \\ (B_r \odot \alpha^i), \quad i = \{0, 1, ..., \frac{F_t - 1}{2} - 1\} & \text{si DM=0} \end{cases} \\ P_i &= (B_r\ W_i + B_i\ W_r). \end{aligned}$$

3.5 Architecture du papillon reconfigurable

Les trois opérateurs arithmétiques reconfigurables réalisés sont connectés entre eux selon le modèle papillon pour former un papillon reconfigurable dont l'architecture est représentée sur la figure 9. Le signal DM contrôle maintenant l'ensemble des opérateurs pour déterminer le mode de fonctionnement du papillon. $DM = 1$ indique l'exécution des opérations dans $\mathbb{C}$ et le multiplexeur au niveau des ROM contenant les coefficients de Fourier et les

puissances de α sélectionne le premier bloc de mémoire (ROM contenant les coefficients de Fourier complexes). Dans ce mode de fonctionnement, toutes les ressources arithmétiques du papillon sont utilisées. Quand DM passe à zéro, les opérations modulo F_t seront exécutées et le multiplexeur sélectionne le deuxième bloc de mémoire ROM contenant les puissances de la racine primitive α. Parmi les quatre multiplieurs, seul le premier est utilisé et les autres sont inactifs. Les sorties du papillon actives dans ce mode de fonctionnement sont P_r^1 et P_r^2. La longueur des mots binaires traités est égale à $2^t + 1$ déterminé par le corps $CG(F_t)$ dans lequel les opérations sont définies.

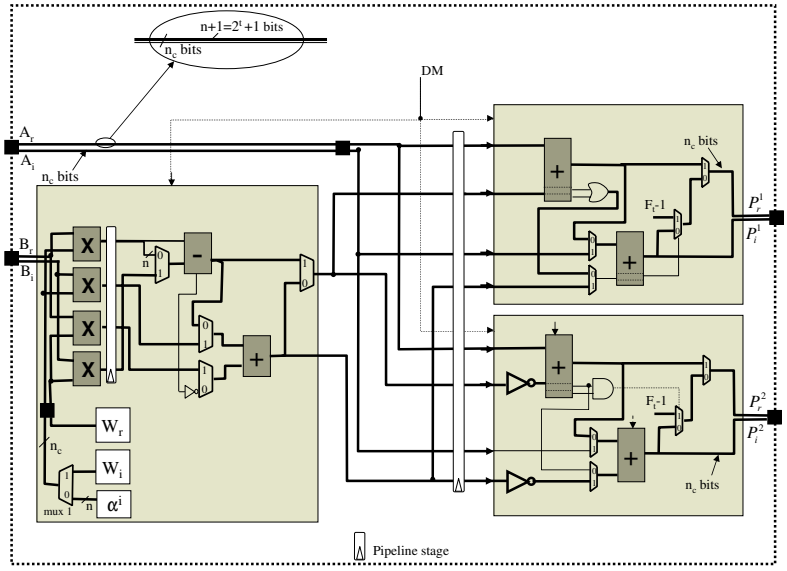

Figure 9: L'architecture du papillon reconfigurable

3.6 Architecture de l'opérateur DMFFT

Dans ce paragraphe nous allons présenter l'architecture de l'opérateur reconfigurable dit DMFFT (Dual Mode FFT). L'élement de base ou bien le coeur du DMFFT est le papillon reconfigurable présenté dans le paragraphe précédent.

Concernant l'implémentation de la FFT complexe, mis à part les besoins en ressources arithmétiques, deux aspects principaux doivent être pris en compte. Le premier est la dynamique des nombres où deux représentations peuvent être utilisées : représentation en virgule fixe ou représentation en virgule flottante. Cet aspect a un effet direct sur la précision, les erreurs de quantification et la complexité matérielle. Une grande dynamique permet d'obtenir une bonne précision mais au prix d'une grande complexité matérielle. Un compromis précision-complexité doit donc être choisi. Dans ce travail, nous avons considéré la représentation en virgule fixe et nous avons étudié la précision de calcul des transformées en fonction du nombre de bits avec lesquels les mots binaires sont représentés.

Le deuxième aspect est le besoin de blocs de mémoires où deux types sont utilisés : les mémoires RAM pour sauvegarder les calculs intermédiaires et les mémoires ROM pour sauvegarder les tables des racines de Fourier. La taille des mémoires exigée est directement liée à la représentation utilisée et à la stratégie d'implémentation de la FFT.

En général, il y a plusieurs façons d'implémenter une FFT complexe. Parmi ces différentes façons, il y a deux méthodes qui peuvent être vues comme deux extrêmes. La première consiste à réaliser la transformée avec un seul papillon dont sa fonctionalité est exécutée $N/2\ logN$ fois, avec N la longueur de la transformée. Un bloc de contrôle et des blocs de mémoires (RAM et ROM) sont nécessaires. Cette méthode permet d'obtenir un circuit très simple mais un temps de calcul très grand ($O(N/2\ logN)$).

La deuxième méthode consiste à implémenter la structure complète de la FFT, c.à.d $logN$ étages et $N/2$ papillons dans chaque étage. Avec cette méthode, il n'y a pas besoin d'utiliser des mémoires RAM puisque les données sont traitées en parallèle. Le circuit résultant permet d'obtenir un temps de calcul réduit ($log\ N$) mais avec une complexité très élevée. Cette méthode est très coûteuse en termes de ressources matérielles. Pratiquement une FFT-16 occupe 65 % d'un STRATIX-II.

Entre ces deux extrêmes, il y a une méthode qui peut constituer un bon compromis entre un temps d'exécution moyen et une complexité matérielle acceptable. Cette méthode consiste à implémenter $logN$ étages et chaque étage est constitué d'un papillon, d'une unité de contrôle et des blocs mémoires (RAM et ROM).

Basée sur cette dernière méthode, nous avons réalisé une architecture composée de $log\ N$ étages et d'une unité de contrôle globale GCU (Global Control Unit). Cette architecture est représentée sur la figure 10. Le GCU permet d'ajuster les paramètres suivants:

1. le mode de fonctionnement FFT/FNT
2. la longueur de la transformée N
3. la taille des mots binaires dans les deux domaines $\mathbb{C}$ et $CG(F_t)$.

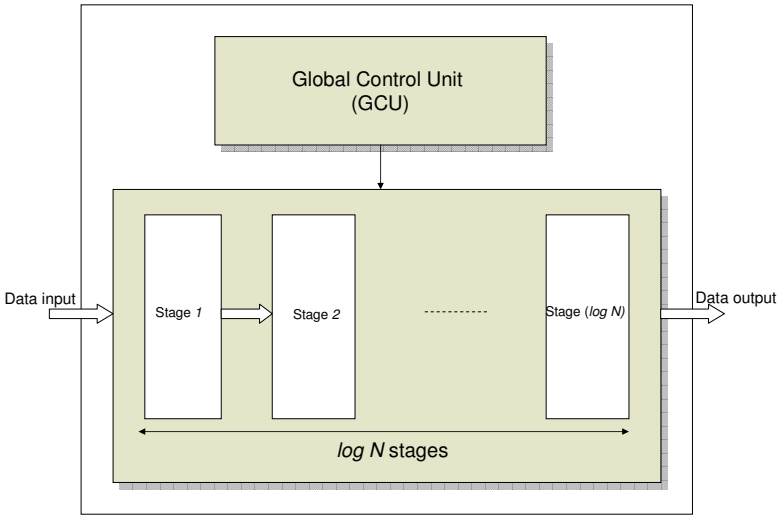

Figure 10: L'architecture de l'opérateur DMFFT

L'architecture interne d'un étage est représentée sur la figure 11. Un étage est constitué d'un papillon reconfigurable nommé RPE (Reconfigurable Processing Element), d'une unité de contrôle de l'étage SCU (Stage Control Unit), des blocs de mémoires RAM et

ROM avec leurs unités de génération de mémoires AGU (Address Generating Unit). Le principe de fonctionnement de chacune de ces unités est décrit en détail dans le chapitre 3. Nous notons que le type de mémoire utilisée joue un rôle important sur le débit global de l'opérateur DMFFT. En effet, les composants Stratix-II disposent de trois types de mémoire :

- RAM-1-Port: un port d'écriture et un port de lecture
- RAM-2-port: deux modes de fonctionnement possibles. Un simple port en écriture et un autre en lecture, ou bien deux ports en écriture et deux ports en lecture
- RAM-3-Port: un port d'écriture et deux ports de lecture.

Selon l'étude de routage de données développée dans le chapitre 3, l'utilisation du premier type de RAM permet à l'opérateur de consommer un symbole par cycle et de fournir en sortie un symbole par cycle. Avec l'utilisation du 3ème type de RAM, le DMFFT peut consommer et produire deux symboles par cycle.

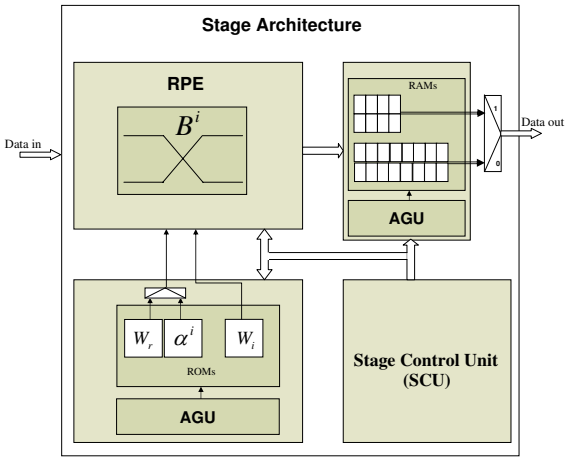

Figure 11: L'architecture de l'étage de l'opérateur DMFFT

3.7 Implémentation sur FPGA

Dans l'introduction, nous avons décrit deux approches possibles pour concevoir un système multi-standards : l'approche Velcro et l'approche reconfigurable. Dans ce paragraphe, nous allons comparer ces deux approches en implémentant les différents circuits (RPE, DMFFT) et en évaluant le paramètre $\eta = \frac{1}{TC} * 10^6$, avec C le nombre des ALUT consommés et T le temps d'exécution exprimé en ns. Ce facteur η introduit dans [97] représente le rapport performance-coût permettant d'évaluer en un seul paramètre la contribution de C et T. Dans cette étude, toutes les expérimentations ont été réalisées sur des composants FPGA STRATIX-II (ALTERA).

3.7.1 Implémentation du RPE

Comme nous l'avons déjà mentionné, l'implémentation de la FFT complexe considérée dans cette étude traite les nombres avec une représentation en virgule fixe. Cette représentation constitue une approximation des nombres traités. Ensuite tout au long du calcul de la transformée, des quantifications sont opérées pour ajuster le nombre de bits à traiter à la capacité des mémoires ou des opérateurs arithmétiques. Nous pouvons donc distinguer quatre sources de bruit :

1. les erreurs dues aux données quantifiées à traiter à l'entrée du circuit.
2. les erreurs dues aux valeurs quantifiées des coefficients $W = e^{-j2\pi/N}$.
3. après une multiplication de deux nombres de n bits, le résultat est sur $2n$ bits et subit une limitation à n bits avec arrondi ou troncature.
4. dans le calcul de la FFT complexe (radix-2) de longueur N, nous avons $log\ N$ étages de calcul et chaque étage comprend $N/2$ papillons. Il a été montré [100] que le module maximal des nombres n'augmente que d'un facteur inférieur à 2 à chaque étage de calcul. Donc, nous pouvons éviter un débordement en incorporant un facteur 1/2 (recadrage) dans le papillon. Ceci est équivalent à un décalage du nombre d'un bit vers la droite.

L'emplacement de la quantification (troncature et recadrage) joue un rôle important sur la précision des calculs et aussi sur la taille des opérateurs arithmétiques. L'analyse de l'erreur de quantification a été étudiée dans [101]. L'auteur a montré que l'emplacement de la quantification selon l'architecture de la figure 12 offre le meilleur compromis précision-complexité. Nous avons adopté ce modèle de quantification dans l'implémentation du RPE. Bien entendu, cette quantification n'intervient que dans le calcul de la FFT complexe et n'affecte pas le calcul de la FNT. Cependant, puisque nous considérons un opérateur FFT/FNT, lors du choix de nombre de bits utilisés dans le calcul de la FFT nous devons prendre en compte la dimension du corps $CG(F_t)$ et les longueurs de la FNT qui seront exécutées. En général, l'implémentation en virgule fixe de la FFT est réalisée avec un nombre de bits entre 12 et 16 bits et il a été montré qu'un nombre de bits $n_c = 13$ est un bon compromis entre la précision et la complexité des opérateurs arithmétiques [101]. Cela permet de traiter des symboles dans $CG(257)$ et ensuite l'implémentation des FNT avec une longueur allant de 16 jusqu'au 256.

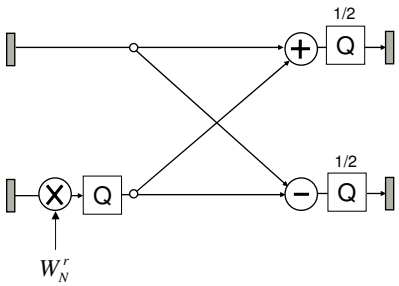

Figure 12: L'emplacement de la quantification dans le papillon

Le tableau 2 représente les résultats d'implémentation du RPE et du papillon Velcro pour différentes tailles de mots. n_c représente la taille des mots complexes et n la taille des symboles dans $CG(F_t)$. Les mesures illustrées dans le tableau 2 montrent que le RPE permet d'économiser environ 18% d'ALUT et offre un gain de rapport performance-coût de 20 % par rapport au papillon de type Velcro.

Table 2: Comparaison entre le papillon reconfigurable (RPE) et le papillon Velcro implémentés sur STRATIX II, EP2S15F484C3.

circuit	n_c=9 n=9	n_c=12 n=9	n_c=17 n=17
Papillon Velcro	403 ALUTs 4.20 ns	629 ALUTs 5.07 ns	1062 ALUTs 5.60 ns
Papillon reconfigurable	326 ALUTs 4.3 ns	514 ALUTs 5.18 ns	875 ALUTs 5.64 ns
Gain en ALUTs	19.1 %	18.2 %	17.6 %
$\eta = \frac{1}{TC} * 10^6$	$\eta_V = 590$ $\eta_R = 713$	$\eta_V = 313$ $\eta_R = 375$	$\eta_V = 168$ $\eta_R = 202$
Gain du rapport performance-coût	**20.8**%	**19.8**%	**20.2**%

3.7.2 Implémentation du DMFFT

Nous présentons dans ce paragraphe les résultats d'implémentation de l'opérateur DMFFT et de l'opérateur Velcro FFT/FNT. L'architecture du DMFFT est implémentée d'une façon générique permettant de tester plusieurs longueurs de transformées et avec différentes tailles des mots. De la même façon, chacun des opérateurs FFT et FNT constituant l'opérateur Velcro est implémenté.

Dans cette implémentation, les données d'entrée et les coefficients $W = e^{-j2\pi/N}$ de la DMFFT (dans le mode complexe) et de la FFT sont représentées avec le même nombre de bits n. Les blocs des mémoires RAM utilisés dans cette implémentation sont des RAM-1-port.

Le tableau 3 récapitule les résultats d'implémentation en termes d'ALUT, de temps de calcul, d'économie de mémoire et de gain performance-coût pour une longueur de transformée $N = 64$ et pour différents nombres de bits. Selon les résultats donnés dans ce tableau, le DMFFT permet d'obtenir, par rapport à l'opérateur Veclro une économie de mémoire allant de 21.9% à 33%, un gain en nombre d'ALUT allant de 9% à 26 % et un gain performance-coût allant de 9.7 % à 37 %. Ces gains varient avec le nombre de bits n_c utilisés. Cela peut s'expliquer par le fait que lorsque n_c augmente, n étant constant, la complexité du DMFFT augmente d'une façon globale au profit de la FFT complexe tandis que $n = 9$ bits est suffisant pour réaliser des FNT de longueur (maximale) 64.

Le tableau 4 récapitule les mêmes mesures pour une longueur de transformée $N = 256$. L'évolution du gain en ALUTs et du gain performance-coût est la même qu'avec une longueur de transformée $N = 64$ mais avec des valeurs moins élevées. Cette diminution de la valeur des gains peut s'expliquer par le fait que la complexité de l'opérateur DMFFT

Table 3: Comparaison entre le DMFFT-64 et l'opérateur Velcro FFT/FNT-64 implémentés sur Stratix II, EP2S15F484C3

n_c	9	10	11	12	13	14	15	16
Opérateur Velcro	4205 ALUTs 4.86 ns	4768 ALUTs 5.27 ns	5156 ALUTs 5.08 ns	5831 ALUTs 5.46 ns	6064 ALUTs 5.74 ns	6844 ALUTs 5.81 ns	7302 ALUTs 5.79 ns	8143 ALUTs 6.62 ns
DMFFT	3109 ALUTs 4.78 ns	3744 ALUTs 4.97 ns	4112 ALUTs 5.0 ns	4857 ALUTs 5.45 ns	5182 ALUTs 5.76 ns	5975 ALUTs 5.85 ns	6469 ALUTs 5.86 ns	7387 ALUTs 6.65 ns
Economie de mémoire	33 %	31 %	29 %	27.2 %	25.7 %	24.3 %	23 %	21.9 %
Gain en ALUTs	26 %	21.4 %	20.2 %	16.7 %	14.5 %	12.7 %	11.4 %	9.2 %
gain du facteur η	37.4 %	35 %	27.5 %	20 %	16.7 %	13.9 %	11.5 %	9.7 %

est dominée par celle de la FFT. Cette dernière augmente rapidement avec la taille de la transformée et ce qui fait chuter le gain en nombre d'ALUTs.

Table 4: Comparaison entre le DMFFT-256 et l'opérateur Velcro FFT/FNT-256 sur Stratix II, EP2S15F484C3

n_c	9	10	11	12	13	14	15	16
Opérateur Velcro	5327 ALUTs 5.02 ns	6079 ALUTs 4.95 ns	6566 ALUTs 5.13 ns	7518 ALUTs 5.45 ns	7885 ALUTs 5.58 ns	8966 ALUTs 5.84 ns	9513 ALUTs 6.1 ns	10770 ALUTs 6.55 ns
DMFFT	4466 ALUTs 4.86 ns	5365 ALUTs 4.9 ns	5911 ALUTs 5.0 ns	6819 ALUTs 5.5 ns	7336 ALUTs 5.63 ns	8546 ALUTs 5.9 ns	9124 ALUTs 6.11 ns	10389 ALUTs 6.58 ns
Economie de mémoire	33 %	31 %	29 %	27 %	25.5 %	24 %	22.8 %	21.9 %
Gain en ALUTs	16.2 %	11.7 %	9.97 %	9.29 %	7 %	4.68 %	4 %	3.5 %
Gain du facteur η	24 %	15.1 %	12.1 %	9.4 %	6.6 %	4.2 %	4.06 %	3.54 %

Pour sélectionner le nombre de bits offrant le meilleur compromis entre la précision de calcul (de la FFT) et la complexité matérielle, nous avons calculé la FFT complexe avec Matlab dont le calcul se fait en virgule flottante afin d'évaluer le rapport SQNR (Signal to-Quantization-Noise Ratio) défini par

$$SQNR = 10\ log(\frac{E[|S(k)^2|]}{E[|N(k)^2|]}),$$

avec $E[|S(k)^2|]$ et $E[|N(k)^2|]$ représentant respectivement les moyennes quadratiques des FFT obtenues avec Matlab $S(k)$ et de l'erreur $N(k) = S(k) - S_f(k)$.

La conclusion de cette étude est qu'un nombre de bits égal à 13 constitue un bon compromis précision-complexité.

4 DMFFT et FFT dans $CG(2^m)$: vers un opérateur FFT tri mode

L'objectif de cette partie est de proposer des solutions pour réaliser un opérateur FFT tri mode, opérant sur trois domaines différents, à savoir $\mathbb{C}$, $CG(F_t)$ et $CG(2^m)$. Ceci est motivé par la nécessité de prendre en compte les codes RS définis sur $CG(2^m)$ mais aussi en raison de la simplicité des opérations arithmétiques dans $CG(2^m)$.

Le premier opérateur proposé est le TMVFFT pour Triple Mode Velcro FFT. Une évolution de cet opérateur est alors proposée selon deux scénari, décrits dans la suite.

4.1 L'architecture de l'opérateur TMVFFT

L'architecture proposée est constituée de l'opérateur DMFFT et d'une FFT définie sur $CG(2^m)$ (notée par la suite FFT-GF2). Cette architecture est représentée sur la figure 13. Cette structure intuitive constitue une première version d'un opérateur tri-modes. Nous allons maintenant proposer deux scénari d'évolution : le scénario 1 permet d'optimiser la réutilisation de chacun des deux opérateurs constituant le TMVFFT ; le scénario 2 vise à combiner ces deux opérateurs (DMFFT et FFT-GF2) pour évoluer vers un opérateur tri mode plus optimal et reconfigurable.

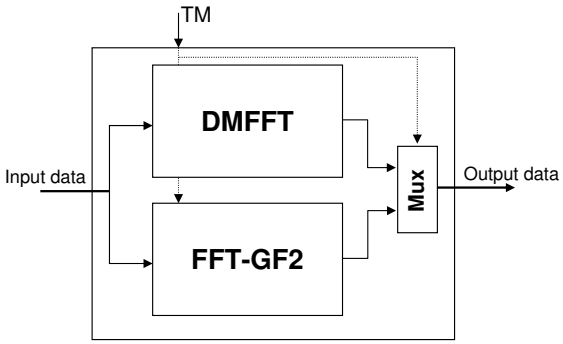

Figure 13: L'architecture du TMVFFT

4.1.1 Scénario 1 : évolution de l'architecture TMVFFT

En général, un opérateur commun est supposé remplacer deux autres opérateurs à la condition qu'il soit capable de produire leurs fonctionnalités pendant la même durée d'exécution de chacun des deux opérateurs. Ceci est illustré sur la figure 15.

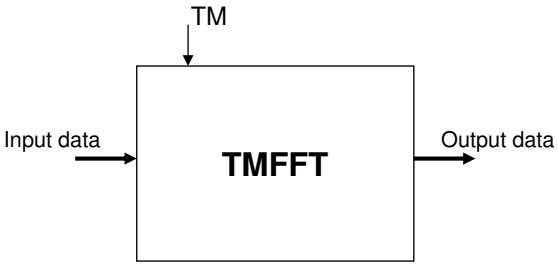

Figure 14: L'architecture du TMFFT

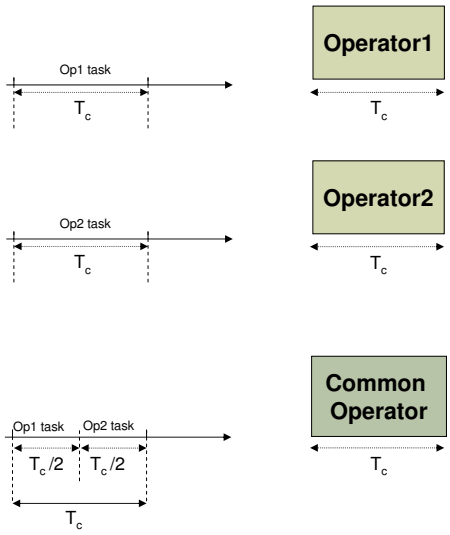

Figure 15: Diagramme des tâches d'un opérateur commun

L'optimisation du DMFFT selon le principe du scénario 1 peut être réalisée en augmentant le débit de l'opérateur. Comme déjà évoqué, ceci peut être atteint en utilisant des RAM-3-port permettant au DMFFT de fournir deux symboles par cycle. De cette façon, le nombre de cycles nécessaires pour réaliser la transformée (FFT/FNT) est $\frac{N}{2}$ ce qui permet à cet opérateur opérant dans le mode FNT d'être capable de remplacer les deux circuits nécessaires pour exécuter le calcul des syndromes et l'algorithme de Chien.

Dans la logique de ce scénario 1, nous avons proposé une implémentation du FFT-GF2 permettant de réduire son temps d'exécution d'un facteur deux à huit, avec une complexité raisonnable.

Cette implémentation est basée sur une méthode de calcul de la transformée de Fourier proposée dans [105]. Nous avons reformulé cette méthode d'une façon à être adaptée à l'implémentation matérielle.

La méthode est basée sur la factorisation de la matrice de la tranformée en produit des matrices circulantes et matrices circulantes diagonales. Cette factorisation est possible

grâce à la décomposition cyclotomique des polynômes définis dans $CG(2^m)$. Comme les auteurs le montrent, cette méthode comparée à d'autres algorithmes existants dans la litérature [102] [106] [107] est plus efficace puisqu'elle nécessite un nombre plus faible d'opérations (multiplications et additions).

Nous avons étudié cette méthode et nous avons constaté que la décomposition cyclotomique polynomiale permettant la réduction des nombres des opérations peut être exploitée pour augmenter le débit du FFT-GF2 et réduire le temps de calcul.

Le principe de cette méthode est détaillé dans le chapitre 4. Nous avons présenté la décomposition cyclotomique des éléments du $CG(2^m)$ pour m=3, 4, 5, 6, 7 et 8 et en se basent sur cette décomposition nous avons proposé une architectue matérielle de l'opérateur FFT-GF2. Cette architecture permet de traiter les symboles par groupe, ce qui permet de réduire le temps de calcul. L'architecture que nous avons proposée est constituée de deux unités : une unité principale et une unité secondaire (figure 16). L'unité principale, dédiée à traiter les groupes principaux du $CG(2^m)$ (par exemple $\{f_1, f_2, f_4, f_8\}$,...etc. dans $CG(2^4)$), se compose de quatre étages dont le quatrième a une structure régulière et paramétrable. Ainsi, le nombre de cellules implémentées dans cet étage, défini suivant la décomposition cyclotomique du $CG(2^m)$ concernée, peut être modifié afin d'offrir une flexibilité dans le temps de calcul et le débit. L'unité secondaire est dédiée à traiter les groupes secondaires ($\{f_5, f_{10}\}$) du $CG(2^4)$.

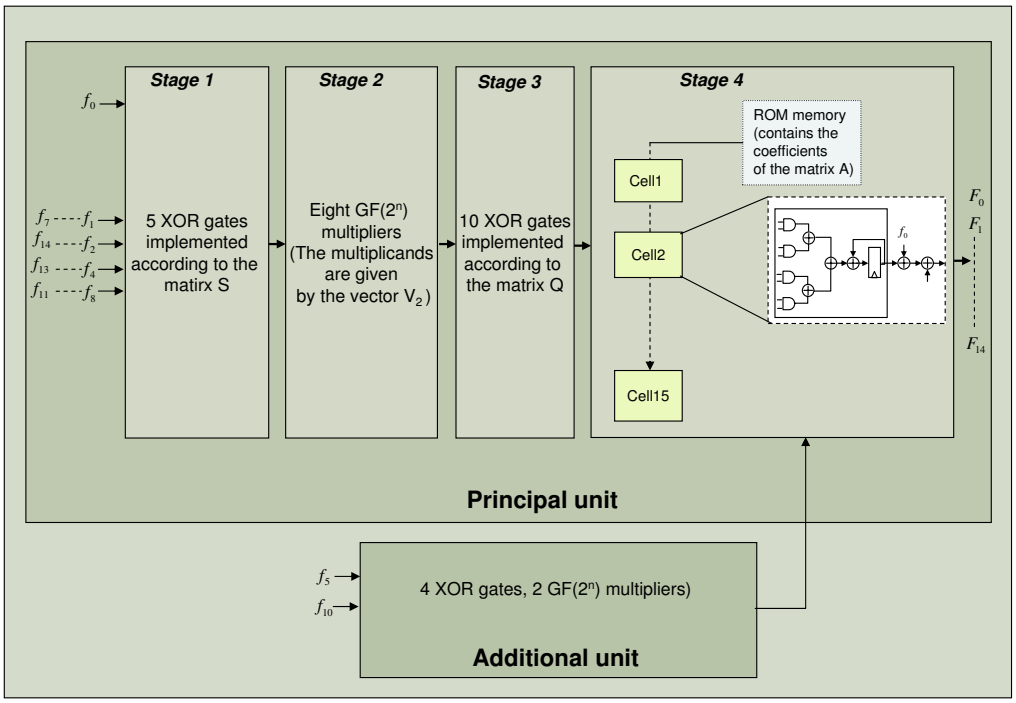

Figure 16: L'architecture de la FFT cyclotomique

Nous avons comparé cette architecture à celle proposée dans [102] en termes de ressources arithmétiques et temps de calcul. Les deux architectures exigent le même nombre de multiplieurs définis dans $CG(2^m)$. En terme d'additionneurs ou portes XOR, notre architecture

Table 5: Tableau des temps de calcul

CG	**Temps de calcul** (architecture de [102])	nb. de cellules N_c	**Temps de calcul** (architecture proposée)
$CG(2^4)$	16 T_c^*	8	6 T_c
		15	3 T_c
$CG(2^5)$	32 T_c	16	12 T_c
		31	6 T_c
$CG(2^6)$	64 T_c	32	18 T_c
		63	9 T_c
$CG(2^7)$	128 T_c	64	36 T_c
		127	18 T_c
$CG(2^8)$	256 T_c	64	120 T_c
		128	60 T_c
		255	30 T_c

* T_c : temps de calcul d'une multiplication dans $CG(2^m)$

nécessite un nombre plus grand de portes XOR. Des blocs ROM sont aussi nécessaires pour sauvegarder les coefficients matriciels. Quant à l'architecture proposée dans [102], elle nécessite des blocs de contrôle. Le principal avantage que présente l'architecture que nous proposons est qu'elle permet de réduire le temps de calcul d'un facteur allant de deux à huit (selon le $CG(2^m)$) produisant ainsi un débit très élevé tandis que l'autre architecture a un temps de calcul plus grand et opère à un débit fixe et plus faible.

Le tableau 5 montre les différents temps de calcul pour lesquels l'architecture du FFT-GF2 proposée peut opérer et ceux de l'architecture proposée dans [102]. Pour chaque $CG(2^m)$, il y a différents temps de calcul possibles pour notre architecture. Ceci est directement lié à la struture du quatrième étage de l'unité principale, i.e. au nombre de cellules N_c implémentées dans cet étage. Par exemple, dans $CG(2^4)$, notre architecture peut fonctionner à des vitesses deux et cinq fois plus rapide.

Nous avons implémentés l'architecture proposée pour une longueur de transformée $N = 15$ sur des composants STRATIX II pour évaluer ses performances et sa complexité en terme de nombre d'ALUT. Le tableau 6 représente les résultats de cette implémentation. Pour un temps de calcul égal à $3T_c$, l'architecture consomme 343 ALUTs. Le nombre d'ALUTs peut être réduit à 203 en diminuant le nombre de cellules implémentées dans le quatrième étage à 8. Ceci est au prix d'un temps de calcul plus grand qui sera $6T_c$.

4.1.2 Scénario 2 : vers un opérateur tri-mode reconfigurable TMFFT

Dans cette partie nous proposons quelques solutions permettant de combiner les deux opérateurs DMFFT et FFT-GF2 d'une façon à obtenir un seul opérateur reconfigurable TMFFT permettant de réaliser les trois transformées de trois opérateurs : FFT-$\mathbb{C}$, FNT et FFT-GF2. Le scénario 2 suppose la réalisation du TMFFT en deux étapes :

Table 6: Les résultats d'implémentations d'un FFT-15 (dans $CG(2^4)$) sur un composant STRATIX II, EP2S15F484C3

temps de calcul	N_c	Complexité	T_c
$t' = 3\ T_c$	15	343 ALUTs	2 ns
$t' = 6\ T_c$	8	203 ALUTs	2 ns

- réalisation des opérations arithmétiques dans $CG(2^m)$ à l'aide des opérateurs implémentés dans le DMFFT
- incorporation de la structure physique du FFT-GF2 dans l'opérateur DMFFT.

Nous considérons dans ce paragraphe la réalisation de la première étape. Une addition dans $GF(2^m)$ peut être réalisée à l'aide des simples portes XOR. Quant à la multiplication, plusieurs algorithmes ont été proposés pour réaliser des multiplieurs opérant dans $CG(2^m)$ [114] [115] [116] [117]. Dans cette étude, nous considérons la réalisation d'un multiplieur dans $CG(2^m)$ en utilisant la structure classique d'un multiplieur binaire. Dans cet optique, deux travaux principaux ont été développés proposant la réalisation des multiplieurs combinés sur DSP [118] et sur ASIC [119] capables de réaliser des multiplications binaires classiques ainsi que dans $CG(2^m)$. Les multiplieurs proposés dans [118] [119] montrent une réalisation d'un multiplieur opérant dans $CG(2^8)$ sur une structure d'un multiplieur binaire de taille 16×16 bits. En se basant sur le même principe, nous avons proposé une architecture d'un multiplieur permettant de réaliser une multiplication dans $CG(2^m)$, pour m=6, 4 et 8 à l'aide d'un multiplieur binaire normale. Un multiplieur binaire de taille 8×8 bits représente une structure suffisante pour réaliser une multiplication dans $CG(2^8)$, mais comme ce multiplieur est destiné à réaliser des calculs dans le domaine complexe, sa taille sera fixée selon la précision souhaitée de la transformée de Fourier complexe.

L'architecture du multiplieur combiné est illustrée sur la figure 17 pour un multiplieur de base 8×8 bits. Ce multiplieur est composé de trois étapes principales : la génération des produits partiels, la réduction de ces produits selon la méthode de Walace [112] et ensuite l'addition de ces produits dans le cas d'une multiplication binaire normale et leur réduction polynomiale dans le cas de multiplication dans $CG(2^m)$.

Nous considérons maintenant le principe de fonctionnement de ce multiplieur. En effet, dans le mode binaire normal, la production des produits partiels peut être réalisée à l'aide des portes logiques AND. Ensuite, la réduction de ces produits partiels peut être réalisée suivant le schéma arborescent de Wallace [112] construit à l'aide des éléments de base "Full Adder" dénoté par $W3$ et "Half Adder" dénoté par $W2$.

Cet arbre de Wallace permet d'effectuer la somme des produits partiels d'une façon efficace. Cette somme tient évidement compte de la propagation de la retenue tout au long de l'arbre. Dans le $CG(2^m)$, cet arbre doit se comporter différement. C'est à dire, puisqu'il s'agit d'une addition modulo 2, la propagation de la retenue doit être évitée. Pour cela il faut reconfigurer les interconncetions des cellules $W3$ et $W2$ d'une façon à

éviter la propagation de retenue au niveau de chaque cellule. Dans une implémentation FPGA, cela est possible en utilisant des LUT permettant de mettre à zéro l'entrée d'une cellule connectée à une sortie représentant la retenue d'une autre cellule.

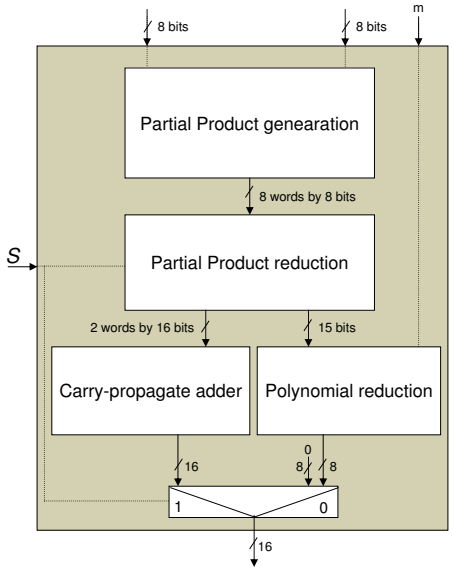

Figure 17: Diagramme bloc du multiplieur combiné

Dans la troisième étape et à la sortie de l'arbre, un additionneur final termine la multiplication binaire normale. Dans le cas de la multiplication dans $CG(2^m)$, une réduction polynomiale est nécessaire pour réduire un produit de $2m-2$ bits à un produit sur m bit. Cette réduction est faite modulo un polynôme dit primitif caractérisant le corps de Galois $CG(2^m)$.

La figure 18 représente la structure de l'unité de réduction polynomiale permettant d'opérer dans $CG(2^m)$ pour m=6, 7 et 8. Bien entendu, il est possible de l'étendre pour d'autres valeurs de m, mais nous avons choisi ces quatres valeurs qui représentent l'ordre des corps de Galois dans lequels les codes de Reed-Solomon les plus utilisés sont construits.

Le passage d'un mode de fonctionnement à un autre du multiplieur de la figure 17 est assuré par un signal de contrôle S permettant de reconfigurer les interconnections de l'arbre de Wallace et de sélectionner la sortie souhaitée. Un paramètre m est nécessaire pour préciser l'ordre du corps de Galois et sélectionner ensuite les représentations binaires des coefficients α^i correspondants sauvegardés dans des blocs ROM.

Le multiplieur combiné de la figure 17 peut être facilement intégré dans le multiplieur reconfigurable implémenté dans l'opérateur DMFFT. Cela permet d'obtenir un multiplieur tri-mode capable d'opérer dans trois domaines différents: $\mathbb{C}$, $CG(F_t)$ et $CG(2^m)$.

De cette façon, nous avons toutes les ressources arithmétiques nécessaires pour réaliser les opérations requises par l'opérateur TMFFT envisagé. La première étape du scénario 2 est donc accomplie. Concernant la deuxième étape, des études supplémentaires sont nécessaires pour sa réalisation.

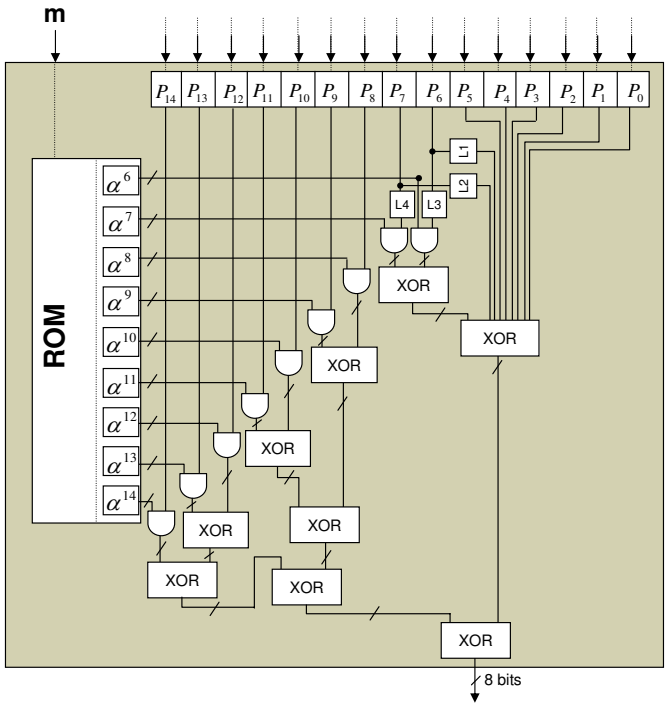

Figure 18: Réduction polynomiale parallèle pour $m = 6$, 7 and 8.

5 Conclusions

Ce travail se situe dans le domaine de la radio logicielle et plus précisément dans la recherche de structures communes pouvant aboutir à la réalisation de terminaux reconfigurables. Cette approche appelée paramétrisation peut être abordée de façon théorique ou pragmatique. Seule la déclinaison pragmatique a été étudiée dans ce travail. Il s'agit d'identifier des opérateurs capables de réaliser plusieurs fonctionnalités, à condition de rendre les opérateurs reconfigurables.

L'opérateur de FFT a été étudié afin qu'il soit capable d'effectuer des transformées de Fourier aussi bien dans le corps infini des nombres complexes que dans les corps finis tels que les corps de Galois. Un tel opérateur a ainsi la possibilité de réaliser les opérations classiques où intervient la transformée de Fourier (filtrage, (dé)modulation OFDM, égalisation, etc) mais aussi des opérations de décodage de canal des codes de Reed-Solomon. Pour des raisons structurelles, seuls les codes de Reed-Solomon définis sur $CG(F_t)$ (F_t : nombre de Fermat) ont la possibilité d'être traîtés avec une architecture semblable à celle de la FFT classique. C'est la raison pour laquelle ils ont été étudiés dans ce travail. Leurs performances sont quasiment identiques à celles des codes RS définis sur $CG(2^m)$.

Ces études ont abouti à la définition, la réalisation, la validation et l'implémentation d'un opérateur dual (DMFFT) capable d'effectuer des transformées dans $\mathbb{C}$ mais aussi dans $CG(F_t)$. Le passage d'une transformée à l'autre se fait alors par reconfiguration de l'architecture. Il a été montré que cette approche est bien plus optimale qu'une approche

de type Velcro.

Enfin, nous avons proposé des solutions afin de converger vers la réalisation d'un opérateur tri mode capable d'effectuer des transformées dans $\mathbb{C}$, dans $CG(F_t)$ mais aussi dans $CG(2^m)$, permettant ainsi de prendre en compte les codes RS définis sur $CG(2^m)$, codes qui sont aujourd'hui très répendus dans les standards de communication.

Les perspectives de ce travail sont nombreuses. La principale sera d'aller plus loin dans la réalisation de l' opérateur tri-mode. Une première solution a été proposée dans ce travail sur la base d'un multiplieur tri-mode capable d'effectuer des multiplications dans $\mathbb{C}$, dans $CG(F_t)$ mais aussi dans $CG(2^m)$. C'est une première étape. Il faudra aller plus loin pour réaliser le papillon en entier puis le TMFFT pour Tri Mode FFT.

Une autre étude sous jacente à la réalisation d'un opérateur reconfigurable est liée à l'ordonnancement des opérations et au partage des ressources. En effet, un tel opérateur commun est par nature partagé entre plusieurs traitements. Il est alors nécessaire de gérer le partage de cet opérateur et la gestion des données qui s'en suit. L'étude d'une reconfiguration dynamique de cet opérateur, permettra de rendre la gestion de l'opérateur plus flexible et le gain apporté par l'approche OC plus rentable.
Comme cette thèse l'a montré, il est tout à fait possible de réaliser des opérations de décodage de canal à l'aide de FFT, ce qui permet de mutualiser l'utilisation de cet opérateur. Dans cet optique, l'étude et l'implémentation de l'algorithme de Gao [109] en utilisant cet opérateur, pourra constituer une exploitation efficace de l'opérateur FFT puisque, comme l'auteur le mentionne, toutes les principales opérations de décodage des codes RS peuvent être réalisées à l'aide de la FFT. Les codes étudiés dans cette thèse sont les codes de Reed-Solomon, une étude plus prospective pourra se pencher sur l'application de la FFT pour des codes LDPC non-binaires et pour des codes convolutifs.

General Introduction

1 Background and context

In recent years there has been an enormous proliferation of standards in broadcast television, radio and in mobile communications. Current examples include digital television (DVB, ISDB), digital radio (DAB), wireless LAN(Hiperlan, 802.11a, 802.11b, ..., 802.16m), 2.5/3G and future 4G mobile communications. These standards form the basis for an ever-growing number of sophisticated consumer electronic devices, each with the potential to sell in very high volumes.

In typical designs, these complex standards are implemented using dedicated architectures, which are optimized to reduce cost to the absolute minimum. Products developed using dedicated architectures are often difficult to upgrade in order to support changes to the standards or to add new features.

At the beginning of the 90's, a concept called *SoftWare Radio* (SWR) has emerged from demonstrations in military research to become a cornerstone of the third generation strategy for affordable, ubiquitous and global communications [1]. This SWR technology is a way to design a sufficiently programmable and reconfigurable architecture able to support many different transmission standards on a common platform. The reconfigurability of a SWR system can offer a range of benefits at different levels. A radio system implemented on a reconfigurable architecture can be upgraded to fix bugs or to add functionalities, and it can also support new standards as it is assumed that there is sufficient flexibility in the architecture.

The communication chains of different standards, intended to be implemented on a common platform, have some common signal processing operations such as channel coding, modulation, equalization, etc. In order to exploit to a great advantage the commonalities among these communication tasks for different standards, one need firstly to identify these commonalities and secondly find the optimal way to implement a generic hardware platform with programmable modules. In this sense, a technique called *parametrization* has been introduced. The key idea is to get an optimal sharing between hardware and software resources and find a best way to reuse some hardware and software modules without affecting the system's performances.

This thesis fits into the context of SWR and more precisely in the parametrization schemes where two approaches are considered, Common Function (CF) approach and Common operator (CO) approach. Next section describes the main scope and the main objectives of this study.

2 Scope and objectives

Since the publication of the Cooley-Tukey algorithm [32] which allows to enormously reduce the order of complexity of some crucial computational tasks like Fourier Transform, the frequency processing has began to find its use in a wide range of applications. This processing technique was rapidly credited by the tremendous increase of interest in Digital Signal Processors (DSP) started in the seventies and continued to play a key role in the widespread use of digital signal processing in a variety of applications like telecommunications.

In [21], it was shown that several communication tasks like filtering, channelization, modulation, despreading, Rake function, ..., can be realized in frequency domain. In SWR context, the authors proposed the Fast Fourier Transform (FFT) as a CO operator that could be implemented in a way to be able to match the requirements of the different tasks intended to be realized in frequency domain. The frequency processing of these functions is defined in the traditional field of complex numbers $\mathbb{C}$.

One of the most important and mandatory steps in a communication system is the channel coding. The codes used for this function are defined in a less familiar domain such as Galois Field (GF). The first objective of this thesis is to investigate the study of channel coding (particularly cyclic codes) in frequency domain and draw an analogy between this frequency processing and the classical frequency signal processing based in the domain of complex field. In light of this analogy, comes the second objective aiming to design a reconfigurable architecture of a common FFT operator able to operate over two different domains $\mathbb{C}$ and GF. For simplicity of exposition, we denote by FFT-$\mathbb{C}$ and FFT-GF2 the Fourier transforms defined over $\mathbb{C}$ and $GF(2^m)$ respectively. In this way, the designed FFT operator can be used in two different contexts: channel coding and any communication task requiring the FFT-$\mathbb{C}$ functionality. The codes considered in this study are the Reed-Solomon (RS) codes. These codes are chosen to be treated in this study for two reasons. Firstly, for their powerful error correcting capability particularly in the case of burst errors. Secondly, their non-binary mathematical structure makes their treatment adapted to be efficiently performed with the arithmetical operators available in the FFT-$\mathbb{C}$ architecture.

3 Contents and major achievements

This thesis is composed of four chapters. In chapter 1, we describe the SWR technology with its important area of parametrization technique. This parametrization technique with its two approaches CF and CO is described. We present some examples of CF and CO and focus our attention on the CO approach where we consider the FFT as the operator to be designed in a reconfigurable way to be subsequently considered as a CO able to support two different types of transform computations.

Chapter 2 presents an overview of channel coding and the frequency processing of cyclic codes particularly RS codes defined over $GF(2^m)$. For these RS codes we present their frequency encoding and decoding, their principal characteristics towards their implementation using FFT-GF2 for the encoding and some principal decoding processes. However, the transform length of the FFT-GF2, being of the form $2^m - 1$, represents a strong challenge facing the design of a common structure for the FFT-$\mathbb{C}$ and FFT-GF2.

For this, we revive a specific class of RS codes defined over a specific GF, such as the GF of characteristic F_t where F_t is a Fermat number. This $GF(F_t)$ permits to define a transform called Fermat Number Transform (FNT) having a very similar structure of that of FFT-$\mathbb{C}$. This similarity allows to efficiently implement the FNT onto the FFT-$\mathbb{C}$ structure. We show that the FNT operator can be efficiently used to encode and perform some decoding processes of these codes defined over $GF(F_t)$. We also show that these specific RS codes defined over $GF(F_t)$ have almost the same performances in terms of Bit Error Rate (BER) as compared to classical RS codes defined over $GF(2^m)$. Nevertheless, according to the simulation results, we demonstrate that the frequency encoding of the RS codes defined over $GF(F_t)$ degrades the code's performances and the time domain encoding should be used to obtain good performances.

Chapter 3 deals with the design of reconfigurable FFT operator called Dual Mode FFT (DMFFT) operator. In this chapter we propose the architectures of all the reconfigurable arithmetical operators (multiplier, adder and subtracter) leading to a Reconfigurable Processing Element (RPE) representing the well known *butterfly operation*. This RPE implemented in the classical FFT-$\mathbb{C}$ structure leads to the DMFFT. The rest of this chapter focuses on the FPGA implementation of the designed DMFFT operator where we have evaluated its complexity and speed performances and have compared them to those of the Velcro FFT/FNT operator composed of two self-contained operators: FFT-$\mathbb{C}$ and FNT.

In chapter 4, we investigate some opportunities to redesign the DMFFT in such a way to be able to provide the FFT-GF2 functionality. The reason lies in the fact that the RS codes implemented in the actual standards are RS codes defined over $GF(2^m)$. This extra functionality of the DMFFT makes it a triple mode FFT operator. At first, we present the Velcro structure of this operator which we call Triple Mode Velcro FFT (TMVFFT), that is an operator consisting of two self-contained operators DMFFT and FFT-GF2. Then, we give two scenarios to maximize the reuse of the TMVFFT and evolve its Velcro structure. The first scenario aims to increase the efficiency of the reuse of each operator included in the TMVFFT. The second scenario aims to merge the two operators (DMFFT and FFT-GF2) and obtain a single and reconfigurable operator called Triple Mode FFT (TMFFT) operator able to provide three functionalities: FFT-$\mathbb{C}$, FNT over $GF(F_t)$ and FFT-GF2 over $GF(2^m)$.

Finally, we conclude this thesis with a summary of the achieved results and present some perspectives for future research.

Chapter 1

The parametrization for SoftWare Radio systems

Contents

This chapter presents the SoftWare Radio technology and its related technique "the parametrization", which is the context of interest of this thesis. It describes some SWR architectures and gives two parametrization approaches: theoretical and pragmatic approaches. Under the pragmatic approach, we introduce two sub-approaches: Common Function (CF) approach and Common Operator (CO) approach. As common operators, we give two examples: MulDiv and FFT. This latest operator, candidate to be a common and reconfigurable operator for several communications tasks and in different environments, will constitute the keystone of this work.

1.1 The SoftWare Radio technology

The new context of SWR mobile communications could provide multi-standard terminals for flexible radios. Embodiment of the commonalities among different standard modules

by realizing a generic and reconfigurable standard entails replacement of the analog modules by digital ones. A main reason for replacing analog with digital signal processing is the possibility to "softly" reconfigure the system, thereby enabling the implementation of different air interfaces on a given hardware platform. This can lead to programmable modules and finally built an open-architecture based radio system. This long term objective paves the way to numerous technical challenges such as wide band antenna, linear amplification of a multi-standards signal, wide band analog to digital conversion, link and cross layer adaptation, high frequency digital architecture, etc.

The SWR concept was first introduced in the literature around 1990 thanks to the pioneering works of J. Mitola [1] and W. Tuttlebee [2]. There is no specific definition of the SWR, but in our understanding, when we talk about SWR system (transmitter/receiver), it is understood as a system whose functions are realized, piloted and executed by software. SWR technology has generated tremendous interest in the wireless industry for the wide ranging economic and deployment benefits it offers. It can be used to implement military, commercial and civil applications. A wide range of radio applications like Bluetooth, WLAN, Global Positioning System (GPS), Wideband Code Division Multiple Access (W-CDMA), etc. can be implemented using this new technology while most of their dedicated functions can be implemented by software.

Today's SWR is challenged by the implementation of wireless communications infrastructure equipments completely in hardware which make the wireless communications industry facing the following problems:

- The continuously evolving wireless network from second generation (2G) to 2.5/3G and then further onto (fourth generation) 4G force subscribers to buy new handsets whenever a new generation of network standard is deployed. This is due to the significant difference in link-layer between each generation of network. So, legacy handsets may be incompatible with newer generation network.
- The air interface and link-layer protocols differ across various geographies. European wireless networks are predominantly Global System for Mobile communication (GSM)/Time Division Multiple Access (TDMA) based while in US the wireless networks are predominantly IS94/CDMA based. This problem has inhibited the deployment of global roaming facilities causing great inconvenience to user who travel frequently from one continent to another.

In order to resolve the hardware problem or at least find a solution, SWR was introduced to enable the implementation of radio functions in networking infrastructure equipment and user terminal as software modules running on a generic hardware platform. This significantly eases migration of network from one generation to another since the migration would involve only a software upgrade.

A SWR system would then have the following features:

- **Reconfigurability**: SWR system is dynamically reconfigurable that allows the implementation of different standards. The co-existence of multiple software modules

permits to run the required standard by just downloading the appropriate software module.

- **Ubiquitous connectivity**: Having an air interface standard as software modules helps in realizing global roaming facility. If the terminal is incompatible with the available network technology in a particular region, the terminal can be upgraded by a simple download for an appropriate software module.

1.2 Receiver architectures

The multi-band, multi-mode operation of a SWR introduces stringent requirements on the underlying system architecture. The requirement of supporting multiple frequency bands affects the design of the Radio Frequency (RF) front end and the requirement of Analog-to-Digital (ADC) and Digital-to-Analog (DAC) converters [3]. The RF front end should be adjustable or directly suitable for different frequencies and bandwidths required by the different standards that the SWR system intends to support. In the following subsections we discuss the various RF front end architectures and then we present the today's feasible SWR architecture.

1.2.1 Radio frequency front end

Architecture of a typical digital radio system can be represented by the block diagram of Fig. 1.1. The transmitter is divided into an information source, a source encoder, an encryptor, a channel encoder, a modulator, a DAC and a RF front end block. Correspondingly, the receiver consists of an RF front end, an ADC, a synchronization block, a demodulator, a detector, a channel decoder, a decryptor and a source decoder. The main functions of RF front end are down and up conversion, channel selection, interference rejection and amplification. The exact point where the conversion between digital and analog waveforms is done depends on the architecture. In conventional radio architectures, the conversion is done at the baseband, whereas in some specific SWR sub-architectures such as Software Defined Radio (SDR) that will be presented in subsection 1.2.5, the typical place for the ADC and DAC is between the stages of channel modulation, at an intermediate frequency. To be transformed into an ideal SWR architecture, the architecture of the Fig. 1.1 must employ the digital processing block (i.e. the ADC and DAC) right beside the antenna.

Although an ideal radio would have a very minimal Analog Front End (AFE), consisting of an ADC placed as close as possible to the antenna, any practical implementation still needs some analog parts of RF front end, and the design of a reconfigurable RF part remains a very complicated issue [1][3]. The receiver section is more complex than the transmitter and the ADC is the most critical part limiting the choice of the RF front end architecture [3]. The transmitter side of the RF front end takes the signal from the DAC, converts the signal to the transmission radio frequency, amplifies the signal to a desired level, limits the bandwidth of the signal by filtering in order to avoid interference and feeds the signal to the antenna [2]. The receiver side converts the signal from the antenna to a lower center frequency such that the new frequency range is compatible with the ADC, filters out noise and undesired channels and amplifies the signal to the level suitable for the ADC. The common part of every receiver architecture apart from fully digital ones is that

the antenna feeds signal through an RF Band Pass Filter (BPF) to a Low Noise Amplifier (LNA). Automatic Gain Control (AGC) keeps the signal level compatible with the ADC. A main objective during the design of an optimal RF front end is to achieve a suitable dynamic range and minimizing additive noise while minimizing the power consumption. Usually, there has to be trade-off between power consumption and dynamic range.

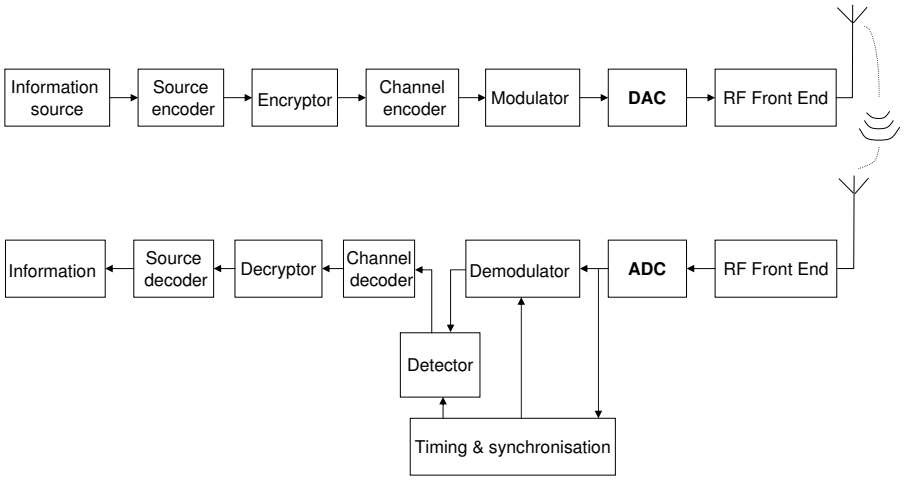

Figure 1.1: Block diagram of a digital radio system

1.2.2 The classical superheterodyne architecture

The superheterodyne architecture has become an obvious choice in receivers since its invention by Armstrong in 1917. Fig. 1.2 shows the classical superheterodyne architecture. In this architecture, the received signal is translated into a fixed Intermediate Frequency (IF) that is lower than the center of the RF signal. The frequency translation is done in two stages because of many advantages of such an architecture: it has lower filtering and quality factor requirements and relaxes the need for isolation between the mixer inputs and the Local Oscillator (LO). The bandwidth or center frequency of the superheterodyne's filters cannot be changed. They are designed according to specific standards that makes this architecture unsuitable for the wideband RF front end of a SWR system.

1.2.3 Ideal SoftWare Radio architecture

The ideal SWR architecture is given in Fig.1.3. In this architecture the digital part of the receiver is placed as close to the antenna as possible.

The need for this software radio architecture raises a number of technical challenges, which play a significant role in the development of future personal communication systems generation. Most of these technical challenges are related to analog-to-digital conversion, which in many cases cannot provide the advanced hardware platforms needed to support the demanding telecommunication services.

Then, the major problem that the SWR technology faces is that the actual ADCs are not able to cope with that very high frequency signals. All signals need to be converted

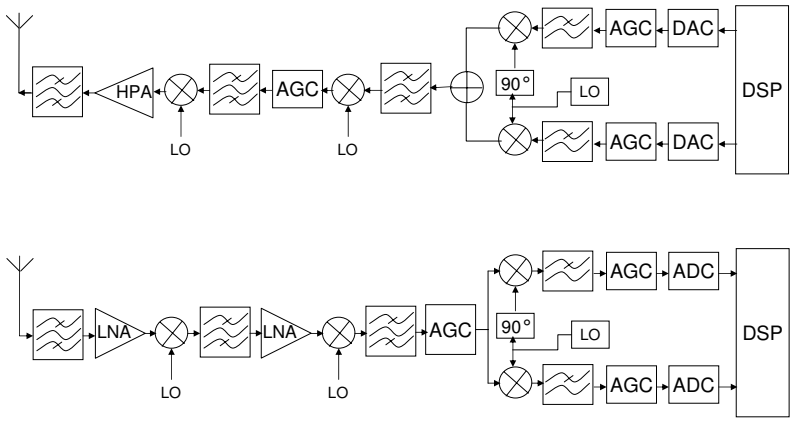

Figure 1.2: Superheterodyne transmitter/receiver

to the digital domain, otherwise processing makes no sense. Thus, the ADC is a key component of a software radio terminal [4]. The available ADCs sample at rates of nearly 100 Million Samples Per Second (MSPS) and quantize the signal with 14 bits. These performances do not fulfill the desired level of the required dynamic range mainly when the ADCs have to cope with signals of large bandwidth and high dynamic range. Still, it should be mentioned that beside the dynamic range, the sample rate has to fulfill the Nyquist criterion. These considerations leads us to conclude that the ideal software radio architecture of Fig.1.3 is not feasible today. Therefore, the bandwidth the ADC has to digitize must be reduced. The solution would be provided by SDR architectures as shown in the following subsections.

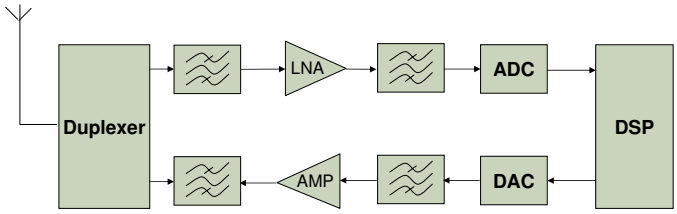

Figure 1.3: The ideal software radio architecture

1.2.4 Direct conversion architecture

The direct conversion architecture needs lower number of parts and it is conceptually attractive due to its simplicity. Its main advantage is that there are no need for any translation into IF. In the receiver side, the signal is directly down converted to baseband. The down converted signal is then prefiltered by a variable frequency anti-aliasing filter and, after analog-to-digital conversion, desired channels are chosen by software filters. Fig. 1.4 illustrates the transmitter-receiver architecture. Direct conversion has been so far suitable only for modulation methods that do not have significant part of signal energy near DC

(Direct Current). Despite its advantages, this architecture presents two main drawbacks. The first one is associated with the fact that the LO is at the signal band which poses possible unauthorized emissions and internal interference. Thus, this architecture needs an extremely stable LO. This problem can be compensated with digital post processing. The second inconvenience comes from the fact that this LO is not capable to synthesize all the carrier frequencies of the different standards the receiver has to support. Although these inconveniences, the direct conversion architecture was suggested as promising architecture for the future SDR systems [5] since it can offer the possibility to switch between some specific bands.

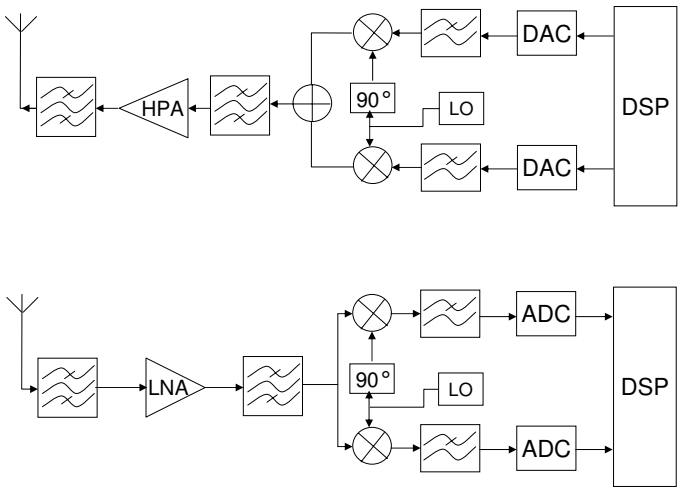

Figure 1.4: Direct conversion architecture

1.2.5 Feasible SDR architecture

Under the strong constraints discussed above, many publications [2] [6] [7] claim that to cover all services to be supported by the software radio terminal, a limited bandwidth has to be selected out of the full band by means of analog conversion and IF filtering. This concept leads to feasible SDR architecture sketched in Fig.1.5, where the ADC splits the communication chain into two parts: the AFE and the Digital Front End (DFE).

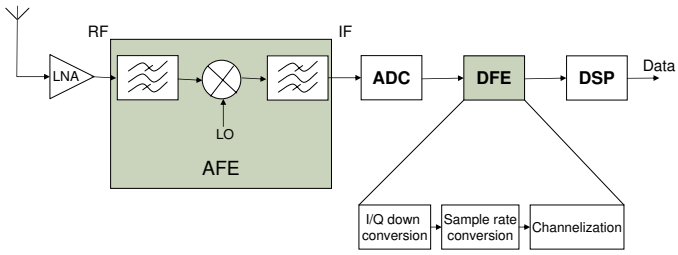

Figure 1.5: The feasible software radio receiver architecture

The AFE selects a bundle of channels and shifts their bandwidth from RF to an IF with which the ADC has to cope with. The DFE is a part of the receiver realizing front-end functionalities digitally that were formerly realized by means of analog signal processing (i.e., down conversion, channelization and sample rate conversion). Channelization comprises of all tasks necessary to select the channel of interest. This includes conversion to baseband, channel filtering, and possibly despreading. Sample rate conversion is a functionality that comes from the idea that it is surely sensible to sample the analog signal at a fixed rate. This simplifies clock generation for the ADC, which would otherwise be parameterizable. However, signals generally have to be processed at symbol or chip rates dictated by the different standards. Both facts lead to the necessity to digitally convert the digitization rate to the rate of the current standard of operation.

The conclusion we can draw from the above discussion is that the SWR system able to be realized today is a system at half digitized and the second half cannot be digitized before several years. That is, before the having availability of advanced ADCs able to provide an extreme dynamic range and very high sample rate capable to digitize the bandwidth of all services to be supported by the terminal and directly after the antenna.

Next section gives a brief description of ADC needed for SWR.

1.3 A/D and D/A conversion

As previously mentioned, the ADC and DAC are among the most important components [3] for SWR systems. In many cases, they define the bandwidth, the dynamic range and the power consumption of the radio. The wideband ADC is one of the most challenging tasks in software design. The wideband and the dynamic range of the analog signal has to be compatible with the ADC. An ideal SWR would use data converters at RF which would result in conflicts requiring a very high sampling rate, a bandwidth up to several GHz and a high effective dynamic range while avoiding intolerable power consumption. The physical upper bound for capabilities of ADCs can be derived from Heisenberg's uncertainty principle. For instance, at 1 GHz bandwidth the upper limit for dynamic range is 20 bits or 120 dB [1]. However there are other limiting factors including aperture jitter[(1)] and thermal effects. Unfortunately, the advances in ADC performances are very slow, unlike in many other technologies related to software defined radio.

The Nyquist rate $f_s/2$ determines the maximum frequency for which the analog signal can be faithfully reconstructed from the signal consisting of samples at the sampling rate f_s. Higher frequencies cause aliasing and therefore the ADC should be preceded by an anti-aliasing filter. The number of bits in the ADC defines the upper limit for achievable dynamic range. A higher dynamic range requires a higher stop-band filter attenuation. For instance, a 16-bit ADC needs over 100 dB attenuation in order to reduce the power of aliased signal under the half of the energy of the Least Significant Bit (LSB)[1]. The state of the art ADCs for wirless devices operate at bandwidth of over 100 MHz with 14-bit resolution and 100 dB Spurious Free Dynamic Range (SFDR[(2)]), but there is already commercial demand for even better converters for base stations [2]. The analog front end

[(1)] APERTURE JITTER is the variation in aperture delay from sample to sample. Aperture jitter shows up as input noise

[(2)] SFDR is a measure of the relative size of the largest harmonic with respect to the fundamental for a defined range of pure sine-wave input frequencies.

of an ADC has a direct influence on the dynamic range. Different air interface types and standards have different demands on the dynamic range. A large SFDR is needed to allow recovery of small scale signal when strong interferences are present. Different types of receiver architectures need different sampling methods [2]. A superheterodyne receiver or a direct conversion receiver may have an I/Q baseband signal as the analog output, for which quadrature baseband sampling is needed. Another possibility is an intermediate frequency analog output, for which a suitable sampling strategy is required. As an example, IF band pass sampling by using a sigma-delta ADC. Direct sampling is a suitable method for low IF analog signals.

After this brief introduction of SWR technology and the various related architectures and aspects, our studies will focus on the digital part of the receivers and specifically on the physical layer. The SWR's problematic, in this work, will be tackled from the viewpoint of a research of commonalities between several standards and in the standards themselves. In this context, an important technique called *parametrization* that aims to optimize the resources use in the SWR system was introduced [8]. This technique will be presented in details in the next section where there are two approaches. These are theoretical and pragmatic approaches which can describe the conception and realization of parameterizable SWR systems. Before doing this, let us present some major SDR projects.

1.4 Software Defined Radio projects

This section reviews some principal international, European and French SDR projects that contributed to the evolution of SDR from 1990 up till today.

- **SPEAKeasy** : SPEAKeasy was a US department of Defence program that aimed to prove the concept of multi-band, multi-mode software programmable radio operating from 2 MHz to 2 GHz [9]. The SPEAKeasy was designed as a totally open architecture that provided secure connections, interoperability and programmability.

  This SPEAKeasy project was divided into two phases: SPEAKeasy Phase I and SPEAKeasy Phase II. The main goal of the SPEAKeasy Phase I (1992-1995) was to develop a reconfigurable modem with an open architecture and to demonstrate its feasibility [10]. The objectives were to prove the potential of the SDR, to solve interoperability issues and provide a software architecture that would support the addition of new waveforms.

  The objective of SPEAKeasy Phase II was to extend the operational scope from the modem to an open, modular and reconfigurable architecture for the whole radio [10]. The capabilities were supposed to include reprogrammable security, wideband modem and continuous RF coverage up to 2 GHz. Motorola was the main contractor of the Phase II and has designed a wideband RF transceiver, which reduced distortions caused by the IF processing by using the homodyne design [10].

- **JTRS** (Joint Tactical Radio System): The JTRS program, also funded by the US department of Defense, was a process consisting of three steps that aimed to define, standardise and implement an architecture for software defined radios [11] [12]. The SCA (Software Communication Architecture) defined by the JTRS, is the current architecture of reference in the US and international military area. Commercial

systems could even go towards some modified SCA. SCA is the most important topic at SDR forum.

- **JCIT** (Joint Combat Information Terminal): The JCIT is a multi-band, multi-mode SDR developed by the US Navy Research Laboratory (NRL) [13]. The JCIT was designed as product for Army avionics, operating in frequency bands from HF up to 2.5 GHz. The focus of the design was on the hardware capacity, i.e. the extensive use of FPGAs and DSPs. The JCIT program has also made a significant contribution to the SDR Forum.

- **CHARIOT** (Changeable Advanced Radio for Inter-Operable Telecommunications): The CHARIOT project was designed at Virginia Tech. The focus was on the formalized architecture that allowed the use of dynamically reconfigurable hardware in SDRs [3]. The architecture was designed to be scalable and flexible by using a layered model. The layered Radio architecture for interfacing comprises three layers: the Soft Radio Interface (SRI), the Configuration Layer (CL) and the Processing Layer (PL).

- **GNU Radio**: GNU Radio is a software project that, combined with a minimal hardware, can be used for building radios whose waveform processing is defined in software [14] with the conventional sharing rules of GNU licensing. The purpose of GNU radio is to perform signal processing in software. GNU Radio consists of GNU Radio Software and GNU Radio Hardware. The hardware required for building a receiver is composed of an RF front end and an ADC. GNU Radio Software is organized in a such way that a graph which describes the data flow in the radio system is handed off to the runtime system for execution.

In addition to these projects, some forums involving governmental and industrial actors play important roles in SDR activities. The most active one is the **SDR Forum** [15]. This forum is the main actor of the Software Radio and Cognitive Radio community. It is historically based in the US and regrouping academic, governmental and industrial actors (Aeronix, Air Force Research Laboratory-US, France Télécom, Hitachi Ltd, NEC, NTT DoCoMo, Siemens AG, Thales, Toshiba, Virginia Tech, Xilinx, ...). Its main objective is to provide an international forum to research and engineering organizations, software and technology developers, radio communication service providers and other interested parties to exchange ideas, develop concepts, establish requirements, recommendations, specifications and standards.

European Effort

European Commission funded research activities that are mainly implemented within two frame work programmes: ACTS (Advanced Communications Technologies and Services) and IST (Information Society Technologies).

In the context of the ACTS and IST, Europe has funded several projects related to SDR [16] [17]. The main projects are listed below:

FIRST: Flexible Integrated Radio System Technology
SORT: Software Radio Technology

SUNBEAM: Smart Universal Beamforming
CAST: Configurable radio with Advanced Software Technology
DRIVE: Dynamic Radio for IP services in Vehicular Environments
MOBIVAS: Download Mobile Value Added Services Through Software Radio and Switching integrated Platform
PASTORAL: Platform and Software for Terminals Operationally Reconfigurable
SODERA: Reconfigurable radio for Software Defined Radio for 3rd generation mobile terminals
TRUST: Transparent Reconfigurable Ubiquitous Terminal
SCOUT: Smart User-centric Communication Environment
E2R: End-to-End Reconfigurability
E2R-II: End-to-End Reconfigurability phase II
ORACLE: Opportunistic Radio Communications in Unlicensed Environments
NEWCOM: Network of Excellence in Wireless Communications

End to End Reconfigurability: E2R project constitutes the major project including about 32 organizations (Motorola, Nokia, France Télécom, CEA, Eurecom, Supélec, ...). The key objective of the E2R project is to devise, develop, trial and showcase architectural design of reconfigurable devices and supporting system functions to offer an extensive set of operational choices to the users, application and service providers, operators, and regulators in the context of heterogeneous systems [18].

In addition to these projects, the European Commission has participated to several actions (conferences, meeting and workshops) related to the SDR: ACTS in 1998 and IST Mobile and Wireless Communication Summit'00 to Summit'07.

Some French efforts

In France, several **RNRT** (Réseau National de Recherche en Télécommunications) projects are funded. RNRT priorities for year 2002 focused among others on high level system design and associated verification tools of the Software Radio architecture implementation on the platform. The main targeted application domains are Software Radio system for 3G and 4G and SoC in particular. We quote here the main RNRT projects:

PETRUS: Plateforme d'Evaluation des Technologies Radio pour l'UMTS-TDD et des Services

PLATON: PLATeforme Ouverte pour les Nouvelles générations de communications mobiles

RHODOS: Réseau Hybride Ouvert pour le Déploiement des Services mobiles

PLATONIS: PLATeforme de validaTiOn et d'expérimentatioN multI-protocoles et multi-Services

A3S: Adéquation Architecture/Application Système

MOPCOM: Modélisation et spécialisatiOn de Plates-formes et COmposants MDA

IDROMel: Impact des Equipements Reconfigurables pour le Déploiement des Futures Réseaux Mobiles

1.5 The parametrization technique

A conventional approach to the implementation of multi-standard radio is the utilization of multiple transceiver chains each dedicated to an individual standard. Such an approach is not flexible, as most of the hardware needs to be replaced whenever the characteristics of the interface change. This conventional approach called "Velcro approach" does not exploit any common aspects between the different standards. In order to exploit to great advantage the commonalities among the various signal processing operations for different standards, one need firstly to identify these commonalities and secondly find the optimal way to implement a generic hardware platform with programmable modules. This new platform will be capable to run the appropriate software module depending on the software requirements. In this context, a technique called *parametrization* was introduced [8]. This technique can lead a designer of multi-standard system to an optimal architecture that balances complexity and performances. The key idea is to get an optimal sharing between hardware and software resources and a best way to reuse some hardware and software modules without affecting the system's performances. Initially, this technique which is regarded as a conception methodology appears to be as a pure theoretical approach, but it can become a pragmatic and practical approach as soon as a set of rules are settled helping the evaluation of a developed system. Researchers are proceeding along several directions [8][19][20][21]. The aim was always to get a flexible radio system, but until now, there have not been an accurate definition of the parametrization technique. In this work, starting with some conceptual considerations we can say that the parametrization is a technique that aims at optimizing the cost-performance tradeoff while building a multi-standard terminal. It can be considered as a theoretical procedure while identifying the common functionalities between several standards and in the standards themselves, to become a pragmatic procedure whenever the performance and the cost of the SWR system has to be evaluated. At this point, the parametrization should involve identifying an optimal level of granularity, from which a component can be considered as a "common communication block" enabling its reuse by several applications. The selection of the most appropriate level of granularity helps the parametrization to balance between economy and computing efficiency. But how can one proceed to identify or select the most appropriate level of granularity ? A very promising procedure is to consider the parametrization as a technique based on two approaches which are the CF approach (higher level) and the CO approach (lower level). Such a procedure can lead a designer of a multi-standard SDR system to optimal architecture that balances costs and performances. The next subsections give a description of the theoretical approach as well as the pragmatic approach with its two sub-approaches.

1.5.1 Theoretical approach

Under its theoretical approach, the parametrization technique can be viewed as a conception methodology of a multi-standard system. It consists in elaborating a structural description of a SDR system intended to support several standards whose the different modules can be illustrated by a graph that represents the hierarchical level of each module functionality. Fig. 1.6 shows the graph of several standards (UMTS, GSM, IS95, etc.). The roots of this graph, at the coarsest grain (highest level) are the communication standards that the reconfigurable system needs to support. The lower levels represent the

decomposition of several processing elements (coding, equalization, synchronization, etc). Having the graphic illustration of each standard, the task focuses now on identifying the commonalities between the different standards functionalities in order to find the optimal path(s) with which the multi-standard system will be conceived at the minimum cost. Proceeding in this direction, the designer allows several standards to use the same components and share its costs, thereby enabling a global optimization in terms of manufacturing costs. On the other hand, minimizing the manufacturing cost will possibly decrease the performance of the system in terms of computational time that may be unacceptable in some practical applications. Consequently, during the parametrization phase, one should take into account the trade-off between manufacturing cost and performance.

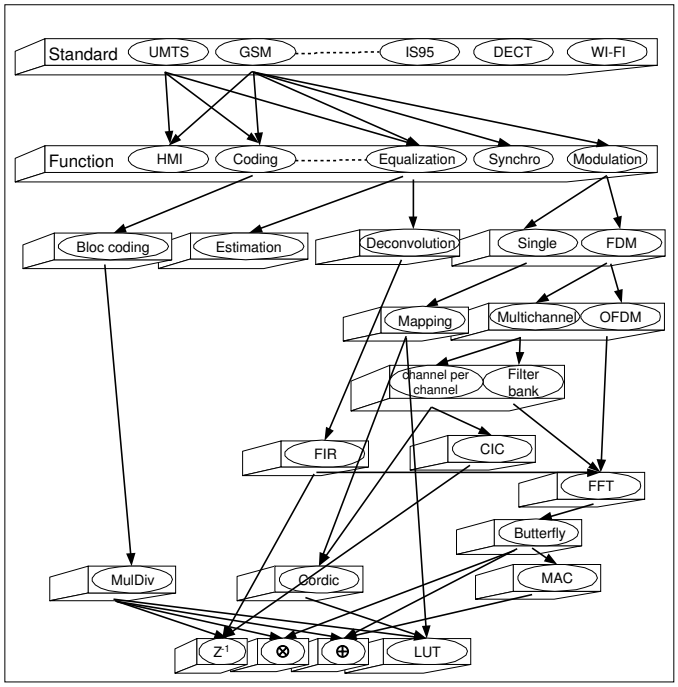

Figure 1.6: Global structure of the graph of several standard

In this sense, some researchers [22] have proceeded and introduced a mathematical model to identify the optimal architecture for a multi-standard reconfigurable radio. They modelled the radio as graph of progressively simpler functional modules and the optimal path can be selected according to a cost function value that accounts two key parameters: *monetary cost* and *computational cost*. This function is optimized by an heuristic based on simulated annealing and validated by comparison with an exhaustive search. The exhaustive search computes all the feasible solutions and compares them all and gives the true optimal solution. As the authors stated, this latest optimization method is very slow compared to the simulated annealing method [23], which provides the optimal solution with a good probability at relatively low computing cost.

To better illustrate the approach introduced in [22], we reproduce in Fig. 1.7 the developed graph in which the authors considered the investigation of the best choice between several design alternatives for the *equalizer* in the context of a multi-standard system

already including an *Orthogonal Frequency Division Multiplex* (*OFDM*) communication mode. The considered graph shown in Fig. 1.7 is a sub-graph corresponding to the decomposition of several processing elements (*equalization*, *multi-channel*, *OFDM*) that could be part of a SDR multi-standard system. Root nodes at the top (standards) are consequently not represented, but would be at a higher level not shown here. The numerical value under each node represent the *monetary cost* (MC)/*computational cost* (CC). The arcs are tagged with a number of calls (number of times a lower level block is invoked to perform a higher level function). Concerning the equalizer, its functionality can be realized either through the Finite Impulse Response (*FIR*) filter or through the *FFT*. By evaluating the cost function, the authors deduced that if we consider the independently implementation of the equalizer, it is less expensive to implement it through the FIR (1500=500+1000) than through the *FFT* (2000=1000+2× 500). A multiplication factor '2' settled between *equalization* and *FFT*, since the *FFT* and *Inverse Fast Fourier Transform* (*IFFT*) are used to perform the equalization and the *IFFT* is equivalent to *FFT* in terms of computation complexity. But in a combined design involving *OFDM* and equalizer modules, the overall cost of the design is only 2500 if the equalizer uses the *FFT*, versus 3000 if the equalizer uses the *FIR* (more details about the cost computations can be found in [22]). This is because the *monetary cost* of the *FFT* counts only once in the calculation of the overall cost of *OFDM* plus *equalizer*.

As the authors stated (in [22]), the simulated annealing optimization method used in their approach, is not specifically oriented to the problem at hand, and cannot guarantee a solution near the "true" optimum. For this, the authors recasted the problem as a "network design problem" and suggested that this method can lead to an optimal architecture [24]. They considered the graph of Fig. 1.7 and they discussed the building of the optimal path by applying network design methods which have extensive literature [25]. According to the authors ([24]), even if not formally proven that every conceivable graph of design choices cannot be exactly represented as a network design problem, the established method may provide in some cases a good approximation to the optimal design. To be more realistic, the authors stated to take into account the following issues: (i) building the hypergraph off all design choices, (ii) considering the time needed to reconfigure the radio while switching from a standard to another, (iii) determining the "travel time" of signals from a component to another, (iv) defining the contention among high level modules for the service of the same lower-level module when the considered radio intend to support simultaneous communication over several standards and (v) the energy consumption.

The above discussion deals with the theoretical conception of the SDR system. The practical realization of such system needs a methodology of conception which is, in its own right, an active area of research. From this perspective, [26] shows an efficient exploration methodology at a high level of abstraction. The proposed methodology is based on structural and physical estimations allowing the design of several architectures and the computation of their corresponding area and computing time. This methodology can be applied to the conception of SDR system while it permits to explore the candidate solutions and predict the cost of their technological mapping on the target hardware devices.

1.5.2 Pragmatic approach

In the theoretical approach, we have presented and discussed some methods developed to be applied to design an optimal multi-standard architecture. During the discussion,

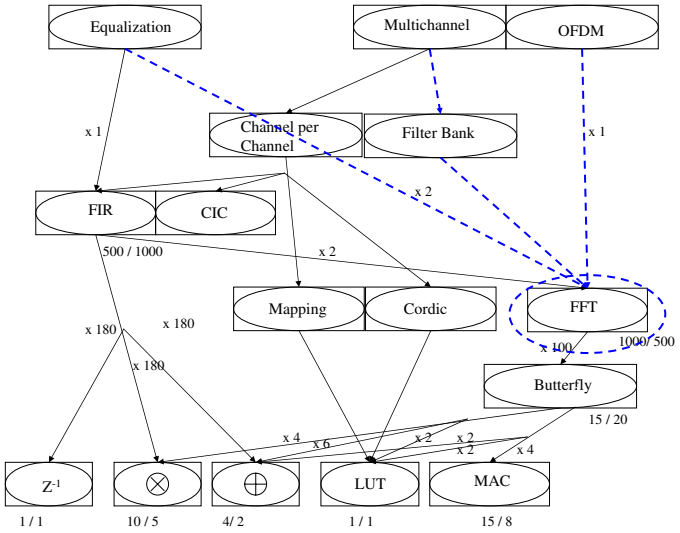

Figure 1.7: Partial hypergraph with some design parameters

realization of some functions as the equalization by adopting various ways was considered to illustrate the method and evaluate it by some figures. Although the useful insights that these methods provide, the approach remains theoretical while it does not reach the practical realization of the concerned multi-standard system.

By tackling the practical realization, the approach become more and more pragmatic and realistic. In this sense and by considering the parametrization technique, we distinguish between two common approaches: *Common Function* and *Common Operator*.

1.5.2.1 Common Function

By considering the structural illustration of the different communication components previously shown in Fig. 1.6 for a predefined set of standards, we can intuitively spot three intuitive hierarchical or granularity levels of these communications blocks. The functions (coding, equalization, modulation ...,) at the higher level and the primitive operators (multiplier, adder, etc.) at the lower level. Between these two levels, one can find an intermediate and flexible level that can be attached to one of the intermediate layers containing the remaining communications blocks (*Filter bank*, *FIR*, *FFT*, *butterfly* or *MAC*). The choice should be based on the application constraints: manufacturing cost and computational time.

Let us begin the discussion with the higher level of granularity voluntary chosen to be attached to the called CF approach. At this level, CF can be defined as a communication function needed by two or several standards. As an example, the coding function is a task required for all the concerned standards. So, instead of having dedicated coding functions, one can build a generic and reconfigurable one capable of responding to the requirements of the set of standards. One example of the common function approach, for GSM, UMTS and Professional Mobile Radio (PMR), concerning channel coding aspects is given in [19]. In this paper the author highlights the fact that parametrization allows communication

systems to be built with flexible components, under the restrictive assumption that these components belong to a predefined set of transmission modes. Using a single processing function, one can cover all standards under consideration.

In [27], a common structure called VITURBO for ordinary convolutional decoder and turbo decoder was proposed. This structure permits to perform the Viterbi and Turbo decoding.

In [20], an architecture for SWR receiver for GSM and UMTS Terrestrial Radio Access-Frequency Division Duplexing (UTRA-FDD) has been proposed. In this architecture several signal processing blocks dedicated to an individual standard are duplicated and another is built in such a way to be common to the two considered standards. For instance, there are two different methods used for equalization. In the case of GSM a trellis-based equalizer according to the Maximum A Posteriori Probability (MAP) criterion is employed while for UMTS-FDD a Rake receiver is used, which exploits the advantage of the orthogonality of the different multipath signals generated by using spreading codes. For channel estimation, both standards use the same function consisting in correlating the known training or pilot sequence with the received sequence. As for the demodulation, the authors explain in [28] the demodulation process for both GSM and UMTS signals. Moreover, two MAP algorithms are implemented in this architecture required for the turbo decoder of UMTS. In order to share common functions the authors proposed to adapt these two algorithms to be used as Viterbi-equalizer and consecutive convolutional decoder in the case of GSM. Indeed, as the equalization for GSM can be seen as a convolutional decoding with coding rate equal to 1, this same algorithm can also be used for the first convolutional decoder for turbo decoding in the case of UMTS. The second-MAP algorithm or convolutional decoder can be used by GSM for ordinary convolutional decoding as well as for the UTRA-FDD. In this latest case, the MAP-algorithm should be adapted for the use as a second convolutional decoder in the case of turbo decoding.

In the transceiver side, the same author explains in details in [8] how one can develop a general modulator structure that can process signals for several standards : GSM, IS-136, UTRA-FDD and Digital Enhanced Cordless Telecommunications (DECT). The modulation mode used in GSM is Gaussian Minimum Shift Keying (GMSK). GMSK is a special form of frequency shift keying which is a non linear modulation mode. In contrast to GMSK, the modulation modes Quadrature Phase Shift Keying (QPSK) (used in UMTS) and its derivative π-DQPSK (used in IS-136) are linear. Therefore, the authors [28] proposed to introduce the linearized version of GMSK into a SDR system. As the authors explain, the procedure consists in splitting-up the complex envelope for the GMSK signal into two summands with one representing the linear and the other representing the non-linear part. Moreover, the linear part contains about 99% of the signal energy. Therefore the GMSK signal may be well approximated by its linear part, i.e. in a software radio the symbols to be transmitted by a GMSK signal may be pulse shaped with an impulse $C_0(t)$ that leads to a linear modulation. Using this linear approximation, GMSK signals can be produced by the same linear I/Q modulator employed for PSK signals [8]. The proposed general modulator is shown in Fig. 1.8. This modulator is driven by several parameters and the modulation mode can change by an adjustment of these parameters. More details about the parametrization and functioning principle of this modulator can be found in [8].

All the above mentioned common structures of CF approach have a main drawback that these structures are directly related to a predefined set of standards. Consequently, if the

receiver architecture has a certain evolution, the CF should be redefined and redesigned to be capable to meet the requirements of the advanced standard. From here, comes the idea to proceed toward another approach that will give the possibility to built an open structure. By open structure, we mean a structure whose functionality can be used independently of the processing context or of the communication mode. This new approach called Common Operator approach will be discussed in details in the next section where some examples are given.

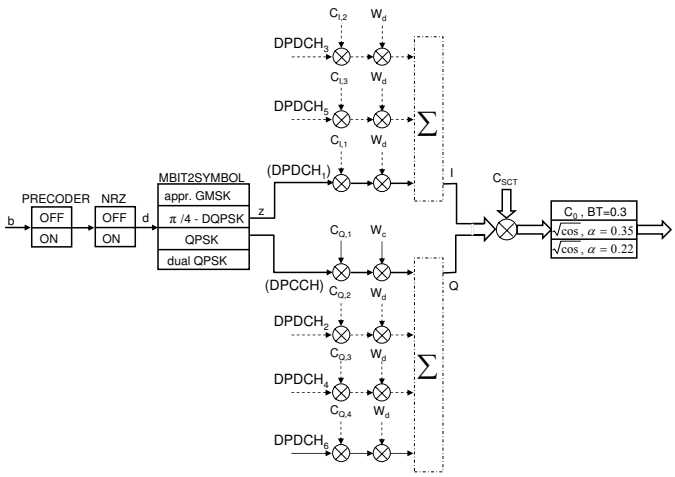

Figure 1.8: Generalized parameterizable modulator [8]

1.5.2.2 Common Operator

Let us consider the communication chain previously shown in Fig. 1.1 as an example to tackle our discussion about the CO approach. This Figure shows a communication chain composed of different processing blocks. The transmitter chain, for example, consists of source encoding, encryption, channel encoding, modulation and some functions performed by the RF front end. The functionality of each of these processing blocks is achieved by lower level processing operations. In the previous subsection we have discussed the commonality aspects between standards by identifying the CF among the various components of given standards. But if we go to lower level and also seek for commonality aspects we can find a wide area of common elements. This research can continue to attain a lower level where we find the primitive operators (adders, multipliers, etc). Although the goal is to find the maximum of common elements and then share their functionalities between several processing tasks, this research becomes useless and ineffective when the latency of systems exceeds certain limitations. So, the optimization in terms of hardware or software resources is directly dependent on the performance in terms of delay or execution time. In order to attain the best cost-performance trade-off, one should identify a level of granularity at which the designer will be based to implement the processing elements of communication standards.

In [21], the FFT has been identified as a CO. The authors show that many important tasks (such as filtering, equalization, channelization, OFDM modulation, etc) of a commu-

nication receiver can be implemented through the FFT. This FFT operator, identified as a common element, can be allotted during a given time slot to each function that necessitates the use of Fourier transform, which is exactly what the concept of common operator is about. Then, we can define a CO as an operator that, independently of the application context, can be reused by each function or processing requiring its functionality. However, this operator needs to be parameterized by some parameters (for example the size of transform, the word length, etc) to perfectly meet the requirements of each application. This is the origin of "parametrization" term.

Obviously, the CO is identified to be at a lower level of granularity than the CF. But in certain cases, these two approaches can meet depending on the required level of granularity. For instance, a FFT can be implemented with "butterfly" operations including some arithmetic operations (multiplication and addition). Then, there's no reason why we can't consider the FFT as CF and the butterfly as a CO. To avoid this ambiguity, we should consider the global granularity of the system and subsequently proceed to identify the common aspects and based the above definitions of CO and CF, we can recognize the identified common approach (CF or CO) according to its level of granularity. The CF is at the higher level whereas the CO is at the lower level. As a logic consequence, a CF can use a CO and the inverse is incorrect.

In the next subsections, two examples of common operators are given: MulDiv and FFT.

1.5.2.3 MulDiv operator

An operator called "MulDiv" has been identified [29]. The reason lies in the fact that most binary error protection operations in telecommunication standards are based on almost the same structure, presented in Fig. 1.9. These operations mainly concern Cyclic Redundancy Check (CRC) and block channel coding whose coding principles involve the use of a generator polynomial. The purpose is then to identify a general common structure applied to cyclic error protection, and the way to parameterize it.

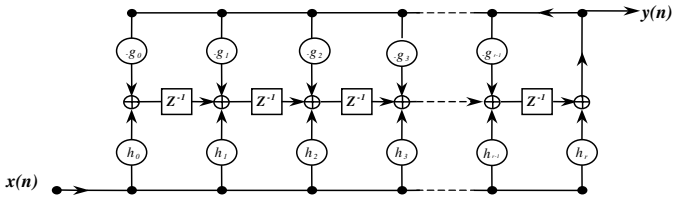

Figure 1.9: General IIR structure

Fig. 1.9 represents an architecture that carries out the transfer function of an Infinite Impulse Response (IIR) filter whose expression is:

$$H(x) = \frac{h_0 + h_1 x + ... + h_r x^r}{g_0 + g_1 x + ... + g_{r-1} x^{r-1} + x^r}.$$

Fig. 1.10 represents the general architecture of the MulDiv operator to be used in cyclic codes. This structure, derivated from the structure of Fig 1.9, is implemented with the

maximum number of shift registers and the feedback connections are parameterizable in such a way to match the maximum number of cyclic code circuits. As shown in Fig. 1.10, two switches have been added to correctly perform the polynomial division required in the encoding process. Appendix A gives several examples illustrating the parametrization of the MulDiv operator used as a CO in CRC calculation and Reed Solomon codes.

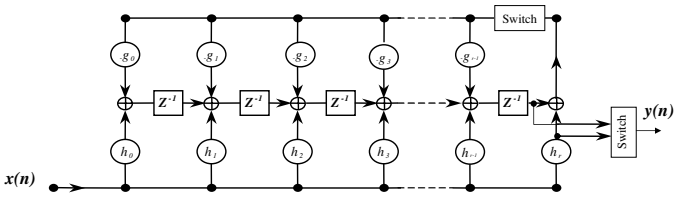

Figure 1.10: MulDiv operator structure

At the end of the master thesis of [29], we have found that MulDiv name is registered name as *Delphi/kylix math function* (*Function MulDiv*(*nNumber*, *nNumerator*, *nDenominator*: *Integer*): *Integer*;) and that Analog Devices [30] designed a similar operator that allows achieving the same basic coding operations in one machine cycle. This operator named *Blackfin DSP*, was proposed as a commercial processor using hardware operators and supporting one cycle instruction functions such as *BXORSHIFT* using Linear Feedback Shift Register (LFSR) principle to implement CRC codes, *BIXMUX* instruction to implement convolutional encoder.

The study in [31] deals with a reconfigurable LFSR as a CO.

1.5.2.4 FFT operator

The Discrete Fourier Transform (DFT), or its special and attractive class FFT, is extremely important in the area of frequency (spectrum) analysis as it takes a discrete signal in the time domain and transforms that signal into its discrete frequency domain representation. This area in digital signal processing was started with the publication of the Cooley-Tukey algorithm [32] which allows to reduce the order of complexity of some crucial computational tasks like Fourier Transform and convolution from N^2 to $N\ log_2 N$, where N is the transform length. This processing technique was rapidly credited by the tremendous increase of interest in Digital Signal Processors (DSP) beginning in the seventies and continued to play a key role in the widespread use of digital signal processing in a variety of applications like telecommunications, medical electronics, seismic processing, radars, etc.

Starting from these useful considerations of the FFT in several and important steps of digital communications, the authors in [21] have considered the FFT as CO and have detailed the various tasks which can be performed using the FFT operator.

The authors begin their discussion by presenting the filtering function, initially defined as a convolution product, as a product of the signal transform (input of the filter) and the frequency response of the filter. To keep the circular characteristics of the convolution, it is necessary to use specific techniques known as overlap-add and overlap-save techniques.

For the channel equalization, the authors show that any type of equalizer (except the MLSE) can be implemented in Frequency Domain (FD); starting from Fast Least Mean

Squares (FLMS) [33] and Unconstrained FLMS (UFLMS) [34] through implementation in FD [35], Quasi Newton algorithm in FD [36] and Decision Feedback Equalizer in FD [37] to new Single-Input Multi-Output (SIMO) semi-blind algorithms [38].

It was also shown in [21], that the channelization, channel estimation, (de)correlation multi-user detection, OFDM (de)modulation, despreading and Rake function can all be performed in FD using FFT. Fig. 1.11 illustrates by a graph the decomposition of three functions: equalization, multi channel and OFDM. By considering the implementation of these three functions in FD using the FFT, one can build a global communication block (block at the middle in the Figure) containing a Filter bank and a FFT (with its corresponding computational units) capable to perform the set of three functions. This result in saving some processing elements and consequently optimize the manufacturing cost of the corresponding multi-standard system.

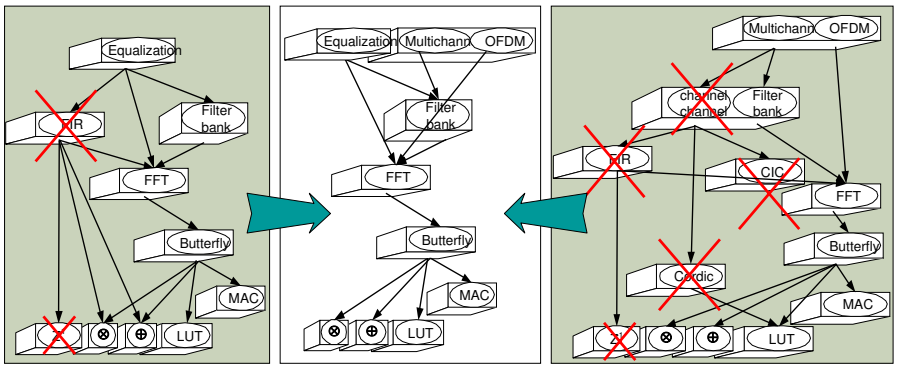

Figure 1.11: Optimal path for the channelization, equalization and OFDM demodulation functions using the FFT CO

1.6 Conclusions

Unlike the conventional frequency domain analysis of functions mentioned above, it is not clear what is meant by the terms "time domain" and "frequency domain" when we are working with channel coding function. The corresponding operations or transform computations of this function are defined over finite fields. In this work we will emphasize the notion of frequency processing of channel coding function and specifically the cyclic codes. The main motivation is to try to integrate the FFT operator in channel coding. At this point, the question is: how we can proceed to make the well known FFT operator able to operate in complex numbers field as well as in finite fields such as Galois Field ? The response will be in chapter 2 where the notion of reconfigurable FFT architecture will be introduced.

Chapter 2

The Fast Fourier Transform and channel coding

Contents

In this chapter we shall study the cyclic codes where we shall focus on frequency domain treatment of these codes. The choice of cyclic codes was based on our motivation to investigate the channel coding in the frequency domain. Being adapted to this processing, the cyclic codes will be studied in the setting of the Fourier transform. The most important classes of cyclic codes, the Reed-Solomon (RS) codes, are studied in this work. Encoding and decoding in both time and frequency domain are presented. The originality of our work does not lie in the discovery of the frequency processing of the cyclic codes but the idea is to highlight the interest of the use of the Fourier Transform. We insist on this vocabulary "Frequency processing" of cyclic codes and we emphasize the benefit of such a transform on the encoding and decoding processes of RS codes.

This chapter is organized as follows. After a brief description of the state of the art of the channel coding, we will provide in section 2.2 some elementary knowledge of algebra that will aid in the understanding and processing of cyclic codes. In sections 2.3 and 2.3.2, the basics of linear and cyclic codes are described. Section 2.4 presents the theory of finite field Fourier transform that we use in section 2.5 to give a general spectral interpretation of the cyclic codes and see how one can build an encoder in frequency domain. We also describe techniques and some principal algorithms for decoding the cyclic codes, including RS codes. The rest of the chapter will be dedicated to present a specific class of finite field transform such as the Fermat Number Transform (FNT) and their associated RS codes constructed over $GF(F_t)$. The investigation of these specific codes was motivated by the search of a transform that has a highly composite length allowing the adaptability between the finite field transform and the complex Fourier transform to subsequently design a common and reconfigurable architecture of the FFT which will operate over two different fields: complex and Galois fields.

2.1 Channel coding: state of the art

In any communication or recording system, the received signal is always affected by various kinds of parasites, gaussian or non-gaussian noise, interference, fading, dispersion, etc. The communication system or storage system must transmit its data with very high reliability in the presence of these channel impairments. During the last decades since the rise of data-transmission, coding techniques have been necessary in digital communications and storage devices to reduce the error rate in the received data stream and improve the efficiency of the information transmission.

The history of data-transmission codes began in 1948 with the publication of a famous paper by Claude Shannon. Shannon demonstrated that, by proper encoding of the information, errors induced by a noisy channel or storage medium can be reduced to any desired level without sacrificing the rate of information transmission or storage. Since Shannon's work, a great deal of effort has been expanded on the problem of devising efficient encoding and decoding methods for error control in a noisy environment.

Primitive communication and storage systems may seek to keep bit error rates as small as possible by simply transmitting high signal power or by repeating the message. These

simplistic techniques may be adequate if the required bit error rate is not too stringent, or if the data rate is low, and if errors are caused by noise rather than by defects or interferences. Such systems, however, buy performance with the least expendable resources: power and bandwidth.

In contrast, modern communication and storage systems obtain high performance via the use of elaborate message structure with cross-checks built into the waveform. The message will then undergo several processings, source encoding, channel encoding and modulation. The advantage of these modern communication waveforms is that high rates can be reliably transmitted while keeping the transmitted power and spectral bandwidth small. This advantage is offset by the need for sophisticated computations in the receiver (and in the transmitter) to recover the message. Such computations, however, are now regarded as affordable by using the modern electronic technology.

A communication system connects a data source to a data user through a channel transmission. Telephone lines, high-frequency radio links, coaxial cables, telemetry links, satellite links, and even magnetic and optical disks are examples of channels. A typical transmission (or storage) system may be represented by block diagram shown in Fig. 2.1. The information source can be either a person or a machine (e.g., a digital computer). The source decoder output feed into the destination, can be either a continuous waveform or a sequence of discrete symbols. The *source encoder* transforms the source output into a sequence of binary digits (bits) called the *information sequence* $\mathbf{u}$. The *channel encoder* transforms the information sequence $\mathbf{u}$ into a discrete *encoded sequence* $\mathbf{v}$ called a *code word*.

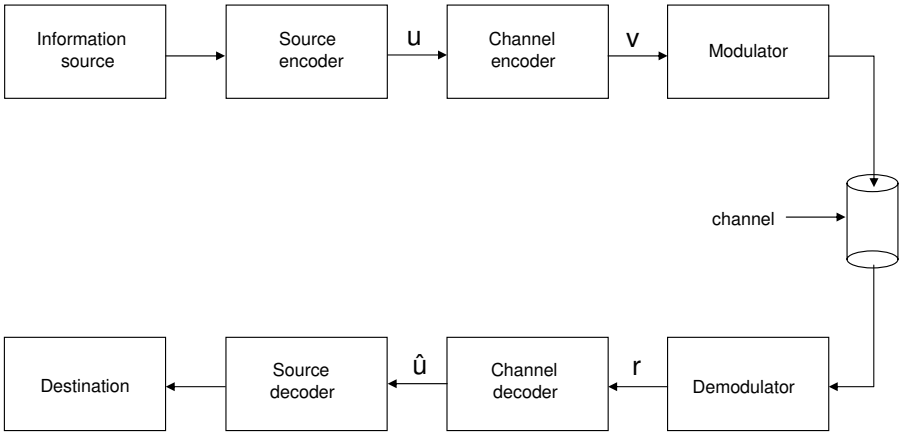

Figure 2.1: Block diagram of a digital communication system

Discrete symbols are not suitable for transmission over a physical channel or recording on a digital storage medium. The *modulator* transforms each output symbol of the channel encoder into a waveform of duration T seconds which is suitable for transmission (or recording). This waveform enters the *channel* (or storage medium) and is corrupted by noise. The demodulator processes each received waveform of duration T and produces an output that corresponds to the encoded sequence $\mathbf{v}$ called the *received sequence* $\mathbf{r}$. The *channel decoder* transforms the sequence $\mathbf{r}$ into a binary sequence $\hat{\mathbf{u}}$ called the *estimated sequence*. The decoding strategy is based on the rules of channel encoding and the noise

characteristics of the channel. Ideally, $\hat{\mathbf{u}}$ will be a replica of the information sequence $\mathbf{u}$, although the noise may cause some decoding errors. The *source decoder* transforms the estimated sequence $\hat{\mathbf{u}}$ into an estimate of the source output and delivers this estimate to the destination.

Considering the channel coding as the subject of our study, we will focus on this communication function and we begin by discussing the various classes of codes existing in the literature.

The are two principal categories of codes commonly used today: convolutional codes and block codes.

Convolutional codes

The convolutional codes were discovered in 1950 and were successfully decoded by sequential decoding algorithms. In 1967, a much simpler algorithm to decode them known as Viterbi algorithm was developed. In 1993, Berrou developed a powerful codes called the *turbo codes* which were seen as the central event of that period.

Block codes

The first block codes were introduced in 1950 when Hamming described a class of single-error-correcting block codes. Shortly thereafter Muller (1954) described a class of multiple-error-correcting codes. The major advances came when Bose and Ray-Chadhuri (1960) and Hocquenghem (1959) found a large class of multiple-error-correcting codes (the BCH codes), and Reed and Solomon (1960) found a related class of codes for nonbinary channels. Although these codes remain among the most important classes of codes, the theory of the subject since that time has been greatly strengthened, and new codes continue to be discovered. The discovery of BCH and RS codes led to a search for practical methods of designing the hardware or software to implement the encoder and decoder. The first good algorithm was found by Peterson (1960). Later, a powerful algorithm for decoding was discovered by Berlekamp (1968) and Massey (1969). Since that time, optimization of this algorithm and other algorithms continue to be published. Soft input and soft output decoding of block codes has been made possible thanks to Chase algorithm [39]. This has given rise to many concatenated block codes, the most famous ones are block turbo codes introduced by Pyndiah & al. in 1994 [40].

During the 1970s, the two different axis of research in channel coding began to draw together in some ways. In 1966, Forney introduced the concatenated codes where a block code (called external code) and a convolutional code (called internal code) are used together. The famous example of concatenated code is which that uses RS code concatenated with a convolutional code. This code was used for a long time on spacecraft-to-ground telemetry links to be replaced by the turbo code after 1993.

We should not forget to quote the special class of linear block codes that is Low-Density Parity-Check (LDPC) codes. This class of codes was first introduced by Gallager's thesis in 1961. After the discovery of turbo codes in 1993 by Berrou, LDPC codes were rediscovered by Mackay and Neal in 1995.

These codes show excellent performances in terms of error correction and start to compete with the turbo codes, particularly LDPC codes constructed in high Galois fields order [41]. Efficient decoding algorithm for these non-binary LDPC codes allowing the reduction of decoder complexity was introduced by Declercq [42].

2.2 Algebraic theory

Real numbers form a familiar set of mathematical objects that can be added, subtracted, multiplied and divided. Similarly, complex numbers form a set of objects that can undergo the same operations. For data-transmission codes, there is less familiar sets of objects. These sets of objects are also defined together with less familiar operations. Here, we talk about modular operations (addition, multiplication, etc). These mathematical structures are known as Galois Field (GF) and constitute the algebraic framework which provides the necessary tools to design encoders and decoders.

These sets with the corresponding operations will be described in the next subsections.

2.2.1 Groups

A group is a mathematical abstraction of an algebraic structure that may occur frequently in many forms.

Definition 2.2.1. *A group G is a set, together with an operation on pairs of elements of the set (denoted $*$), satisfying the following four properties.*

1. Closure : For every a, b in the set, $c = a * b$ is in the set.
2. Associativity : For every a, b, c in the set,

$$a * (b * c) = (a * b) * c.$$

3. *Identity : For every a in the set, there is an element e called the identity element, that satisfies*

$$a * e = e * a = a.$$

4. Inverse: If a is in the set, then there is some element b in the set, called an inverse of a such that

$$a * b = b * a = e.$$

If G has a finite number of elements, then it is called a finite group, and the number of elements in G is called the order of G.
Some groups satisfy the additional property that for all a, b in the group, $a * b = b * a$. This property is called the *commutativity property*. Groups with this additional property are called *commutative groups*, or *abelian groups*. In the case of an abelian group, the symbol for the group operation is written + and is called "addition".

2.2.2 Rings

A ring is an abstract set that is an abelian group and also has an additional structure.

Definition 2.2.2. *A ring R is a set with two operations defined: The first is called addition (denoted by $+$); the second is called multiplication (denoted by juxtaposition); and the following axioms are satisfied.*

1. *R is an abelian group under addition.*
2. *Closure: For any a, b in R, the product ab is in R.*
3. *Associativity:*
$$a(bc) = (ab)c.$$
4. *Distributivity:*
$$a(b+c) = ab + ac, \qquad (b+c)a = ba + ca.$$

The addition operation is always commutative in a ring, but the multiplication operation need not be commutative. A *commutative ring* is one in which multiplication is commutative, that is, $ab = ba$ where all a, b in R.

2.2.3 Fields

A field is a more powerful algebraic structure in which one can add, subtract, multiply, and divide.

Definition 2.2.3. *A field F is a set that has two operations defined on it: addition and multiplication, such that the following axioms are satisfied.*

1. *The set is an abelian group under addition*
2. *The set is closed under multiplication, and the set of nonzero elements is an abelian group under multiplication.*
3. *Distributivity:*
$$(a+b)c = ac + bc$$

for all a, b, c in the set.

Loosely speaking, an abelian group is a set in which one can add and subtract, a ring is a set in which one can add, subtract and multiply, and a field is a set in which one can add, subtract, multiply and divide.

Definition 2.2.4. *Let F be a field. A subset of F is called a subfield if it is a field under the inherited addition and multiplication. The original field F is then called an* extension field *of the subfield.* For example $GF(4)$ is an extension field of $GF(2)$.

2.2.4 Vector spaces

Definition 2.2.5. *Let F be a field. The elements of F will be called scalars. A set V is called a vector space, and its elements are called vectors, if there is defined an operation called vector addition (denoted by $+$) on pairs of elements from V, and an operation called scalar multiplication (denoted by juxtaposition) on an element of F and an element of V to produce an element of V, such that the following axioms are satisfied.*

1. *V is an abelian group under vector addition.*
2. *Distributivity: For any vectors v_1, v_2 and any scalar c,*

$$c(v_1 + v_2) = cv_1 + cv_2.$$

3. *Distributivity: For any vector v, 1v=v and for any scalars c_1, c_2,*

$$(c_1 + c_2)v = c_1v + c_2v.$$

4. *Associativity: For any vector v and any scalar c_1, c_2,*

$$(c_1c_2)v = c_1(c_2v).$$

Given a field F, the quantity $(a_1, a_2, ..., a_n)$, composed of field elements, is called an n-tuple of elements from the field F. Under the operations of componentwise addition and componentwise scalar multiplication, the set of n-tuples of elements from a filed F is a vector space and is denoted by the label F^n. A familiar example of a vector space is the space of n-tuples over the real numbers. This vector space is denoted R^n. Another familiar example is the vector space of n-tuples over the complex numbers, denoted C^n.

2.2.5 Construction of Galois Fields

The field known as *Galois Field* (GF) is the most powerful arithmetic system on which the coding theory is based. In this subsection, we will give the two principal procedures for constructing Galois fields. The first procedure is based on the integer rings and the second one is based on the polynomial rings. The description of these constructions seems to be useful for the understanding of the definition of the RS codes over the different GF.

2.2.5.1 Galois Field based on integer rings

Definition 2.2.6. *Let Z be a commutative ring and q be a positive integer. The quotient ring called the ring of integers modulo q, denoted by $\mathbf{Z}_q$, is the set {0,..., q-1} with addition and multiplication defined by*

$$\begin{aligned} a + b &= (a + b)\ mod(q), \\ a.b &= (a.b)\ mod(q). \end{aligned}$$

Any element a of $\mathbf{Z}$ can be mapped into $\mathbf{Z}_q$ by $a' = a\ mod(q)$. Two elements a and b of $\mathbf{Z}$ that map into the same element of $\mathbf{Z}_q$ are congruent modulo q, and $a = b + mq$ for some integer m.

Theorem 2.2.1. *The quotient ring $\mathbf{Z}_q$ is a field if and only if q is a prime integer.*
This is the easier way to construct a field from an integer ring.

2.2.5.2 Galois Field based on polynomial rings

A polynomial over any field F is a mathematical expression

$$f(x) = f_{n-1}x^{n-1} + f_{n-2}x^{n-2} + ... + f_1x + f_0,$$

where the symbol x is an *indeterminate*, the coefficients $f_{n-1}, ..., f_0$ are elements of F, the index of f_i is the integer i, and exponents of x are integers. A *monic polynomial* is a polynomial with leading coefficient f_{n-1} equal to one. A monic irreducible polynomial of degree at least 1 is called a *prime polynomial.*

The degree of a nonzero polynomial $f(x)$, denoted *deg* $f(x)$, is the index of the leading coefficient f_{n-1}.

Definition 2.2.7. *The set of all polynomials defined over $GF(q)$ where addition and multiplication are defined as the usual addition and multiplication of polynomials is called a polynomial ring. This polynomial ring denoted by the label $GF(q)[x]$ can be defined for each finite field $GF(q)$.*

Construction of Galois field based on polynomial ring

Finite fields can be obtained from polynomial rings by using constructions similar to those used to obtain finite fields from the integer ring. Suppose that we have $F[x]$, the ring of polynomials over the field F. Quotient rings in $F[x]$ can be constructed just as the construction of quotient rings in the ring $\mathbf{Z}$.

Definition 2.2.8. *For any monic polynomial $p(x)$ with nonzero degree over the field F, the ring of polynomials modulo $p(x)$ is the set of all polynomials with degree smaller than that of $p(x)$, together with polynomial addition and polynomial multiplication modulo $p(x)$. This ring is conventionally denoted by $F[x]/\langle p(x)\rangle$.*

Example

Let $p(x) = x^3+1$ a monic polynomial over $GF(2)$. The ring of polynomials over $GF(2)$ modulo $p(x)$ is $GF(2)[x]/\langle x^3+1\rangle$. It consists of the set $\{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$.

Theorem 2.2.2. *The ring of polynomials modulo a monic polynomial $p(x)$ is a field if and only if $p(x)$ is a prime polynomial.*

Using this theorem, whenever one can find a prime polynomial of degree m over $GF(q)$, then one can construct a finite field with q^m elements. In this construction, the elements are represented by polynomials over $GF(q)$ of a degree less than m. There are q^m such polynomials, and hence q^m elements in the field.

As an example, we will construct $GF(4)$ from $GF(2)$, using the prime polynomial $p(x) = x^2+x+1$. This polynomial is easily verified to be irreducible by testing all possible factorizations. The field elements are represented by the set of polynomials $\{0, 1, x, x+1\}$. Appendix A gives a list of prime polynomial over $GF(2)$.

2.2.5.3 Primitive elements

Definition 2.2.9. *A primitive element of the field $GF(q)$ is an element α such that every field element except zero can be expressed as a power of α.*

For example, in the field $GF(5)$ we have: $2^1 = 2$, $2^2 = 4$, $2^3 = 3$, $2^4 = 1$, and thus 2 is a primitive element of $GF(5)$. In contrast, 4 is not a primitive element in $GF(5)$ because 2 cannot be expressed as a power of 4. Primitive elements are useful for constructing fields, because if we can find one, we can construct a multiplication table by multiplying powers of the primitive element. Appendix A shows how one can construct $GF(16)$ using a primitive element of order 15.

2.3 Linear block codes

Definition 2.3.1. *A linear block code $\mathcal{C}$ defined over $GF(q)$ is referred to a $\mathcal{C}(n,k)$ code where n is the blocklength of the code and k is the blocklength of the information sequence. Then, a linear block code is a subspace of $GF(q)^n$ which associates to k information symbols, $n-k$ symbols called "redundancy symbols".*

A linear code is invariant under translation by a codeword. That is, for any codeword $\mathbf{c}$, $\mathcal{C}+\mathbf{c}$=$\mathcal{C}$. This means every codeword in a linear code bears a relationship to the rest of the code that is completely equivalent to the relationship any other codeword bears to the rest of the code. An important characteristic of the linear code is the *minimum distance* of the code. Suppose c_i and c_j are any two codewords in an (n,k) block code. The measure of the difference between the codewords is the number of corresponding elements or positions in which they differ. This measure is called *Hamming distance* between the two codewords and is denoted as d_{ij}. The smallest value of the set $\{d_{ij}\}$ is called the *minimum distance*.

Hence to find a linear code that can correct t errors, one must find a linear code whose minimum distance satisfies

$$d_{min} \geq 2t + 1.$$

2.3.1 Matrix description of linear block codes

A linear block code $\mathcal{C}$ is a subspace of $GF(q)^n$. The theory of vector spaces can be used to study these codes. Any set of basis vectors for the subspace can be used as rows to form a k by n matrix $\mathbf{G}$ called the *generator matrix* of the code. The row space of $\mathbf{G}$ is the linear code $\mathcal{C}$; any codeword is a linear combination of the rows of $\mathbf{G}$. The set of q^k codewords forms the $\mathcal{C}(n,k)$ linear code.

The rows of G are linearly independent, while the number of rows k is the dimension of the code. The matrix $\mathbf{G}$ has rank k. To generate a codeword from a sequence information $\mathbf{m}$ of k symbols, the most natural approach is to use the following [43]:

$$\mathbf{c} = \mathbf{m}\ \mathbf{G}.$$

Let us consider a simple example of a binary linear code. First, take the 3 by 5 generator matrix

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

The data vector $\mathbf{m} = [\ 1\ \ 0\ \ 1\ ]$ is encoded into the codeword

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

Because $\mathcal{C}$ is a subspace of $GF(q)^n$, it has a dimension k. This is equal to the number of rows in $\mathbf{G}$. Because $\mathcal{C}$ is a subspace, it has an orthogonal complement $\mathcal{C}^{\perp}$, which is the set of all vectors orthogonal to $\mathcal{C}$. The orthogonal complement is also a subspace of $GF(q)^n$, and thus itself is a code. $\mathcal{C}^{\perp}$ is called the dual code of $\mathcal{C}$ and has dimension $n-k$. Let $\mathbf{H}$ be a matrix with any set of basis vectors of $\mathcal{C}^{\perp}$ as rows. Then an n-tuple $\mathbf{c}$ is a codeword in $\mathcal{C}$ if and only if it is orthogonal to every row vector of $\mathbf{H}$. That is,

$$\mathbf{c}\mathbf{H}^T = 0$$

This *check equation* provides way for testing whether a word is a codeword of $\mathcal{C}$. The matrix $\mathbf{H}$ is called a check matrix of the code $\mathcal{C}$. It is an $(n-k)$ by n matrix over $GF(q)$. Because $\mathbf{c}\mathbf{H}^T = 0$ holds whenever $\mathbf{c}$ is equal to any row of $\mathbf{G}$, we have

$$\mathbf{G}\mathbf{H}^T = 0$$

Every generator matrix $\mathbf{G}$ can be converted to a generator matrix for an equivalent code called *systematic code*. This new generator matrix is of the form

$$\mathbf{G} = [\mathbf{I}\ \ \mathbf{A}]$$

where $\mathbf{I}$ is a k by k identity matrix and $\mathbf{A}$ is a k by $n-k$ matrix. With this new form of $\mathbf{G}$, the linear block encoder is called "*systematic encoder*" because it maps each dataword into a codeword with the k data symbols unmodified in the first k symbols of the codeword. The remaining symbols are called check symbols.

2.3.2 Cyclic codes

The cyclic codes over $GF(q)$ are an interesting class of linear block codes. Their importance comes from the fact that their structure is closely related to the strong structure of the Galois field. In the underlying GF, the cyclic codes can be encoded and decoded using efficient algorithms in terms of computational complexity and circuit implementations.

Definition 2.3.2. *Let* $\boldsymbol{c} = (c_0, c_1, ..., c_{n-1})$ *be a codeword in a linear code* $\mathcal{C}$. $\mathcal{C}$ *is called a* cyclic code *if* $\boldsymbol{c}' = (c_{n-1}, c_0, ..., c_{n-2})$ *is also in* $\mathcal{C}$. *The codeword* $\boldsymbol{c}'$ *is obtained by cyclically shifting the components of the codeword* $\mathbf{c}$ *one place to the right. Thus the cyclic shift maps the code into itself.*

Every cyclic code can be described by a polynomial description, where a vector $\mathbf{m}$ can be represented by polynomial $\mathbf{m}(x)$:

$$\mathbf{m} = [m_{k-1} \ ... \ m_1 \ m_0] \quad \Leftrightarrow \quad \mathbf{m}(x) = m_{k-1}x^{k-1} + ... + m_1 x + m_0,$$

and the corresponding codeword to a data sequence represented by its polynomial form $\mathbf{c}(x)$ is of the form:

$$\mathbf{c}(x) = \mathbf{m}(x)\mathbf{g}(x),$$

where $\mathbf{g}(x)$ is the *generator polynomial*, that characterizes the code. The polynomials $\mathbf{c}(x)$, $\mathbf{m}(x)$ and $\mathbf{g}(x)$ are polynomial of degree $n-1$, $k-1$ and $n-k$ respectively.

This encoder is nonsystematic because the data polynomial $\mathbf{m}(x)$ is not immediately visible in $\mathbf{c}(x)$. A systematic encoding can be obtained by a more complicated computation. The idea is to insert the data into the high-order coefficients of the codeword, and then choose the check symbols so as to obtain a legitimate codeword. Thus, the codeword can be written as

$$\mathbf{c}(x) = x^{n-1}\mathbf{m}(x) + \mathbf{v}(x)$$

taking into account that $\mathbf{c}(x)$ is a multiple of $\mathbf{g}(x)$, we can write

$$\mathbf{q}(x)\mathbf{g}(x) = x^{n-1}\mathbf{m}(x) + \mathbf{v}(x) \quad \Rightarrow \quad x^{n-1}\mathbf{m}(x) = \mathbf{q}(x)\mathbf{g}(x) - \mathbf{v}(x)$$

and the degree of $\mathbf{v}(x)$ is less than the degree of $\mathbf{g}(x)$. So, $\mathbf{v}(x)$ is the residue of $x^{n-1}\mathbf{m}(x)$ when divided by $\mathbf{g}(x)$.

In the error control codes domain, the most popular Galois field is $GF(q=2)$ which has the two elements "0" and "1". The corresponding extension field is $GF(2^m)$ which is the most useful field and under it most of the encoding and decoding techniques are done. This is the case of BCH codes, Hamming codes, etc. For these codes the time domain symbols are in $GF(2)$ and the spectral components are in $GF(2^m)$. For RS codes, the time domain symbols as well as the spectral components are in $GF(2^m)$ and for this reason the RS codes are called non-binary codes.

We began to talk here about the spectral components of cyclic codes. This notion peculiar to the complex Fourier transform can be quite defined over GF. In the following, we will describe the frequency domain treatment of cyclic codes by giving their frequency domain definition and explaining their encoding decoding process in this domain. Let us begin by giving a brief overview on the finite field transform.

2.4 The Fourier transform over finite fields

Application of the discrete Fourier transform in the complex field occurs throughout the subject of signal processing. Fourier transforms also exist in the Galois Field $GF(q)$ and can play an important role in the study and processing of $GF(q)$-valued signals that is codewords. By using the Fourier Transform, the idea of coding theory can be described in a setting that is much closer to the methods of signal processing.

In complex field, the discrete Fourier transform of $f = (f_0, f_1, ..., f_{n-1})$, a vector of real or of complex number, is a vector $F = (F_0, F_1, ..., F_{n-1})$, given by

$$F_k = \sum_{i=0}^{n-1} f_i e^{\frac{-j2\pi ik}{n}}, \qquad k = 0, ..., n-1, \tag{2.1}$$

where $j = \sqrt{-1}$. The Fourier kernel "$exp(-j2\pi/n)$" is an nth root of unity in the field of complex numbers. In the finite field $GF(q)$, an element α of order n is an nth root of unity. Drawing on the analogy between $exp(-j2\pi/n)$ and α, we have the following definition [44].

Definition 2.4.1. Let $f = (f_0, f_1, ..., f_{n-1})$ be a vector over GF(q), and let α be an element of $GF(q)$ of order n. The Fourier transform of the vector f is the vector $F = (F_0, F_1, ..., F_{n-1})$ with components given by

$$F_j = \sum_{i=0}^{n-1} f_i \alpha^{ij}, \qquad j = 0, ..., n-1, \tag{2.2}$$

and the vector f is related to its spectrum F by

$$f_i = \frac{1}{n} \sum_{j=0}^{n-1} F_j \alpha^{-ij}, \qquad i = 0, ..., n-1. \tag{2.3}$$

It is natural to call the discrete index i *time*, taking values on the *time axis* $0, 1, ..., n-1$, and to call f the *time-domain function* or the signal. Also, we might call the discrete index j *frequency*, taking values on the *frequency axis* $0, 1, ..., N-1$, and to call F the *frequency-domain* or the *spectrum*.

The Fourier transform in a Galois field [45] closely mimics the Fourier transform in the complex field with one important difference. In the complex field an element w of order n, for example, $e^{\frac{-j2\pi}{n}}$, exists for every value of n. However, in the field $GF(q)$, such an element w exists only if n divides $q-1$. Moreover, if for some values of m, n divides $q^m - 1$, then there will be a Fourier transform of blocklength n in the extension field $GF(q^m)$. For this reason, a vector f of blocklength n over $GF(q)$ will also be regarded as a vector over $GF(q^m)$; it has a Fourier transform of blocklength n over $GF(q^m)$. This is completely analogous to the Fourier transform of a real-valued vector: even though the time-domain vector f has components only in the real field, the transform F has components in the complex field. Similarly, for the GF Fourier transform, even though the time-domain vector f is over the field $GF(q)$, the spectrum F may be over the extension field $GF(q^m)$. Any factor of $q^m - 1$ can be used as the blocklength of a Fourier transform over $GF(q)$, but the most important values for n are the primitive blocklength, $n = q^m - 1$. Then $w = \alpha$ is a primitive element of $GF(q^m)$.

The Fourier transform over $GF(q)$ has many strong properties, which are summarized by the following.

1. Additivity: $af + bf' \quad \Leftrightarrow \quad aF + bF'$
2. Modulation: $(f_i \alpha^{il}) \quad \Leftrightarrow \quad F_{(j+l) mod\ n}$

3. Translation: $(f_{(i-l) mod\ n}) \quad \Leftrightarrow \quad F_j \alpha^{jl}$

4. Inverse: $f_i = \frac{1}{n} \sum_{j=0}^{n-1} F_j \alpha^{-ij}, \quad i = 0, ..., n-1,$
where $n = 1 + 1 + ... + 1$ (n terms).

5. Convolution: $e_i = \sum_{l=0}^{n-1} f_{(i-l) mod\ n}\ g_l, \quad i = 0, 1, ..., n-1,$
if and only if $E_j = F_j G_j, \quad j = 0, 1, ..., n-1$

6. $f(x) = \sum_{l=0}^{n-1} f_i x^i$ has a zero at α^j if and only if $F_j = 0$.

2.5 Frequency interpretation of cyclic codes over $GF(2^m)$

Transforms over Galois field have been introduced into the study of error control codes in order to reduce decoder complexity first by Gore [46], later by Michelson [47], Lempel and Winograd [48] and Chien [49]. Blahut [50] has proceeded in the sense to make these transforms play a much more central role in the subject. He showed that the idea of coding theory can be described in a frequency domain setting that is much different from the familiar time domain setting. In this context, by frequency domain reasoning, he described several encoder and decoders schemes. In this section, we present the spectral interpretation of the encoding of cyclic codes.

2.5.1 Frequency encoding of cyclic codes over $GF(2^m)$

Let $f(x)$ be a polynomial whose coefficients f_i, i=0, ..., $n-1$, are elements of $GF(q)$:

$$f(x) = f_0 + f_1 x + ..., f_{n-1} x^{n-1},$$

where n divides $q^m - 1$ for some m integer, and let α be an element of $GF(q^m)$ of order n. The finite field Fourier transform of the vector $f = \{f_0, f_1, ..., f_{n-1}\}$ is the vector over $GF(q^m)$, $f = \{F_0, F_1, ..., F_{n-1}\}$, and as given in equation 2.4 can be written as

$$F_j = \sum_{i=0}^{n-1} f_i \alpha^{ij}. \tag{2.4}$$

For simplicity of exposition and for the first time we will restrict attention to values of n satisfying $n = q^m - 1$. These values of n is called primitive blocklengths. Then α is a primitive element of $GF(q^m)$.

The spectrum polynomial of $f(x)$ or the associated polynomial by means of the finite field Fourier transform can be written as

$$F(x) = F_0 + F_1 x + ..., F_{n-1} x^{n-1}.$$

There is a relationship between the root of $f(x)$ and the coefficients of $F(x)$. The polynomial $f(x)$ has a root at α^j if and only if F_j equals zero and the polynomial $F(x)$ has a root at α^{-i} if and only if the ith time component f_i equals zero. Thus, in finite fields, when one speaks of roots of polynomials or of spectral components equal to zero, one really speaks of the same thing, but the terminology and the insights are different.

A cyclic code over $GF(q)$, as previously presented in section 2.3.2, can be described in terms of generator polynomial $\mathbf{g}(x)$ over $GF(q)$ of degree $n-k$, where k is the length

of the dataword $\mathbf{m}(x)$. Every codeword is represented by a polynomial of degree $n-1$, written as $\mathbf{c}(x) = \mathbf{m}(x)\mathbf{g}(x)$. This is a convolution in time domain

$$c_i = \sum_{k=0}^{n-1} m_k g_{i-k}.$$

Therefore, this time domain convolution can be converted into a componentwise multiplication and the product can be written as

$$C_j = M_j G_j.$$

The encoding operation can be then carried out by a multiplication of spectral components of the two vectors $\mathbf{M}$ and $\mathbf{G}$.

By definition, the generator polynomial can be written as

$$\mathbf{g}(x) = (x-\alpha^{j_0})(x-\alpha^{j_0+1})...(x-\alpha^{j_0+i})...(x-\alpha^{j_0+d-2}),$$

where d is the minimal distance of the code and j_0 is an integer. Thus, in the frequency encoding, the role of the generator polynomial is to specify the spectral components that should be zero, the remaining components of the spectrum are filled with information symbols. The spectral components that are constrained to zero are those whose indices correspond to $\alpha's$ powers (roots of the generator polynomial). The inverse transform of the built frequency vector provides the time domain codeword which will be transmitted. The frequency encoding process is illustrated in Fig. 2.2.

A cyclic code can now be defined as the set of inverse Fourier transforms of all spectral vectors that are constrained to zero in several prescribed components [44].

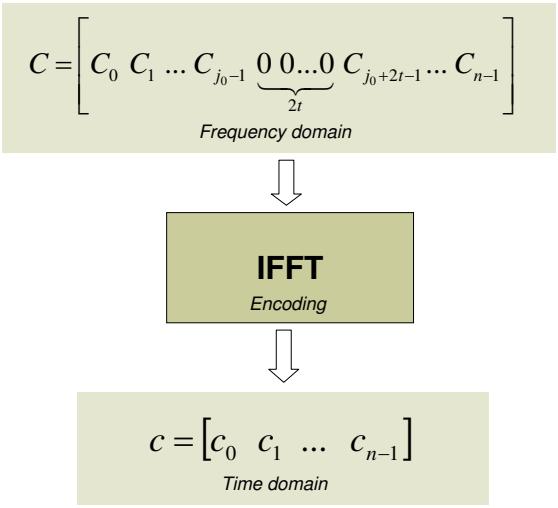

Figure 2.2: Encoding process using Fourier transforms

2.5.2 Frequency encoding of RS codes over $GF(2^m)$

Definition 2.3.3. *A RS code of block length n over $GF(q)$, with n a divisor of $q-1$, is defined as the set of all words over $GF(q)$ of length n whose Fourier transform is equal to zero in a specified block of $d-1$ consecutive components, denoted $\{j_0, j_0+1, ..., j_0+d-2\}$.*

The blocklength of an RS code over $GF(q)$ is directly related to the order of α. If α has an order $n = q-1$, α is a primitive element and the RS code is called a *primitive RS code*. The minimum distance of an RS code is

$$d_{min} = n - k + 1$$

where k is the blocklength of the information sequence.

In this section we consider the RS codes defined over $GF(2^m)$. This class of codes is the most popular class of cyclic codes. The RS codes are characterized by their powerful correction capacity of burst errors [51]. They are used extensively for correcting both errors and erasures in many systems as space communication links, Compact-Discs (CD), audio systems [52], High-Definition (HD) TV [53], Digital Versatile Discs and wireless communication systems. The time and frequency domain symbols are in the same GF. In section 2.10, we revive a less known class of RS codes defined over a specific Galois field and have some specific characteristics that will be discussed in details.

As for any cyclic code, the encoder of RS codes can be either systematic or nonsystematic. One may encode in the natural way using the generator polynomial. This encoder is called a *time domain encoder*. Alternatively, one may choose to encode the RS codes directly in the transform domain by using the data symbols to specify spectral components. This is called a *frequency domain encoder*.

Frequency encoding of an $RS(n,k)$ code is as follows. Some set of $d-1$ frequencies, indexed by $j = j_0, j_1, ..., j_0+d-2$, is chosen as the set of spectral components constrained to zero. The $n-d+1$ unconstrained components of the spectrum are filled with data symbols from $GF(q)$. A nonsystematic codeword is produced by taking the inverse Fourier transform of the frequency vector. The obtained RS code is referred to as $RS(n,k)$ where $k = n-d+1$. The common choice for j_0 is $j_0 = 1$ and the corresponding generator polynomial is

$$\mathbf{g}(x) = (x-\alpha)(x-\alpha^2)...(x-\alpha^{2t}).$$

This is always a polynomial of degree $2t$.

As an example, let us consider the code RS(7,5) with a polynomial generator

$$g(x) = (x-\alpha)(x-\alpha^2) = x^2 + \alpha^4 x + \alpha^3$$

and an information sequence

$$\mathbf{m}(x) = \alpha^6\ x^4 + x^3 + \alpha^4 x^2 + \alpha\ x + 1.$$

Using the two encoding techniques, we shall find the two codewords corresponding to $\mathbf{m}(x)$.

1. Frequency domain encoding: The components that should be constrained to zero are the components indexed by 1 and 2. Thus, the spectral vector can be written as

$$\mathbf{C} = [1 \quad 0 \quad 0 \quad \alpha \quad \alpha^4 \quad 1 \quad \alpha^6]$$

and the nonsystematic time domain codeword

$$\mathbf{c} = IFFT(\mathbf{C}) = [1 \quad \alpha \quad \alpha \quad 0 \quad \alpha^5 \quad \alpha \quad \alpha^6]$$

2. Time domain encoding: The time domain encoding consists in multiplying $m(x)$ by $x^{n-k} = x^2$ and performing the polynomial division x^2 $\mathbf{m}(x)$ by $g(x)$; The coefficients of the remainder $v(x)$ of this division represent the redundancy symbols. The resultant systematic codeword

$$\mathbf{c} = [\alpha^3 \quad \alpha^2 \quad 1 \quad \alpha \quad \alpha^4 \quad 1 \quad \alpha^6]$$

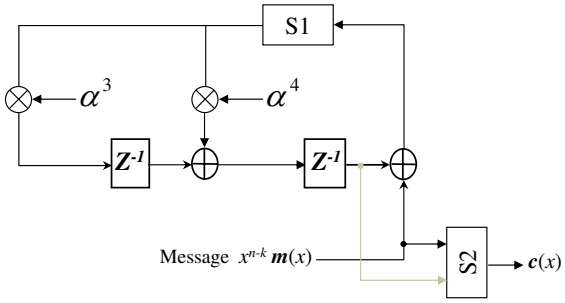

Figure 2.3: Encoding circuit (in time domain) for a RS(7,5) with a generator polynomial $\mathbf{g}(x) = x^2 + \alpha^4 x + \alpha^3$

The time domain encoding can be realized with a division circuit which is a linear $(n-k)$ stage shift registers with feedback connection based on the generator polynomial $\mathbf{g}(x)$. Such a circuit is shown in Fig. 2.3. The encoding operation is carried out as follows: The switch *S1* is turned on, the k information symbols $\alpha^3, \alpha^2, 1, \alpha, \alpha^4, 1, \alpha^6$ are shifted into the circuit and simultaneously through the switch *S2*. Shifting the message $\mathbf{m}(x)$ into the circuit from the front end is equivalent to premultiplying it by x^2. As soon as the complete message has entered the circuit, the two symbols in the register form the remainder and thus they constitute the redundancy symbols (or parity check symbols). The switch S2 is then turned off and the redundancy symbols are shifted out through the switch S2. These two symbols with the k information symbols, form the complete codeword $\mathbf{c}(x)$.

2.5.3 Frequency decoding of RS codes over $GF(2^m)$

During the transmission, it is possible that the encoded message (or codeword) $\mathbf{c}$ will be disturbed by noise that can be represented by an error word $\mathbf{e}$. Then, the received codeword $\mathbf{r}$ can be written as the sum $\mathbf{r} = \mathbf{c} + \mathbf{e}$. The decoder must process the received codeword so that the error word $\mathbf{e}$ is removed; the data is then recovered from the codeword.

The Fourier transform of the received word has components $R_j = C_j + E_j$ for $j = 0, 1, ..., n-1$. The $2t$ spectral components, from j_0 to $j_0 + 2t - 1$, are called the (frequency domain) syndromes. As mentioned in the previous section, the components C_j indexed by $j_0, j_0 + 1, ..., j_0 + 2t - 1$ are the components constrained to be zero whether the encoding is performed in frequency or time domain. The syndromes can be written as

$$S_j = R_{j+j_0-1} = E_{j+j_0-1} \qquad j = 1, ..., 2t.$$

It is convenient to index the syndromes starting with index one. To simplify this equation, we take $j_0 = 1$ and therefore $S_j = R_j$.

The block of $2t$ syndromes provides the window through which the decoder can start the decoding process to recover the data. Let us consider the error vector with its polynomial form

$$e(x) = e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + ... + e_1x + e_0.$$

The $2t$ syndromes can be defined by

$$S_j = \mathbf{r}(\alpha^j) = \mathbf{c}(\alpha^j) + \mathbf{e}(\alpha^j) = \mathbf{e}(\alpha^j) \qquad j = 1, ..., 2t.$$

As the code can correct at most t errors, the error polynomial has at most t coefficients that are nonzero. Suppose that t_1 errors occur, $0 \leq t_1 \leq t$ and that they occur at the unknown locations $i_1, i_2, ..., i_{t_1}$. The error polynomial can be written as

$$e(x) = e_{i_{t_1}}x^{i_{t_1}} + ... + e_{i_2}x^{i_2} + e_{i_1}x^{i_1}$$

where e_{i_l} is the magnitude of the lth error. Then, the syndrome S_1 evaluated at α can be written as

$$S_1 = e(\alpha) = e_{i_{t_1}}\alpha^{i_{t_1}} + ... + e_{i_2}\alpha^{i_2} + e_{i_1}\alpha^{i_1}.$$

Similarly, we can evaluate the other $2t - 1$ syndromes at $\alpha^2, ..., \alpha^{2t}$. For simplicity of notation, the error values e_{i_l} are denoted by Y_l and the error location numbers α^{i_l} by X_l.

Now, we can write the $2t$ as a set of simultaneous equations:

$$\begin{aligned} S_1 &= Y_{t_1}X_{t_1} + Y_{t_1-1}X_{t_1-1} + ... + Y_1X_1 \\ S_2 &= Y_{t_1}X_{t_1}^2 + Y_{t_1-1}X_{t_1-1}^2 + ... + Y_1X_1^2 \\ &\vdots \\ S_{2t} &= Y_{t_1}X_{t_1}^{2t} + Y_{t_1-1}X_{t_1-1}^{2t} + ... + Y_1X_1^{2t} \end{aligned} \tag{2.5}$$

The decoding problem has now been reduced to the problem of solving a system of nonlinear equations. This set of equations is too difficult to solve directly. Instead, a polynomial called locator polynomial is introduced. This polynomial that has t_1 nonzero coefficients is given by

$$\Lambda(x) = \Lambda_{t_1}x^{t_1} + \Lambda_{t_1-1}x^{t_1-1} + ... + \Lambda_1x + 1 \tag{2.6}$$

and defined to be the polynomial with zeros at the inverse error locations X_l^{-1} for $l = 1, ..., t_1$. That is,

$$\Lambda(x) = (1 - xX_1)(1 - xX_2)...(1 - xX_{t_1}). \tag{2.7}$$

The task now is to find the coefficients of $\Lambda(x)$. Once the coefficients are known, the error locations can be obtained by computing the zeros of $\Lambda(x)$.

Multiply the equations 2.6 and 2.7 each by $Y_lX_l^{j+t_1}$ and set $x = X_l^{-1}$. The second equation is clearly zero, and we have

$$Y_lX_l^{j+t_1}(\Lambda_{t_1}X_l^{-t_1} + \Lambda_{t_1-1}X_l^{-(t_1-1)} + ... + \Lambda_1X_l^{-1} + 1) = 0, \tag{2.8}$$

that gives

$$Y_l(\Lambda_{t_1}X_l^j + \Lambda_{t_1-1}X_l^{j+1} + ... + \Lambda_1 X_l^{j+t_1-1} + X_l^{j+t_1}) = 0 \tag{2.9}$$

Such an equation holds for each l and j. By Summing these equations for $l = 1$ to $l = t_1$, we have for each j

$$\sum_{l=1}^{t_1} Y_l(\Lambda_{t_1}X_l^j + \Lambda_{t_1-1}X_l^{j+1} + ... + \Lambda_1 X_l^{j+t_1-1} + X_l^{j+t_1}) = 0 \tag{2.10}$$

or

$$\Lambda_{t_1}\sum_{l=1}^{t_1} Y_l X_l^j + \Lambda_{t_1-1}\sum_{l=1}^{t_1} Y_l X_l^{j+1} + ... + \Lambda_1 \sum_{l=1}^{t_1} Y_l X_l^{j+t_1-1} + \sum_{l=1}^{t_1} Y_l X_l^{j+t_1} = 0, \tag{2.11}$$

where each individual sum represents a syndrome whose index is fixed by the power of X_l. Thus, equation 2.9 becomes

$$\Lambda_{t_1}S_j + \Lambda_{t_1-1}S_{j+1} + ... + \Lambda_1 S_{j+t_1-1} + S_{j+t_1} = 0 \tag{2.12}$$

and

$$S_{j+t_1} = -(\Lambda_{t_1}S_j + \Lambda_{t_1-1}S_{j+1} + ... + \Lambda_1 S_{j+t_1-1}), \tag{2.13}$$

for $1 \leq j \leq 2t - t_1$. Equation 2.13 can be written in general form of linear recursion

$$S_k = -\sum_{j=1}^{t_1} \Lambda_j S_{k-j} \ mod\ n \qquad k = t_1 + 1, ..., 2t_1. \tag{2.14}$$

Clearly, if we consider the more general case where t errors have occurred, the equation 2.14 becomes

$$S_k = -\sum_{j=1}^{t} \Lambda_j S_{k-j} \ mod\ n \qquad k = t + 1, ..., 2t. \tag{2.15}$$

Obviously, the number of errors in the received word is not known. The principle of decoding is to solve the linear recursion for Λ using the smallest value of t_1 [44]. This system of equations is always solvable for Λ and t_1. To solve it there is two ways: direct method and iterative method. In both methods, the decoding process consists of two principal steps: finding the error locations and finding their magnitudes.

2.5.3.1 Direct method

As a direct resolve for the linear recursion, Peterson [54] proposed an algorithm for finding the error locations by inverting a t_1 by t_1 matrix where t_1, as previously mentioned, is the smallest number ($t_1 \leq t$) of occurred errors. For finding the error magnitudes, the Gorenstein-Zierler algorithm [55] is applied.

The linear recursion of equation 2.15 can be written in matrix form:

$$\begin{bmatrix} S_1 & S_2 & ... & S_{t_1} \\ S_2 & S_3 & ... & S_{t_1+1} \\ \vdots & & & \vdots \\ S_{t_1} & S_{t_1+1} & ... & S_{2t_1-1} \end{bmatrix} \begin{bmatrix} \Lambda_{t_1} \\ \Lambda_{t_1-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{t_1+1} \\ -S_{t_1+2} \\ \vdots \\ -S_{2t_1} \end{bmatrix}.$$

Let us consider the matrix of syndromes

$$M_{t_1} = \begin{bmatrix} S_1 & S_2 & ... & S_{t_1} \\ S_2 & S_3 & ... & S_{t_1+1} \\ \vdots & & & \vdots \\ S_{t_1} & S_{t_1+1} & ... & S_{2t_1-1} \end{bmatrix}.$$

Peterson algorithm consists of the following steps:

1. Find the correct value of t_1 by computing the determinants M_t, M_{t-1}, etc. in succession and stopping when nonzero determinant is obtained. The actual number of errors that occurred is then known as t_1.
2. Invert M_{t_1} to compute $\Lambda(x)$
3. Finally, apply the Chien algorithm [56] to find the zeros of $\Lambda(x)$ and then the error locations.

To compute the error magnitudes, we consider the set of equations 2.5 defining the syndromes. In these equations, X_l are known and Y_l must be computed. Y_l for $l = 1, ..., t_1$ can be deduced from the first t_1 equations that can be solved if the determinant of the matrix of coefficients

$$X = \begin{bmatrix} X_1 & X_2 & ... & X_{t_1} \\ X_1^2 & X_2^2 & ... & X_{t_1}^2 \\ \vdots & & & \vdots \\ X_1^{t_1} & X_2^{t_1} & ... & X_{t_1}^{t_1} \end{bmatrix}$$

is nonzero. This is the Gorenstein-Zierler algorithm. But

$$det(X) = (X_1 \;\; X_2 ... X_{t_1}) det \begin{bmatrix} 1 & 1 & ... & 1 \\ X_1 & X_2 & ... & X_{t_1} \\ \vdots & & & \vdots \\ X_1^{t_1-1} & X_2^{t_1-1} & ... & X_{t_1}^{t_1-1} \end{bmatrix}.$$

This Vandermonde matrix does have a nonzero determinant if t_1 errors occur and therefore $(X_1, X_2, ..., X_{t_1})$ are nonzero and distinct. Then Y_l can be written as

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_{t_1} \end{bmatrix} = \begin{bmatrix} X_1 \dots & X_{t_1} \\ X_2 \dots & X_{t_1}^2 \\ \vdots & \vdots \\ X_1^{t_1} \dots & X_{t_1}^{t_1} \end{bmatrix}^{-1} \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{t_1} \end{bmatrix}.$$

For small t values, matrix inversion is reasonable; the number of multiplications necessary to invert a t by t matrix is proportional to t^3. However, when t is large, one should use a more efficient method that obviously will be more intricate but computationally much simpler. The solution is the iterative method.

2.5.3.2 Iterative method

The key step in the RS decoding process is the computation of the error locator polynomial $\Lambda(x)$ from the known $2t$ spectral components of the received words. Berlekamp [57] developed an elegant algorithm based on the language of polynomials which has been described in terms of shift registers by Massey [58]. This algorithm known as *Berlekamp-Massey algorithm* solves the problem by an iterative procedure that consists in designing the shortest linear recursion $(\Lambda(x), t_1)$ capable to produce the known sequence of syndromes. The algorithm consists of $2t$ iterations of computation. At each iteration, a candidate syndrome $\bar{S_k}$ is computed

$$\bar{S_k} = -\sum_{j=1}^{L_k-1} \Lambda_j^{k-1} S_{k-j} \qquad k = 1, ..., 2t,$$

where L_k is the length of linear recursion. The obtained syndrome $\bar{S_k}$ is subtracted from the desired (or known) syndrome S_k to get a quantity

$$\Delta_k = S_k - \bar{S_k} = S_k - \sum_{j=1}^{L_k-1} \Lambda_j^{k-1} S_{k-j}.$$

If Δ_k is zero, then the $(\Lambda(x), L_k)$ is the right linear recursion. Otherwise, $(\Lambda(x), L_k)$ should be modified. The main trick of Berlekamp-Massey algorithm is to use earlier iterations to compute a new minimum-length linear recursion $(\Lambda(x), L_k)$.

Berlekamp-Massey algorithm. In any field, let $S_1, S_2, ..., S_{2t}$ be given. With initial conditions $\Lambda^{(0)}(x) = 1$, $B^{(0)}(x) = 1$, and $L_0 = 0$, let the following set of equations for $k = 1, ..., 2t$ be used iteratively to compute $\Lambda^{(2t)}(x)$:

$$\Delta_k = \sum_{j=0}^{n-1} \Lambda_j^{k-1} S_{k-j},$$

$$L_k = \delta_k(k - L_{k-1}) + (1 - \delta_k)L_{k-1},$$
$$\Lambda^{(k)}(x) = \Lambda^{(k-1)}(x) - \Delta_k x B^{k-1}(x),$$
$$B^{(k)}(x) = \Delta_k^{-1}\delta_k \Lambda^{(k-1)}(x) + (1 - \delta_k) x B^{k-1}(x),$$

where $\delta_k = 1$ if both $\Delta_k \neq 0$ and $2L_{k-1} \leq k - 1$, and otherwise $\delta_k = 0$. Then $(\Lambda^{2t}(x), L_{2t})$ is a linear recursion of shortest length that produces $S_1, S_2, ..., S_{2t}$.

Once the error locator polynomial is computed, the next step is to compute the error values. There are two approaches to perform this last step.

The first approach is to continue the iterative computation. That is, after the $2t$ iterations of computation of $\Lambda(x)$ that produce the $2t$ syndromes, the remaining $n - 2t$ syndromes are produced by computing the discrepancy Δ_k for $k = 2t + 1$, ..., n and at each iteration compute $S_k = S_k - \Delta$. Fig. 2.4 shows a flow diagram for a decoder that computes the frequency-domain codeword $\mathbf{C}$ from the frequency received word $\mathbf{R}$.

As shown, a Fourier transform is placed at the front end of the decoder and an inverse Fourier transform is needed at the end to provide the time domain decoder. We note here that the decoder can be used with an encoder in the time domain, as shown in Fig. 2.5,

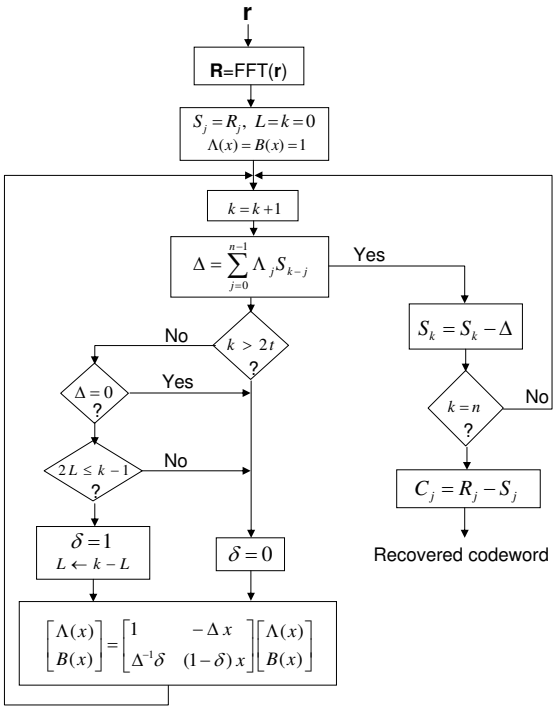

Figure 2.4: A frequency-domain decoder

or an encoder in the frequency domain, as shown in Fig. 2.6. If a time domain encoder is used, then the inverse Fourier transform of the corrected codeword spectrum $\mathbf{C}$ must be computed to obtain the time-domain codeword $\mathbf{c}$, from which the data is recovered. If instead, the encoder uses the data symbols in the frequency domain to specify the values of the spectrum, then the corrected spectrum gives the data symbols directly. That decoder does not compute an inverse Fourier transform. The decoder shown in Fig. 2.4 has a simple structure for $k > 2t$ but when a pipelined decoder is desired, a more attractive approach can be used.

The second approach is based on two elegant algorithms: Chien algorithm [56] to find the error locations and Forney algorithm [59] to compute the error values. Chien algorithm searches iteratively the roots of polynomial $\Lambda(x)$ that can specify the error locations. This algorithm tests the condition

$$\sum_{j=1}^{t} \Lambda_j \alpha^{ij} = 1 \qquad i = 1, 2, ..., n-1. \tag{2.16}$$

If this sum is 1, then the $(n-i)th$ component of the received word is erroneous. By rewriting equation 2.16 as

$$\sum_{j=0}^{t} \Lambda_j \alpha^{ij} = 0 \qquad i = 1, 2, ..., n-1, \tag{2.17}$$

where $\Lambda_0 = 1$. In fact, the n coefficients of equation 2.17 represent the frequency components of polynomial $\Lambda(x)$. Then, by computing the FFT of $\Lambda(x)$, the error locations

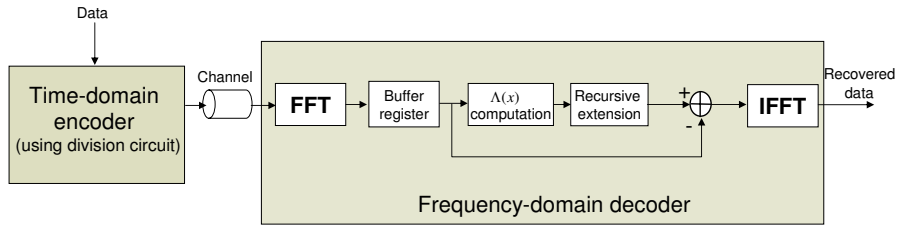

Figure 2.5: A time-domain encoder and frequency-domain decoder for RS codes

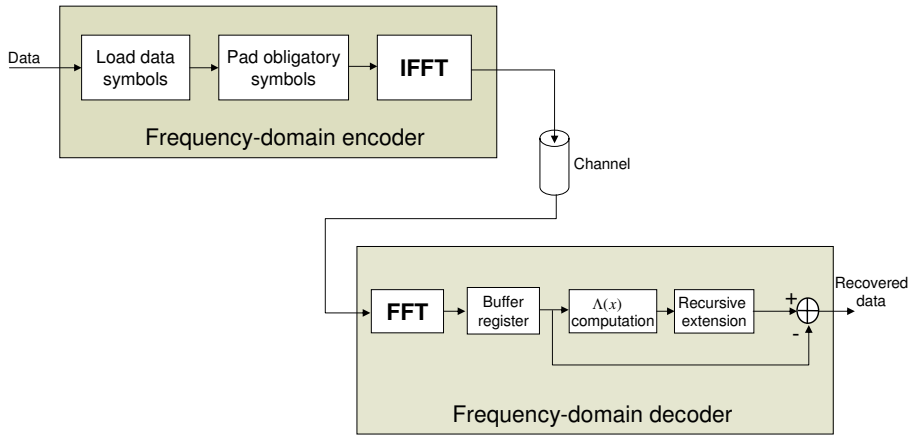

Figure 2.6: A frequency-domain encoder and frequency-domain decoder for RS codes

that correspond to the roots of $\Lambda(x)$ can be determined by testing the spectral components $\Lambda(x)$. That is, if $\Gamma_j = \sum_{j=0}^{t} \Lambda_j \alpha^{ij} = 0$, the $(n-i)th$ symbol is erroneous. This frequency reasoning is advantageous if an efficient algorithm to compute the $FFT(\Lambda(x))$ can be applied. Then, the important task of RS decoding *Chien search* can be performed with FFT.

The error values can be computed using Forney algorithm which evaluates a polynomial called *evaluator polynomial* ($\Omega(x)$) at the specified error locations. The polynomial $\Omega(x)$ can be computed using the polynomial multiplication

$$\Omega(x) = \Lambda(x)S(x) \quad mod(x^{2t+1}),$$

where

$$S(x) = \sum_{j=1}^{2t} S_j x^j.$$

The flow diagram of Fig. 2.7 shows the frequency decoder using Forney algorithm where $\Lambda'(x)$ is the derivative of $\Lambda(x)$. More details about the algorithm can be found in [44].

We note that there are other important algorithms known in the literature used in the decoding of RS codes. Among these algorithms, we find Sugiyama, Euclide, Gao and other

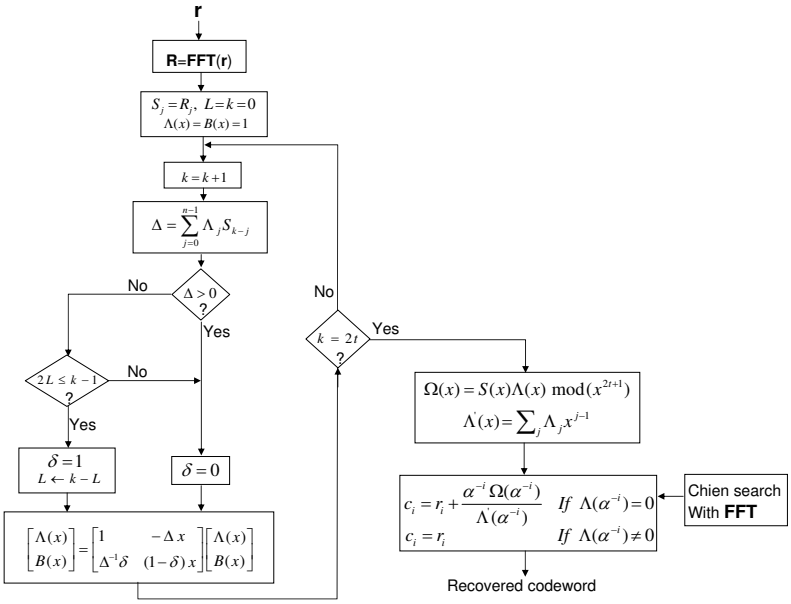

Figure 2.7: A frequency-domain decoder using the Forney algorithm

algorithms. The study of these various algorithms is not the subject of our work. The goal is to highlight the role of the FFT in the encoding and decoding processes by reviewing the most important algorithms adapted to the frequency processing of RS codes.

2.6 Comparison between time and frequency domain decoding of RS codes

In the previous section, the decoding procedure was stated in the language of spectral estimation. In this section, in order to compare the frequency to the time domain decoder, we describe the RS time domain decoding performed without any Fourier transform.

2.6.1 Time domain decoding of RS codes

Starting from the frequency processing, the idea is to replace the frequency domain variables by the time domain ones and then push the Berlekamp-Massey equations into the time domain. Let λ and b denote respectively the inverse Fourier transforms of the vectors $\Lambda(x)$ and $B(x)$ and replace the delay operator x with α^{-i}, and replace product terms with convolution terms. Replacement of the delay operator with α^{-i} is justified by the translation property of Fourier transforms; replacement of a product with a convolution is justified by the convolution theory. Blahut [60] introduced this notion of time-domain decoding and developed a universal time-domain RS decoder [61]. The following algorithm describes the decoding procedure.

Algorithm (Blahut algorithm). *Let $r = c + e$ be the received noisy RS codeword, where c is the transmitted word and e the error vector. With initial conditions $\lambda_i^{(0)} = 1$,*

$b_i^{(0)} = 1$ *for all* i *and* $L_0 = 0$, *let the following set of equations for* $r = 1, ..., 2t$ *be used iteratively to compute* $\lambda_i^{(2t)}$ *for* $i = 0, ..., n-1$:

$$\Delta_k = \sum_{i=0}^{n-1} \alpha^{ik} (\lambda_i^{k-1} r_i,$$

$$L_k = \delta_k (k - L_{k-1}) + (1 - \delta_k) L_{k-1},$$

$$\lambda_i^{(k)} = \lambda_i^{(k-1)} - \Delta_k \alpha^{-i} b_i^{k-1},$$

$$b_i^{(k)} = \Delta_k^{-1} \delta_k \lambda_i^{(k-1)} + (1 - \delta_k) \alpha^{-i} b_i^{k-1},$$

where $\delta_k = 1$ if both $\Delta_k \neq 0$ and $2L_{k-1} \leq k - 1$, and otherwise $\delta_k = 0$. Then $\lambda_i^{2t} = 0$ if and only if $e_i \neq 0$ and the equation

$$c_i = r_i - s_i,$$

produce the recovered codeword. The vector s_i is initialized by the received codeword and updated at each iteration as shown in the Fig. 2.8.

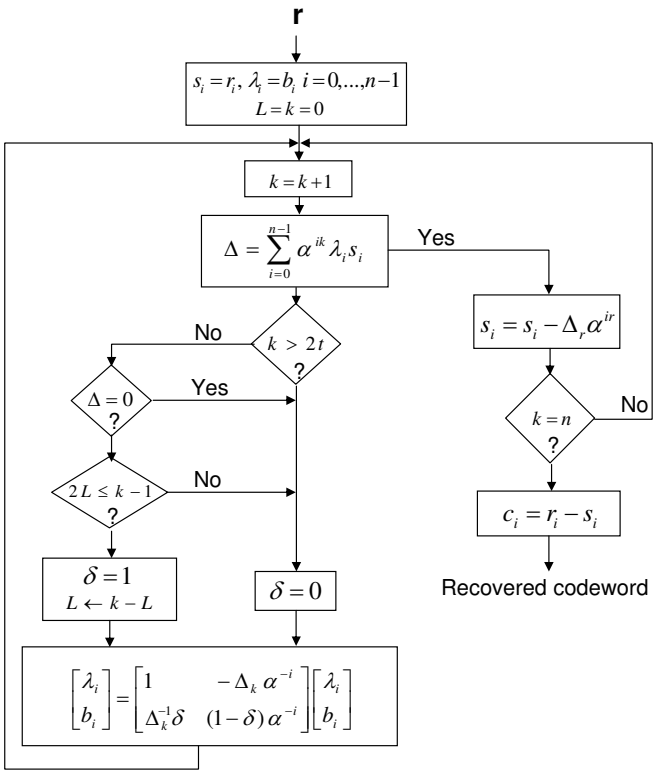

Figure 2.8: A time-domain RS decoder

2.6.2 Comparison

Let us now consider the comparison between these two ways of decoding. For the frequency domain approach, the matrix update, i.e. update of $\Lambda(x)$ and $B(x)$, requires at most $2t$ multiplications per iteration and the calculation of Δ_k requires at most t multiplications per iteration. There are $2t$ iterations and so no more than $6t^2$ multiplications are required to compute the error locator polynomial. In the other hand, the time domain approach works directly on the raw data word as received without any transform. The frequency-domain vectors of Berlekamp-Massey algorithm of length t are replaced by time-domain vectors of length n. Then the time domain decoder has a computation complexity proportional to n^2 while the one of the frequency domain decoder is proportional to t^2.

The time domain decoder can be attractive because there is no syndrome computation or Chien search. This means that the decoder has a very simple structure, but the penalty is a longer running time. As for the frequency domain decoder, its structure is more complex but this drawback is largely compensated by its major advantage that is a shorter running time. For this latest reason, the frequency domain decoders are more attractive and have found their use in various applications and specifically in applications that require high throughput rates.

2.7 Extended RS codes using Fourier transform

In this section, we will present the extended RS codes. The objective is just to describe the principle of the frequency processing of these codes and show that the new length of an extended RS code does not affect the length of the Fourier transform to be used in the encoding and decoding processes.

Generally, in some applications, codewords of RS codes need to be extended to satisfy certain specific requirements. To lengthen an RS code, there is an orderly way to append up to two extra symbols to the codewords. When a code is obtained by appending one or both of the extra symbols to each codeword, that code is called an extended RS code. If only one extension symbol is appended, the code is called a singly extended RS code. If two symbols are appended, the code is called a doubly extended RS code.

Each appended symbol can be viewed either as a data symbol or as a check symbol, that is, it may be used either to expand a code by increasing the rate, or to lengthen a code by increasing the minimum distance. In this section, we consider the singly extended RS code and we shall briefly describe its encoding and decoding processes.

Encoding in frequency-domain of the singly extended RS code is as follows. Let $g(x)$ be the generator polynomial with zeros at $\alpha^2, \alpha^3 ..., \alpha^{2t-1}$ and choose a block of $d-2$ consecutive components, indexed by $2, 3, ..., 2t-1$, of the spectrum to be zero. The remaining components of the spectrum are loaded by arbitrary data symbols from $GF(q)$. The value of the spectrum on the edge of the block of check frequencies is also arbitrary symbol from $GF(q)$ and this symbol is appended to the edge of the codeword as c_e. A $(q-1)$ inverse Fourier transform preceded by c_e produce the time domain codeword. The new codeword is composed of q symbols. Since the appended symbol does not undergo any transform, the transform length remains $q-1$. Fig. 2.9 shows such a frequency encoder.

To encode an (n, k) singly extended RS code in the time domain, use $g(x)$, defined above, to encode k data symbols into $c_0, c_1, ..., c_{n'-1}$ where $n' = n-1$. Then the extension symbol is defined by

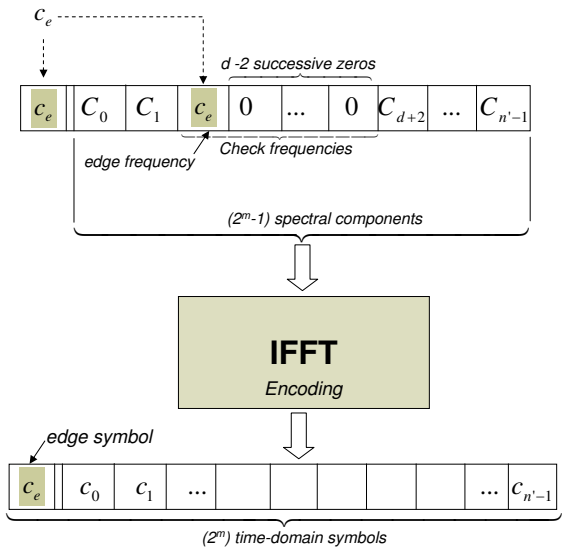

Figure 2.9: Encoding an extended RS code in the frequency domain

$$c_e = \sum_{i=0}^{n'-1} c_i \alpha^i,$$

and it is attached to $c_0, c_1, ..., c_{n'-1}$ where $n' = n - 1$ to produce the lengthening code word $c_e, c_0, c_1, ..., c_{n'-1}$. The minimum distance of the extended code is $d_{min} = n-k+1 = q - k + 1$.

As for the decoding, any encoder for a RS code can be used to decode an extended RS code.

2.8 Discussion

In this chapter, until now, we have discussed the important and central role the frequency processing can play in the encoding and decoding processes of RS codes. Compared to the time domain processing, the frequency domain processing presents important advantages in terms of running time and efficiency of hardware and software implementation of principal algorithms. But, the transforms used in that frequency domain processing are defined over $GF(2^m)$ and they have the following two principal characteristics:

1. The transform length is equal to $2^m - 1$, where m is an integer.

2. The arithmetic operations are done modulo 2.

Indeed, our study aims to insert the complex FFT operator in the channel coding function. However, the first characteristic which is the transform length of the finite field transform over $GF(2^m)$ does not match the one of the complex FFT defined over the complex field $\mathbb{C}$, which is of the form 2^m. This characteristic is a strong constraint that challenges the adaptation or the combination of the $GF(2^m)$ FFT structure with

the complex FFT one, since the most efficient algorithms for the FFT computations are applied to transforms of length 2^m. Under this strong constraint, we thought to seek out a transform satisfying the complex FFT criteria while keeping in mind to include the classical finite field transform, used in the RS coding defined over $GF(2^m)$, in the intended common and reconfigurable FFT structure. Next section is dedicated to find the appropriate transforms.

2.9 Fermat Number Transform and $GF(F_t)$

The strong constraint discussed in the previous section, has led us to seek other class of transforms which in turn will lead to other and specific class of RS codes. Our research on the state of the art of finite field transform and RS codes led us to spot specific class of transforms and get out the corresponding class of RS codes. These specific finite field transforms as well as the corresponding RS codes are defined over $GF(F_t)$ where F_t is a Fermat prime number. These transforms called Fermat Number Transforms (FNT) constitute a subclass of the Number Theoretic Transforms.

2.9.1 A brief history of Number Theoretic Transform

The definition of the Number Theoretic Transforms (NTT) appeared in 1971 when Pollard [45] discussed transforms having the cyclic convolution property in a finite ring of integers. Later in 1972, Rader [62] [63] proposed transforms over rings of integers modulo both Mersenne and Fermat numbers that can be used to compute error-free-convolutions of real integer sequences. Rader first proposed the application of such transforms to digital signal processing and showed that these transforms could be calculated using only additions and bit shifting. He also showed the word-length constraint and suggested two-dimensional transform as a possible relaxation of that constraint. Agarwal and Burrus [64] extended Rader's Fermat Number transform theory by lengthening the transform size and showed that the usual FFT algorithm can be used to calculate the Fermat transforms. In 1976, Justesen [65] proposed that transforms over the finite field $GF(F_t)$ can be used to define RS codes and improve the decoding efficiency of these codes.

Starting from Justesen proposition, the rest of this chapter will constitute the investigation of this specific class of Galois field $GF(F_t)$ where the construction as well as the encoding and decoding of RS codes will be examined. In turn, the treatment of these RS codes over $GF(F_t)$ in frequency domain will constitute the window through which we look at the channel coding as being one of the communication tasks that can be performed using the FFT operator in the Software Radio context.

2.9.2 Principal characteristics of FNT

Fermat Number Transforms are one of the most promising Number Theoretic Transforms where the modulus is chosen to be a Fermat number:

$$F_t = 2^{2^t} + 1,$$

and F_t is called the tth Fermat number. Originally, Fermat [66] conjectured in 1650 that these numbers were all primes but at present, only F_t for $t = 0, 1, 2, 3, 4$ are known

as prime numbers. For $t \geq 5$ the Fermat numbers are composite. The first five Fermat prime numbers are: F_0, F_1, F_2, F_3, F_4 equal to 3, 5, 17, 257 and 65537 respectively. For $t = 5$, $F_5 = 4294967297 = (641) * (6700417)$.

FNT have a transform length power of 2, but this transform length is directly related to the element α chosen as a root of unity of order n, where n is the least positive integer such that

$$\alpha^n = 1.$$

Let us discuss the relationship between the transform length and the generator element (or the nth root of unity) α. We begin with the integer 2. The integer 2 is an α of order $n = 2^{t+1}$ (i.e. $2^{2^{t+1}} = 1$) and the corresponding transform length is $n = 2^{t+1}$. In this case the FNT can be computed very efficiently and is called the Rader transform [63]. As the author showed, the arithmetic used to perform these transforms requires only integer additions and circular shifts since the multiplications by powers of two can be performed by simple bit rotations. This is the primary advantage of the Rader transform.

To lengthen the transform size, Agarwal and Burrus [67] proposed to use the generator $\alpha = \sqrt{2}$ for the transform rather than $\alpha = 2$. If $\sqrt{2}$ is used as the generator of the transform, the transform can be computed using a very fast algorithm known as FFT with as many as 2^{n+2} points of integer data. In order to treat longer FNTs, Reed and Truong [68] showed that $\sqrt[8]{2}$ is an element of order 2^{t+4} in $GF(F_t)$ and it can be used to define FNT of as many as 2^{t+4} data symbols. Moreover, the authors showed that these transforms can be used to decode RS codes with codewords of as many as 2^{t+4} symbols.

To better understand the character of these prime moduli, we consider the example for $F_2 = 17$. Table 2.1 shows the various values of α and the corresponding transform lengths.

Table 2.1: The powers of α and the corresponding transform lengths over $GF(F_2)$

n=0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1^n=1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^n=1	2	4	8	16	15	13	9	1	2	4	8	16	15	13	9	1
3^n=1	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1
4^n=1	4	16	13	1	4	16	13	1	4	16	13	1	4	16	13	1
6^n=1	6	2	12	4	7	8	14	16	11	15	5	13	10	9	3	1

As we see in Table 2.1, integer 2 is of order 8 and 4 is of order 4. For the integers 3 and 6, $n = F_t - 1 = 16$ is the least positive integer for which 3^{16}, and 6^{16} are equal to 1 (mod 17). Note that $6 = \sqrt{2}$ in the sense that $6^2 = 2$ (mod 17) . Then, 3 and 6 are primitive roots of $GF(17)$ and their successive powers for $n = 0$ to 15 allows to generate all the elements of $GF(17) = \{0, 1, 2, ..., 2^{2^t}\}$. Therefore, by using one of the primitive roots, one can reach the largest possible length of FNT.

Having the complex FFT implementation issues in mind, the goal is to implement the finite field transform FNT using the hardware resources available in the complex FFT circuit. However, the FNT with the primitive element $\alpha = 3$ seems to be an appropriate choice for the following reasons. Firstly, if $\alpha = 3$ is taken as the generator of the trans-

form, the maximum possible transform length is attained and the structure of the FNT is similar to the complex FFT structure. Secondly, the arithmetic operations required for the FNT computation are operations modulo F_t. Thus, these operations can be computed as ordinary real operations using the complex operators (multiplier, adder and subtracter) implemented in the FFT and then a modulo F_t reduction returns the desired result. Therefore, by redesigning the operators constituting the FFT circuit, we can compute a finite field transform such as the FNT. This way leads to an optimal exploitation of the arithmetic resources which is exactly what the concept of common operator is about. This is the corner stone of our work where a prototype hardware of reconfigurable operator able to support FFT and FNT computation is discussed in the next chapter.

Our problematic study is then positioned. Let us now consider the specific class of RS codes. According to the above discussion, the RS codes that will be considered in the first stage of our study are the RS codes defined over $GF(F_t)$ where the FNT will be efficiently implicated. In the next section, we will study the encoding and decoding processes of these codes as well as their performance in term of Bit Error Rate (BER) and Frame Error Rate (FER).

2.10 Fermat transform-based RS codes

In this section we rediscover the RS codes defined over $GF(F_t)$ and we show that these codes have almost the same performances in terms of BER compared to the classical RS codes defined over $GF(2^m)$. We note here that the RS codes defined over $GF(F_t)$ was recommended for Spacecraft communication [69] where the RS(256,224) over GF(257) was studied to be used together with a convolutional code. We suppose that the reason for which these specific RS codes are not so widespread compared to classical RS codes lies in the arithmetical structure of the Galois field over which they are defined. That means, the arithmetical structure of $GF(2^m)$ is more simple than that of $GF(F_t)$.

In this study, we revive these codes for the main motivation that their hardware implementation can be partially performed with the aid of pre-implemented FFT where the building of a SWR system is considered.

Let us discuss the characteristics of $GF(F_t)$. In fact, as shown in previous section, this finite field can be represented as the set of numbers 0, 1 , 2, ..., 2^{2^t} with addition (or subtraction), multiplication and division modulo (F_t). It should be pointed out that a word of length 2^t bits is required to represent the elements of $GF(F_t)$ and an extra bit is needed to represent the last number 2^{2^t}. This symbol in $GF(F_t)$ cannot be represented easily as a 2^t-bit word. To remedy this, there are two solutions: the first one assumes the information symbols are restricted in the range from 0 to $2^{2^t}-1$ and, if the symbols 2^{2^t} occur as a parity check symbol, this value is deliberately changed to 0 and an error is committed. We can suppose that the decoder will correct such an error automatically. The second approach consists in representing the information symbols by 2^t bits and the parity check symbols by 2^t+1 bits. An obvious third solution that consists in representing each one of the elements of $GF(F_t)$ by 2^t+1 bits should not be taken into account because it decreases the useful data rate. The choice between the first two solutions should be done according to the simulation results of these RS codes.

2.10.1 Encoding of RS codes defined over $GF(F_t)$

In section 2.5.1, we have described in details the two ways of RS encoding over $GF(2^n)$ in time and frequency domains. The same principles can be applied to the RS codes defined over $GF(F_t)$ with the difference that the arithmetic operations are done modulo F_t rather than modulo 2. The encoding in the time domain uses a generator polynomial to calculate the parity check symbols. In the frequency domain, the encoding consists in constraining some specified spectral components to zero and filling the unconstrained components of the spectrum with data symbols of $GF(F_t)$. Then, the inverse Fermat transform generates a nonsystematic codeword.

Now let us discuss which one of the two encoding ways is more suitable to obtain the best performances in terms of BER. In the frequency encoding case, after filling the constrained and unconstrained spectrum components, the symbols of the codeword are generated by an inverse Fermat transform. As a consequence, each of the codeword symbols may take the value 2^{2^t}. In the time domain encoding case, the information symbols are 2^t-bit symbols and a polynomial division, using the generator polynomial, produces a systematic code word composed of the original information symbols and of the parity check symbols. The probability to obtain the symbol "2^{2^t}" is restricted to the parity check symbols. Thus, the probability that the symbol 2^{2^t} occurs during the time domain encoding, is quite inferior than that of the frequency domain encoding. For this reason, the time domain solution seems to be most appropriate for encoding the RS codes defined over $GF(F_t)$. Such a decision will be justified in the simulation results. The information symbols will be then represented by 2^t-bits and for the parity check symbols, there are two possibilities. It is either to add an extra bit for each symbol or to change 2^{2^t} to zero when it occurs. The results of simulations in the subsection 2.10.3 specify the most suitable way to represent such symbols.

2.10.2 Decoding of RS codes defined over $GF(F_t)$

In section 2.5.3, we have described the different algorithms used for the decoding of RS codes defined over $GF(2^m)$. These algorithms can be applied to decode the RS codes defined over $GF(F_t)$. However, the highly used RS decoding procedure in actual applications is that composed of the Berlekamp-Massey algorithm, Chien algorithm and Forney algorithm (see Fig. 2.7). In this section, we consider this procedure and we will show the efficient role the FNT can play by performing the most time-consuming steps of the decoding process. To better illustrate the task, we divide the decoding process into three phases as illustrated in Fig. 2.10. The first step consists of a Fermat transform to perform the syndrome computation, and the second performs the spectral analysis using the Berlekamp-Massey algorithm and some polynomial computations. The third step also consists of a Fermat transform to perform the Chien search followed by Forney algorithm to correct the erroneous symbols.

As illustrated in Fig. 2.10, each of the syndrome computation and Chien search takes n clock cycles, where n is the block length of the code, which is the most long time in the decoding process. The suggested FNT to be implemented to replace the corresponding circuit of each of the syndrome computation and Chien search has a transform length equal to power of 2. Thus, using the fast algorithms to compute the transforms, FNT can be implemented into $log\ n$ stages where each stage is composed of $n/2$ computational

units. By computational unit, we mean the "butterfly" consisting of a multiplier, adder and subtracter. This efficient implementation of the FNT permits to reduce each n cycles to $log\ n$ cycles. Even tough this seems to be ideally, this suggestion is realistic and promising as it leads to a discussion of building a very fast RS decoder. If a fast FNT seems affordable, this opens another discussion about the use of the FNT as a common operator while it is capable to perform the required tasks at least two times faster than the existing solutions. Since these theoretical approaches are directly related to the practical implementation of the FNT circuit, the ideas will be more clear in the next chapter where the FNT and FFT implementations are considered.

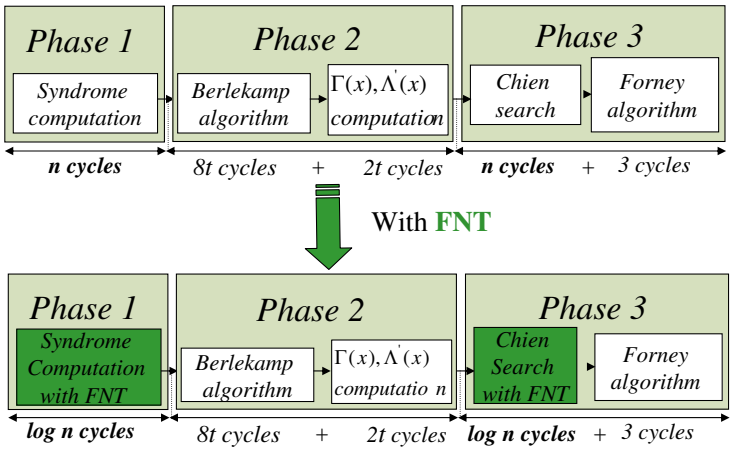

Figure 2.10: The three phases of RS decoding process over $GF(F_t)$

2.10.3 Performances comparison between RS over $GF(F_t)$ and RS over $GF(2^m)$

In this subsection, we give simulation results in terms of BER and FER vs $\frac{E_b}{N_0}$ (E_b is the energy per bit and N_0 the power spectral density of the channel noise) for different RS coding schemes with correcting capacity t_c equal to 2. In this simulation, random data with a BPSK modulation are transmitted over an Additive White Gaussian Noise (AWGN) channel. In Fig. 2.11 and Fig. 2.12, we have a set of five BER and FER curves: (i) uncoded; (ii) RS(16,12) over $GF(17)$ with frequency encoding and frequency decoding using the Berlekamp-Massey algorithm and the recursive extension, the codeword symbols are represented by 5 bits each; (iii) RS(16,12) over $GF(17)$ with time domain encoding and frequency domain decoding using the algorithms indicated in Figure 2.10, the information symbols and the parity check symbols are represented by 4 bits (4-4); (iv) it is the same case of (iii) with the only difference that each parity check symbol is transmitted over 5 bits (5-4); (v) RS(15,11) over $GF(16)$.

We start this analysis with curve (ii) of Fig. 2.11. In this case, for low $\frac{E_b}{N_0}$ values, the BER is high compared to other coding schemes. This phenomena can be explained by the following: firstly, the loss of useful data rate by transmitting each symbol over $2^t + 1$

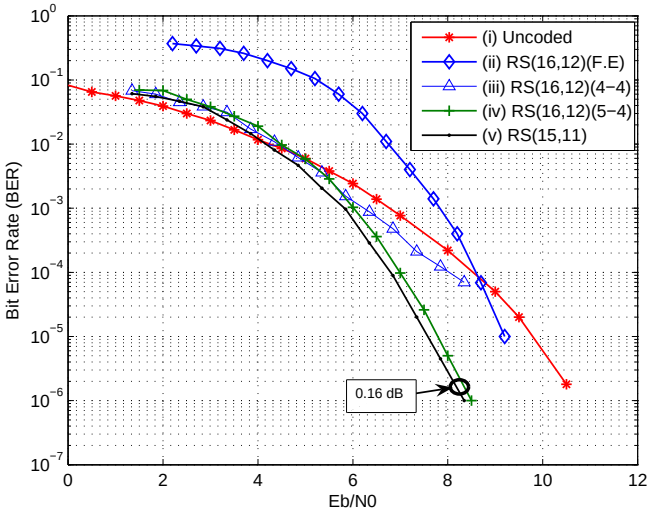

Figure 2.11: Performances of RS(16,12) over $GF(17)$ and RS(15,11) over $GF(16)$ with BPSK modulation on AWGN channel. (F.E: Frequency Encoding))

bits increases the channel error probability and consequently the BER. Secondly, if the number of erroneous symbols is higher than the correcting capacity, the decoding fails and since the symbols of code words are generated by the Fermat transform, the symbols will be all erroneous resulting in an high BER. Curve (iii) also indicates a high BER due to the fact that the symbol $2^{2^t} = 16$ is set to zero knowing that its probability occurrence is not equal to zero. We have calculated this probability for different $\frac{E_b}{N_0}$ values to be around 5.8 %, which in turn degrades the code performances. This degradation is clearly illustrated in Fig. 2.12 where we observe that at high (E_b/N_0), the performances in terms of FER of curve (iii) worsen as compared to curves (ii). This can be explain by the fact that the probability of occurrence of symbol 2^{2^t} (estimated to be around 5.8 %) and that does not depend on the energy of signal, deliberately decreases the code's error correcting capability.

Now, let us consider the two curves (iv) and (v) of Fig. 2.11. To solve the problem of the symbol "2^{2^t}", we have transmitted each information symbol over 2^t bits and each parity check symbol over 2^t+1 bits. In this way, the symbol "2^{2^t}" is correctly represented. Fig. 2.11 shows that the performances of the RS(16,12) defined over $GF(17)$, using the time domain encoding and the frequency domain decoding according to the algorithms presented in Fig. 2.10, has similar performances as RS(15,11) defined over $GF(16)$.

The asymptotic difference is 0.16 dB and can be expressed by the expression (in dB):

$$\Delta = 10log(1/R_1) - 10log(1/R_2), \tag{2.18}$$

where R_1 and R_2 are the code rates of RS(15,11) and RS(16,12) respectively. Here, $\Delta = 10log(15/11) - 10log(68/48) = 0.16$ dB. This difference decreases with the code length. To justify this, let us consider the two RS codes: RS(255,223) over GF(256) and RS(256,224) over GF(257), where the codes of each pair have the same error correcting

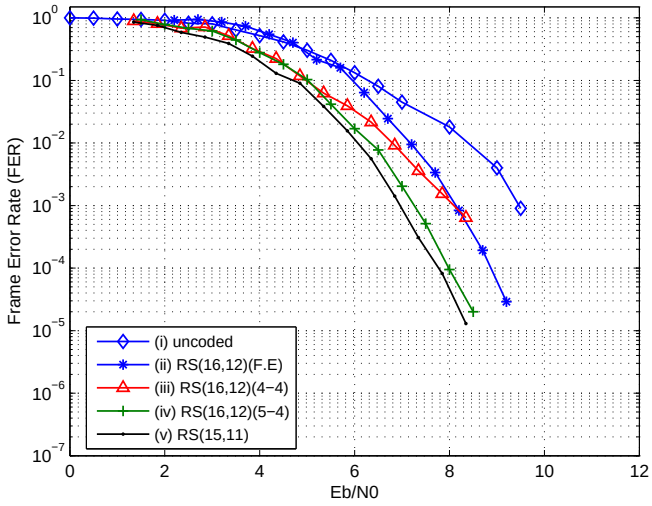

Figure 2.12: FER of RS(16,12) over $GF(17)$ and RS(15,11) over $GF(16)$ with BPSK modulation on AWGN channel.

capability (t_c). The associated code rates R_1 and R_2 are equal to 0.87 and 0.86 respectively. Applying equation 2.18, Δ in this case is equal to 0.05 dB. This drop of Δ can be explained also by evaluating, for each RS code, the channel error probability

$$P_i = \frac{1}{2} erfc\sqrt{R_i \frac{Eb}{N_0}} \ , \quad i = 1, 2. \tag{2.19}$$

Figure 2.13 shows the channel error probability of each RS code. As we remark, the difference between the two curves of channel error probability in the case of RS(255,223) and RS(256,224) is very low compared to that of the RS(15,11) and RS(16,12).

Theoretically, the RS codes over $GF(F_t)$ have the same error correcting capability, the small difference in terms of BER is due to the difference between the channel error probabilities. In turn, the channel error probability varies according to the code rate. The lower the difference between the code rates, the lower the difference between the channel error probabilities. Consequently, for long RS codes the difference in term of BER becomes negligible.

2.11 Conclusions: towards the reconfigurable FFT operator DMFFT

In order to maximize the reusability of the FFT operator we proceeded in the sense of building a flexible FFT operator able to support two different kinds of computations in two different environments. Starting from the complex and classical structure of the FFT dedicated to perform OFDM modulation, frequency equalization ... etc, the idea is to gracefully exploit this complex structure with the included arithmetic resources to perform a finite field transform able to be used in the channel coding. For this, we sought

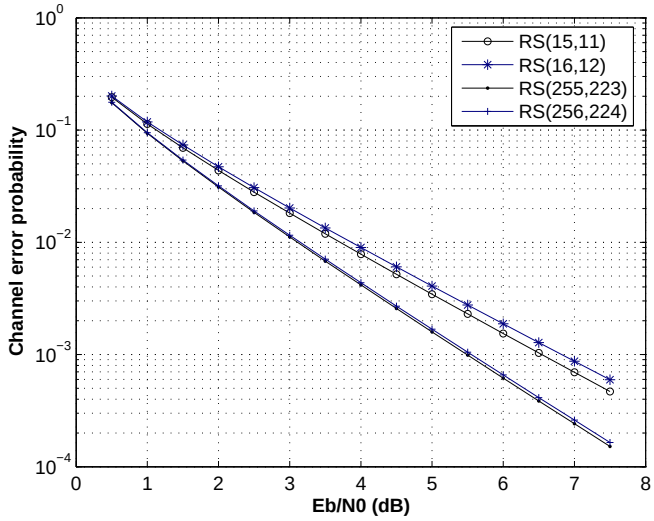

Figure 2.13: The channel error probability for different RS codes.

a finite field transform that satisfies the complex FFT criteria, such as the FNT. We then opt to design an operator capable to perform the FFT and FNT. This operator will be called "DMFFT" (Dual Mode FFT). By dual mode we mean complex mode (over complex field $\mathcal{C}$) and finite mode (over finite $GF(F_t)$). In this way, we can call the DMFFT as a common and reconfigurable operator. Common means to carry out two different tasks in two different contexts and reconfigurable means that the operator's hardware configuration can change to match the requirements of each functioning mode.

Next chapter will be dedicated to describe in details the hardware design and FPGA implementation of the DMFFT. In chapter 4, we will discuss some opportunities to make the DMFFT a triple mode operator by including the $GF(2^m)$ transform functionality among the functionalities that can be provided by the intended triple mode operator.

Chapter 3

Architecture of the DMFFT operator

Contents

In the previous chapter we described the efficient use of the FNT in the decoding process of the RS codes defined over $GF(F_t)$ and we discussed the possibility for designing it onto the FFT architecture. In this chapter we shall turn our attention to the implementation of the global operator "DMFFT" which is able to produce the FFT and FNT functionalities. This implementation will be based on the implementation techniques of the FFT. FFT denotes, throughout this chapter, the complex Fourier transform.

In general, there are many different ways of implementing the FFT algorithms. Most of the research to date for the implementation of these algorithms have been performed using general purpose processors [70], Digital Signal Processors (DSPs) [71] [72] and dedicated

FFT processors [73]. However, thanks to their high computation capacity at high performance and economical price, Field Programmable Gate Arrays (FPGAs) have become a viable solution for performing computationally intensive tasks such as FFT. In this sense, applications for custom chips and programmable DSP devices have been tackled, such as the FFT Megacore function [74] [75].

In the present work, the target hardware for the implementation and evaluation of the proposed architectures is the Altera's FPGA devices using the traditional hardware description language VHDL. We note that another mapping language has been widely used in the literature such as C++ language, System-C language [76] [77] and Handel-C language [78].

This chapter begins with the basics of FFT algorithms where we select the algorithm which will be implemented. In Section 3.2, we describe the architectures of the proposed reconfigurable arithmetic operators and the architecture of the heart of the FFT algorithm known as the "butterfly" operation. Section 3.3 provides the architecture of the reconfigurable DMFFT operator. In Section 3.4, the FPGA-based implementation complexity study of the proposed operators as well as for the DMFFT is given. A conclusion ends the chapter.

3.1 Fast Fourier Transform algorithms

We recall that the Discrete Fourier Transform (DFT) of an N-point discrete-time complex sequence $f(n)$, indexed by $n = 0, 1, ..., N-1$, is defined by

$$F(k) = \sum_{n=0}^{N-1} f(n) W_N^{kn}, \qquad k = 0, 1, ..., N-1 \tag{3.1}$$

where $W_N^{kn} = e^{-j2\pi/N}$ and $j = \sqrt{-1}$. W_N^{kn} is referred as the *twiddle factor*.

As well known, the complexity of the DFT direct computation can be significantly reduced from N^2 to $Nlog_2N$ by using fast algorithms that use a nested decomposition of the summation in equations to exploit various symmetries in the complex multiplications. One of the most famous algorithms is the Cooley-Tukey radix-r which recursively divides the input sequence into N/r sequences of length r and requires log_rN stages of computation. Each stage of the decomposition typically shares the same hardware, with the data being read from memory, passed through the FFT unit and written back to memory. Each pass through the FFT unit is required to be performed log_rN times. Popular choices of the radix are r=2, 4 and 16. Increasing the radix of the decomposition leads to a reduction in the number of passes required through the FFT unit at the expense of device resources. An elaborate description of various FFT algorithms can be found in [79].

In general, each algorithm can be represented either as Decimation In Time (DIT) or Decimation In Frequency (DIF). For Cooley-Tukey radix-2 algorithm, the decimation-in-time (DIT) recursively partitions the DFT into two half-length DFTs of the even-indexed and odd-indexed time samples

$$F(k) = \sum_{n=0}^{N/2-1} f(2n) W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} f(2n+1) W_{N/2}^{kn}, \qquad k = 0, 1, ..., N/2-1 \tag{3.2}$$

and

$$F(N/2+k) = \sum_{n=0}^{N/2-1} f(2n)W_{N/2}^{kn} - W_N^k \sum_{n=0}^{N/2-1} f(2n+1)W_{N/2}^{kn}, \qquad k = 0, 1, ..., N/2-1. \quad (3.3)$$

The radix-4 algorithm can be obtained by decomposing equation 3.2 and 3.3 into X_k, $X_{N/4+k}$, $X_{N/2+k}$ and $X_{3N/4+k}$, $k = 0, 1, ..., N/4-1$.

To further improve the computational complexity of the radix-r algorithms, an algorithm called split-radix algorithm was proposed [80]. The basic idea is to mix two different radixes: radix-2 for the even part of the transform and radix 4 for the odd part. The number of required real additions and multiplications is usually used to compare the efficiency of different FFT algorithms. In terms of the multiplicative comparison, the split-radix FFT is computationally superior to all the other algorithms. However, the split-radix FFT is inherently irregular because radix-2 stages are used for the even-half operations and radix-4 stages are used for the odd-half operations. This irregularity leads to an unbalanced delay of the pipeline path.

An objective choice for the best FFT algorithm can not be made without knowing the constraints imposed by the environment in which it has to operate under. The main criteria for choosing the most suitable algorithm other than the amount of required arithmetic operations (costs) is the regularity of structure. Several other criteria (e.g. latency, throughput, scalability, control) also play a major role in choosing a particular FFT algorithm. The search of the best FFT algorithm does not make the subject of our study. The objective is to test the feasibility of the reconfigurable FFT operator (DMFFT) and then evaluate its performances. For this, we have chosen radix-2 FFT implementation for our system because it has advantages in terms of regularity of hardware, ease of computation and number of processing elements.

Obviously, for a given transform length N power of 2 (or power of 4), the algorithm chosen to be applied to perform FFT should be valid to perform the FNT. Indeed, since the symmetry and periodicity properties

$$\alpha^{k+N} = \alpha^k, \qquad \alpha^{k+N/2} = -\alpha^k \quad (3.4)$$

are accomplished, every radix-r algorithm applied to FFT can be applied to the FNT.

Fig. 3.1 shows the traditional 64-point FFT dataflow diagram. What we need to do here, is to manipulate this circuit in a way to perform the FNT. The heart of this circuit known as the "butterfly" operation is the element which will be mainly concerned in the manipulation. Here, manipulation means the reconfiguration of the operators constituting the butterfly as well as the connection between those operators. The switching from FFT mode to FNT mode should be accompanied by the replacement of the twiddle factor W by the primitive element α of the given Galois field. Fig. 3.2 shows the butterfly structure with the two operating modes. The internal architecture of this butterfly as well as the design of modular arithmetic operators will be discussed in details in the next section.

3.2 The FFT-like butterfly architecture

The FFT-like butterfly architecture as shown in Fig. 3.2 consists of three arithmetic operators: multiplier, adder and subtracter. In the FFT mode these operators process

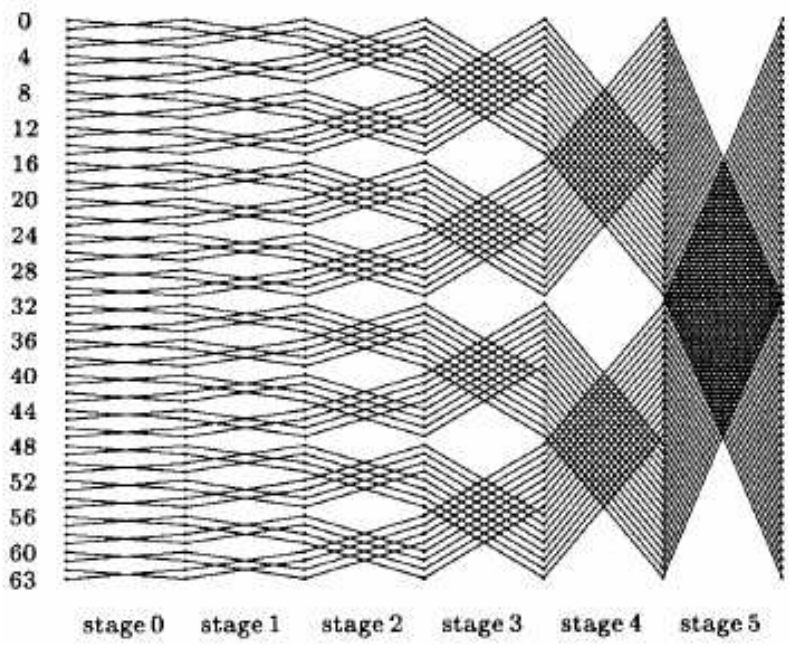

Figure 3.1: A traditional FFT dataflow diagram

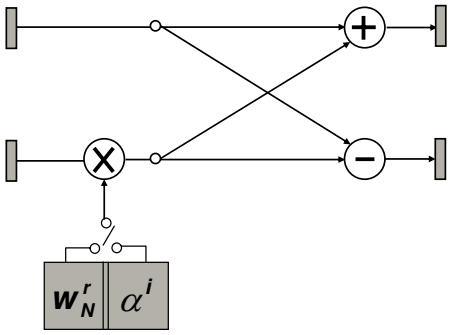

Figure 3.2: The FFT/FNT butterfly

complex data by performing complex multiplications and additions. In the FNT mode, the data is defined over finite field that is $GF(F_t)$ and the operations performing the FNT computations are needed to be done modulo F_t. So, these arithmetic operators should be redesigned to be able to support complex and modular operations. The realization of reconfigurable operators (adder, subtracter and multiplier) is described in details in the following subsections.

3.2.1 The reconfigurable adder

By using the logic resources employed in the complex adder used in the FFT-like butterfly we will implement a modulo F_t adder. Modulo arithmetic has been used in various

applications. The most important application is in Residue Number Systems (RNS) [81] [82]. First, let us describe the basic principles and circuit structures used for the ordinary addition of binary numbers to discuss afterwards the different ways to realize modular adders.

3.2.1.1 Foundations

Considering binary addition, one can find several architectures that offer a variety of time delay and area complexity [83].

- **Carry Propagate Adders (CPA):** A Carry Propagate Adder (CPA) adds two n-bit operands $A = (a_{n-1}, a_{n-2}, ..., a_0)$ and $B = (b_{n-1}, b_{n-2}, ..., b_0)$ and an optional carry-in c_{in} by performing carry propagation. The result is an irredundant $(n+1)$-bit number consisting of the n-bit sum $S = (s_{n-1}, s_{n-2}, ..., s_0)$ and a carry-out c_{out}. The advantage of this adder is that it offers a small design but the penalty is that the computation time grows linearly with the operand wordlength due to the serial carry-propagation.

- **Carry Save Adders (CSA):** The Carry Save Adder (CSA) avoids carry propagation by treating the intermediate carries as output instead of advancing them to the next higher bit position, thus saving the carries for later propagation. The operands are represented with the numbers $a_i \in \{0, 1, 2\}$ using two digits such as: $a_i = a_{i,c} + a_{i,s}$ where $a_{i,c}$ and $a_{i,s} \in \{0, 1\}$. The result is a redundant n-digit carry-save number, consisting of the two binary numbers S (sum bits) and C (carry bits). The carry save adder has a constant delay (i.e., independent of n).

- **Parallel Prefix/Carry-Lookahead Adders (PPA/CLA):** Parallel-prefix adders (PPA) are adders using the direct parallel-prefix scheme for fast carry computation. They are also called carry-lookahead adders (CLA). Let us consider n inputs $(x_{n-1}, x_{n-2}..., x_0)$ and an arbitrary associative operator "$\bullet$". In the prefix problem, n outputs $(y_{n-1}, y_{n-2}, ..., y_0)$ can be computed from the n inputs using the operator $\bullet$ as follows:

  $y_0 = x_0$
  $y_1 = x_1 \bullet x_0$
  $\vdots \quad \vdots$
  $y_{n-2} = x_{n-2} \bullet x_{n-3} \bullet \cdots \bullet x_1 \bullet x_0$
  $y_{n-1} = x_{n-1} \bullet x_{n-2} \bullet \cdots \bullet x_1 \bullet x_0$

  The problem can also be formulated recursively:
  $y_0 = x_0$
  $y_i = x_i \bullet y_{i-1}; \quad i = 1, 2, ..., n-1$

  In other words, in a prefix problem every output depends on all inputs of equal or lower magnitude, and every input influences all outputs of equal or higher magnitude. Binary carry-propagate addition can be formulated as a prefix problem using intermediate prefix variables [84]: the carry generate term g_i and the carry propagate term p_i. This addition, treated hereinafter, has been considered in the realization of the most efficient modulo $(2^n \pm 1)$ adders. Other addition schemes exist in the

literature as Carry-Skip Adder (CSA), Carry-Select Adder(CSA), Carry-Increment Adder (CIA), etc. Reference [83] gives good details about the comparison between the different algorithms.

3.2.1.2 Modulo $2^n + 1$ addition: state of the art

The most important work that treat the modular addition (modulo $2^n \pm 1$) is based on the parallel prefix scheme and diminished-one number system [85][86][87][88]. In the diminished one number system, each number X is represented by $X' = X - 1$ and the representation of the value 0 is treated in a special way (i.e. 0 is not used or treated separately). Therefore, diminished-one modulo $(2^n + 1)$ circuits require only n bits for their number representations. The architectures presented in [85][86][87][88] are derived from a normal binary adder. In this subsection, to clearly show how a modular adder can be derived from a normal adder, we consider the work of Zimmermann [85] to illustrate the idea and subsequently compare it to our proposed modulo$(2^n + 1)$ adder treating the numbers in normal representation.

Binary numbers with n bits are denoted as $A = a_{n-1}a_{n-2}...a_0$ and can be written as

$$A = \sum_{i=0}^{n-1} 2^i a_i.$$

The reduction of the number A modulo $(2^n + 1)$ can be accomplished by a division and the remainder constitutes the result, or by iteratively subtracting the modulus until $A < 2^n + 1$. Since

$$2^n \ mod \ (2^n + 1) = 2^n - (2^n + 1) = -1,$$

the reduction modulo $2^n + 1$ can be formulated as

$$A \ mod \ (2^n + 1) = (A \ mod \ 2^n - A \ div \ 2^n) \ mod \ (2^n + 1), \tag{3.5}$$

where $A \ mod \ 2^n$ and $A \ div \ 2^n$ correspond to the low and high n-bit word respectively. The modulo operation on the right hand side is used for final correction if the subtraction yields a negative result (i.e., $2^n + 1$ has to be added once). Thus, the modulo $(2^n + 1)$ reduction can be computed by subtracting the high n-bit word from the low n-bit word and then conditionally adding $2^n + 1$ [89].

Furthermore, the modulo operator has the property that the sum and the product modulo M are equivalent to the sum and the product of their operands modulo M:

$$(A + B) \ mod \ (M) = (A \ mod \ M + B \ mod \ M) \ mod \ M,$$

$$(A.B) \ mod \ (M) = (A \ mod \ M).(B \ mod \ M) \ mod \ M.$$

Using the prefix scheme, Zimmermann proposed an architecture relying on the end-around-carry adder. The binary addition principle is as follows.

Let $A = a_{n-1}a_{n-2}...a_1a_0$ and $B = b_{n-1}b_{n-2}...b_1b_0$ two n-bit numbers and $S = s_{n-1}s_{n-2}...s_1s_0$ their sum. Using the commonly known terms, the carry generate term $g_i = a_i.b_i$ and the carry propagate term $p_i = a_i + b_i$, where + denotes an OR operation, the computation of the carries can be done in parallel by implementing the formula:

$$c_i = g_i + \sum_{j=0}^{i-1}(\prod_{k=j+1}^{i} p_k)g_j + \prod_{k=0}^{i} p_k c_{in},$$

where c_{in} represents the initial carry-in. The bits $s_{n-1}s_{n-2}...s_1s_0$ of the sum S are defined as $s_i = h_i \oplus c_{i-1}$, where $h_i = a_i \oplus b_i$ and $\oplus$ denotes an exclusive-OR operation. By introducing the associative operator $\bullet$, the carry computation can be transformed into a prefix problem [84]:

$$(g_m, p_m) \bullet (g_k, p_k) = (g_m + p_m.g_k, p_m.p_k),$$

and the carries are given by $c_i = G_i$, where G_i is the first member of the group relation (assuming that carry input $c_{in} = 0$):

$$(G_i, P_i) = \begin{cases} (g_0, p_0) & \text{if } i = 0 \\ (g_i, p_i) \bullet (G_{i-1}, P_{i-1}) & \text{if } 1 \leq i \leq n-1. \end{cases}$$

Fig. 3.3 shows the general prefix adder structure. A variety of other prefix structures with different sizes exists, which represents alternative circuit area-delay trade-offs. It is shown in [83] that the class of prefix adders contains the most efficient adder architectures for the entire range of area-delay trade-offs, i.e., from the smallest ripple-carry adder (serial prefix) to the fastest carry-lookahead adder (parallel-prefix).

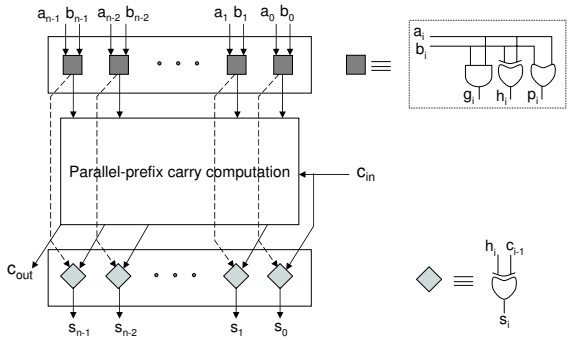

Figure 3.3: The block diagram of a parallel-prefix adder [85]

By using the diminished one number representation, ordinary addition of two numbers A and B represented by $A' = A - 1$ and $B' = B - 1$ respectively, looks as follows:

$$A + B = S$$
$$(A' + 1) + (B' + 1) = S' + 1$$
$$A' + B' + 1 = S',$$

and the modulo $(2^n + 1)$ can be formulated as

$$\begin{aligned} (A' + B' + 1)\ mod\ (2^n + 1) &= \begin{cases} (A' + B')\ mod\ 2^n & \text{if } A' + B' \geq 2^n \\ (A' + B' + 1)\ mod\ 2^n & \text{if } A' + B' < 2^n \end{cases} \\ &= (A' + B' + \overline{c}_{out})\ mod\ 2^n. \end{aligned}$$

In order to realize this modulo $(2^n + 1)$ addition using the parallel-prefix adder, Zimmermann proposed to use an end-around-carry parallel-prefix adder with $c_{in} = \overline{c}_{out}$ (i.e., with an inverter in the carry-feedback path):

$$(A' + B' + 1) \ mod \ (2^n + 1) = (A' + B' + \overline{c}_{out}) \ mod \ 2^n. \tag{3.6}$$

Using an ordinary adder and by connecting the carry output via an inverter back to the carry input may create a combinational loop and then produce an erroneous sum. To fix this problem, Zimmermann proposed to incorporate an extra stage in the prefix adder circuit as shown in Fig. 3.4. The new prefix-structure is increased by n black nodes and the critical path by one black node.

For small and medium operand widths, this adder forms a good compromise in both complexity and delay terms. However, for wide operands, the re-entering carry input has a very large fan-out requirement, leading to considerably slower designs. Moreover, for wide operands, the area grows considerably because of the re-entering carry's buffering requirements.

In [86], to avoid the disadvantage of large fan-out, the authors utilized the idea of carry recirculation in each prefix level, instead of doing that at the end of the addition. The resulting architecture is capable to operate as fast as the fastest known integer modulo 2^n or modulo (2^n-1) with the penalty of increased implementation area. Another structure is proposed by the authors based on CLA architectures with more than one level of CLA and using the idea of performing the addition in two cycles. This CLA adder design produces both faster and smaller implementations for small n. In [87], an efficient adder architecture is proposed. This architecture is based on the idea to decompose the architecture of adder of Fig.3.4, used as a building block, into m blocks (m=2 or 3) and a single AND-OR complex gate is utilized for forming the carry output for each block. The modulo $(2^n + 1)$ addition is considered as a two cycles operation. During the first cycle, the ordinary addition takes place. In the second, the carry output is complemented and connected to the carry-in to be added to the two operands. The resulting adder is called select-prefix adder due to its similarity to carry-select adder. The authors of [87] showed that for $n > 8$ this adder is capable of offering the best compromise delay-area complexity compared to the adders proposed in [86].

All the architectures of modulo $(2^n + 1)$ adder described above, use the diminished one number representation. These diminished-one adders suffer from the problem of the correct interpretation of all the zero outputs, since it may represent a valid zero output (that is, an addition with a result of 1) or a real zero output (that is, an addition with a result of 0). As an example, let us consider the diminished-one modulo 9 addition of $A = 6$ and $B = 4$ with $C = 5$. In diminished-one number we have $A' = 5 = 101_2$, $B' = 3 = 011_2$ and $C' = 4 = 100_2$. The diminished-one modulo additions are presented in the Fig. 3.5.

To detect the correct result from the real zero (incorrect result), it is necessary to implement an additional circuit depicted in Fig. 3.6 [87]. The real zero results when two inputs are complementary. Then, this circuit indicates '0', '1' when a real zero and correct result are obtained respectively.

However, the diminished-one number system often requires the conversion from and to the normal number representation using incrementation/decrementation which might be too expensive when compared to its advantage [85]. In addition, compared to the normal binary adders, the modulo $2^n + 1$ adder requires some extra stages which leads to

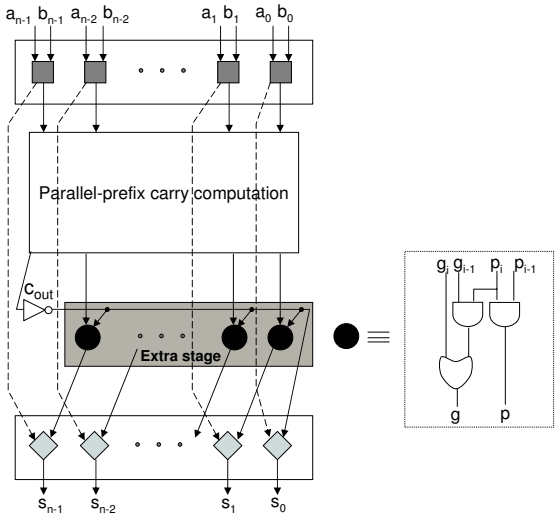

Figure 3.4: The modulo $2^n + 1$ adder for parallel-prefix architecture and diminished-one arithmetic [85]

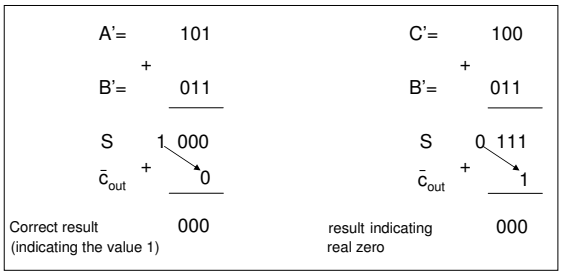

Figure 3.5: Diminished-one modulo additions

an increase of the adder's complexity. Taking these facts into account, and the fact that the modular operators which we aim to realize will be implemented using the arithmetic operators available in the complex butterfly, these modular operators should deal with numbers in normal representation. This type of realization allows us to exploit at a great advantage the existing resources and perform their re-configuration at a minimum cost.

3.2.1.3 The proposed modulo $2^n + 1$ adder

The algorithms described in the previous subsection are generally designed for standard integrated circuits and are based on very-low level basic elements such as NAND and XOR gates. However, as mentioned in the introduction of this chapter, the target hardware of our implementation is FPGA devices. These devices embed dedicated carry logic, memory blocks and some multipliers. Taking advantage of these embedded building blocks, the FPGA-based modular adders that we have proposed can outperform classical architectures.

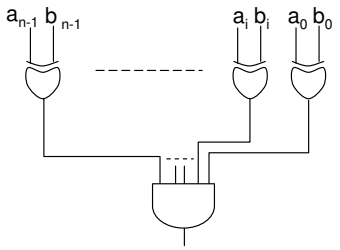

Figure 3.6: Real zero indication circuit

Let us consider the modulo $(2^n + 1)$ addition of equation 3.6. By using the normal number representation, Beuchat [90] proposed to use two adders, one for the binary addition and the second for the modulo $2^n + 1$ reduction. In this case, the symbol 2^n that requires $(n + 1)$-bits for its binary representation should be treated separately. The sum modulo $(2^n + 1)$ of two numbers x and y of the equation 3.6 can be rewritten as

$(x + y + 1)\ mod\ (2^n + 1) =$

$$\begin{cases} 2^n & \text{if } x = 2^n \text{ and } y = 2^n \\ (x + y)\ mod\ 2^n + \overline{c}_{out} & \text{if } 0 \leq x + y < 2^{n+1}. \end{cases} \tag{3.7}$$

The direct implementation of equation 3.7 is illustrated in Fig. 3.7.a. By representing the sum of the two operands x and y of equation 3.7 by $(n + 2)$-bits, (1.6) can be reformulated to be written as [90]

$$(x + y + 1)\ mod\ (2^n + 1) = (x + y)\ mod\ 2^n + s_{n+1}2^n + \overline{s_{n+1} \vee s_n}, \tag{3.8}$$

where s_n represent the nth bit of the output of the first adder. This last equation leads to the architecture illustrated in Fig. 3.7.b that returns the desired result incremented by one [90]. The proof of correctness of this algorithm can be found in [90].

Let us consider the classical structure of the complex butterfly shown in Fig. 3.8. The implemented adder in that structure is a complex operator consisting of two real adders. By exploiting these available adders we can, from one hand, avoid the change of the binary adder and the use of an extra stage for the modular reduction. From the other hand, we can take advantage in terms of speed and complexity improvements that the embedded adders in the FPGAs can allow. Starting from this consideration and the considerations of the diminished-one system mentioned in the previous subsection, we propose a modulo $2^n + 1$ adder based on the idea to use the two binary adders available on the complex butterfly. The adder we propose, depicted in Fig. 3.9.a, gives the desired result directly and the modulo $2^n + 1$ addition can be expressed as:

$$(x + y)\ mod\ (2^n + 1) = \begin{cases} (x + y)\ mod\ 2^n & \text{if } 0 \leq x + y < 2^n \\ ((x + y)\ mod\ 2^n + (2^n - 1))\ mod\ 2^n & \text{if } 2^n < x + y \leq 2^{n+1} \\ 2^n & \text{if } x + y = 2^n. \end{cases} \tag{3.9}$$

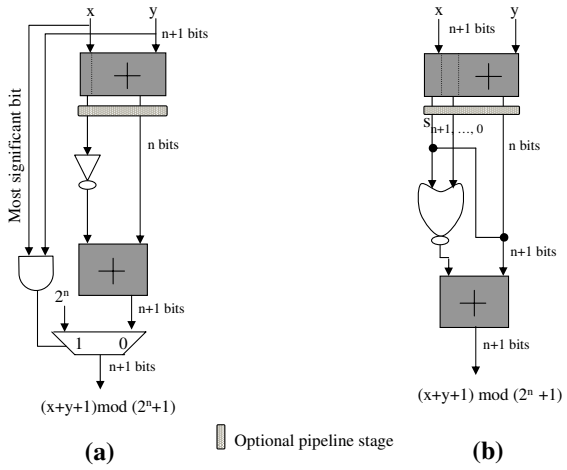

Figure 3.7: Two architectures of modulo $2^n + 1$ adder proposed by [90]

In other words,

$$(x + y)\ mod\ (2^n + 1) = \overline{S_n^2} S_2 + S_n^2 2^n, \tag{3.10}$$

where S^2 denotes the sum of the second adder:

$$S^2 = [S_{n+1}^2 S_n^2 ... S_0^2] = [S_{n-1}^1 ... S_0^1] + (2^n - 1)(S_{n+1}^1 \vee S_n^1),$$

and S^1 the sum of the first adder:

$$S^1 = [S_{n+1}^1 S_n^1 ... S_0^1] = x + y.$$

Now, let us demonstrate the correctness of equation 3.10. First of all, let us consider x and y two elements of $GF(F_t = 2^n + 1)$, $0 \leq x, y \leq 2^n$. Then,

$$0 \leq x + y \leq 2^{n+1}.$$

We have to distinguish between the four following cases to establish the correctness of our algorithm:

1. For $x + y = 2^{n+1}$ (i.e. $x = y = 2^n$), we have

   $$S^1 = 2^{n+1} (i.e.\ S_{n+1}^1 = 1, S_i^1 = 0\ for\ i = 0, ..., n),$$

   consequently $S^2 = 0 + 2^n - 1$, $S_n^2 = 0$ and our algorithm returns $2^n - 1$.

2. For $x + y = 2^n$ (i.e. $x = 0$ and $y = 2^n$ or $x = 2^n$ and $y = 0$), we have:

   $$S_n^1 = 1, S_{n+1}^1 = 0,$$

   and

   $$(S_n^1 \vee S_{n+1}^1 = 1),\ S^2 = 2^n + 2^n - 1 = 2^{n+1} - 1.$$

   In this case $S_n^2 = 1$ and the multiplexer selects 2^n as a result. This is the only case where $S_n^2 = 1$.

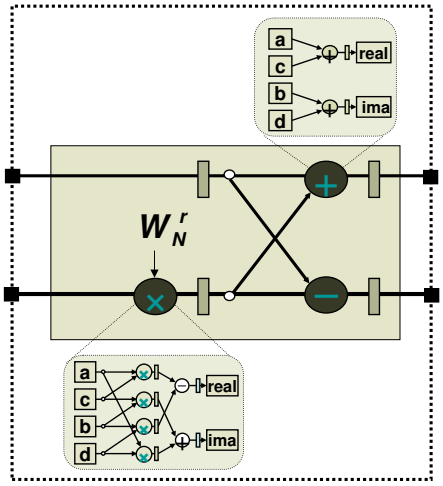

Figure 3.8: Internal structure of the complex butterfly

3. For $2^n < x + y < 2^{n+1}$, we have:

$$S^1 = 0.2^{n+1} + S_n^1 2^n + ... + S_0^1,$$

or

$$2^n \ mod\ (2^n + 1) = (-1 + 2^n + 1)\ mod\ (2^n + 1) = (-1)\ mod\ (2^n + 1).$$

The second adder of Fig. 3.9.a returns an addition modulo 2^n, then (-1) modulo $2^n = 2^n - 1$. Consequently the following equations

$$2^n + 2^n - 1 < S^2 = x + y + 2^n - 1 < 2^{n+1} + 2^n - 1,$$

$$2^{n+1} \leq S^2 < 3 * 2^n - 1,$$

give $S_n^2 = 0$ and our algorithm returns $(x + y + 2^n - 1)\ mod\ 2^n$.

4. Finally, for $0 \leq x + y < 2^n$, we have:

$$S_{n+1}^1 = S_n^1 = S_n^2 = 0 \ \ and \ \ (x + y)\ mod\ 2^{n+1} = x + y.$$

The modulo $2^n + 1$ adder of Fig. 3.9.a can be implemented onto the complex adder (Fig. 3.8) by interchanging the wired connections and incorporating an OR gate and a multiplexer. Hence, we achieve a reconfigurable adder that can perform two different kinds of additions: complex and modular additions. Fig.3.9.b illustrates the architecture of the reconfigurable adder piloted by a parameter "DM" (Dual Mode) that selects the desired functioning mode (complex or modular mode).

3.2.2 The proposed reconfigurable subtracter

As well known, the arithmetic subtracter is usually based on the arithmetic adder structure. An adder can be transformed into a subtracter by inverting the operand to be subtracted

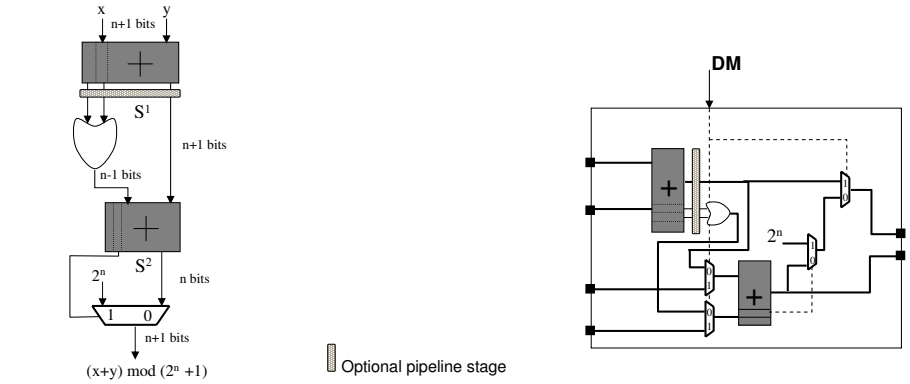

Figure 3.9: The proposed modulo $2^n + 1$ and reconfigurable adders

from the other one and by setting the carry-in to 1. Obviously, this method can be applied, but in our case, by taking advantage of the nature of the modulo $2^n + 1$ subtraction, some improvements for the subtracter can be obtained compared with the adder structure. We suggest here an architecture for the modular subtracter whose subtraction operation can be expressed as:

$$(x - y)\ mod\ (2^n + 1) = \begin{cases} 2^n & \text{if } (x = 2^n \text{ and } y = 0) \\ (x + \overline{y} + 1 + S_n)\ mod\ 2^n & \text{otherwise.} \end{cases} \tag{3.11}$$

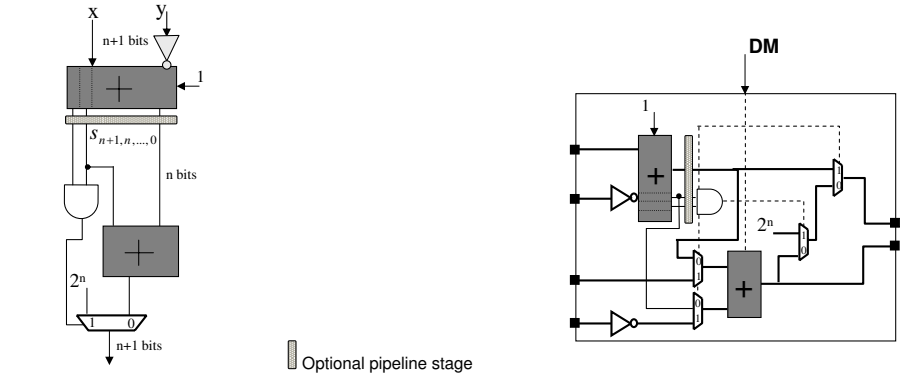

Figure 3.10: The proposed modulo $2^n + 1$ and reconfigurable subtractors

Let us demonstrate the correctness of this algorithm. First of all, let us consider x and y any two $(n+1)$-bit elements in $GF(F_t)$ (y the operand to be subtracted from x) and $S = (x + \overline{y} + 1)$ their $(n+2)$-bit sum before the modular reduction. x and y satisfy the

equation: $0 \leq x + y \leq 2^{n+1}$.
We have:

$$2^n - 1 \leq \overline{y} \leq 2^{n+1} - 1,$$

that gives

$$2^n - 1 + 1 \leq x - y = x + \overline{y} + 1 \leq 2^n + 2^{n+1} - 1 + 1$$

$$2^n \leq x + \overline{y} + 1 \leq 3 * 2^n.$$

We have to distinguish between the three following cases to establish the correctness of our algorithm:

1. if $x \geq y \Longrightarrow x + \overline{y} + 1 \geq 2^{n+1},\ S_{n+1} = 1,\ S_n = 0,$
   and the algorithm returns $x + \overline{y} + 1$.
2. if $x \leq y \Longrightarrow x + \overline{y} + 1 < 2^{n+1},\ S_{n+1} = 0,\ S_n = 1,$
   and the algorithm returns $x + \overline{y} + 1 + 1$.
3. if $(x = 2^n\ and\ y = 0) \Longrightarrow\ S_{n+1} =\ S_n = 1,$
   and the algorithm returns 2^n.

Fig. 3.10.a depicts the hardware architecture of the modulo $2^n + 1$ subtracter. Incorporating this subtracter into the complex one, we obtain the reconfigurable subtracter illustrated in Fig. 3.10.b. Here, we note that the carry-in of the second adder/subtracter of Fig. 3.10.b. is connected to the parameter "DM" which is set to "1" when the complex subtraction is done and passes to "0" when the modular subtraction is performed.

3.2.3 The reconfigurable multiplier

In this section, we shine the light at the different ways to perform multiplication modulo $2^n + 1$ and then our proposed multiplier will be discussed.

3.2.3.1 Multiplication over GF(F_t): state of the art

The modulo $2^n + 1$ multiplication is widely used in the computation of convolutions, Residue Number Systems (RNS) [81] and cryptography (IDEA International Data Encryption Algorithm) [91]. Many algorithms and implementations have been developed for implementing multiplication in standard integrated circuits as well as in FPGAs. Some algorithms are based on very low-level basic elements such as full-adder and NAND/XOR gates. On the other hand, advantages can be taken from embedded arithmetic resources available in the recent FPGAs devices. In this subsection, we describe the principal techniques used for the modulo $2^n + 1$ multiplication and then we discuss the most appropriate algorithms to our application. Modulo $2^n + 1$ multiplication techniques discussed below can be divided into three principal classes [92] [93]:

1. multiplication by means of look-up table ("quarter square" tables, logarithm tables);
2. multiplication by an $(n + 1) * (n + 1)$-bit multiplier;
3. multiplication based on Carry-Save adders;

• **Multiplication by means of Look-Up Tables (LUT):** In this method the products of modulo $(2^n + 1)$ multiplications are stored in a table and the desired result is obtained from a memory cell. The content of this memory cell is indexed by an address formed by the two operands. As mentioned in [92], the table grows exponentially with the wordlength n. The associated ROM implementation requires a memory space of $2^{2n} * n$ bits (n=8: 512 kb; n=16: 64 Gb). Hence, for a wordlength $n \geq 8$, the LUT is not feasible for integration due to its area requirement. To reduce the required memory, two methods were suggested. The first one called "quarter squared" table based on the following equation [94]:

$$xy \ mod \ (2^n + 1) = \left(\left(\frac{x+y}{2} \right)^2 - \left(\frac{x-y}{2} \right)^2 \right) mod \ (2^n + 1)$$

$$= (\Phi(x+y) - \Phi(x-y)) \ mod \ (2^n + 1)$$

where $\Phi(x+y)$ and $\Phi(x-y)$ are evaluated by LUT. This method permits the table to reduce the memory size from $O(2^{2n} \times n)$ to $O(2^n \times n)$.

The second method is based on the fact that the multiplication modulo p is isomorphic to the addition modulo $(p-1)$ [95]. In this method, the logarithms of the two operands are determined first and the result is then obtained by calculating the anti-logarithm of their sum. Compared to the first method, table sizes remain the same but there is no need for an explicit modulo $(2^n + 1)$ arithmetic unit. From these considerations, the authors in [92] conclude that LUT methods cannot be a satisfactory alternative for modulo $(2^n + 1)$ multiplication when large word length are processed.

• **Multiplication by an $(n+1) * (n+1)$-bit multiplier:** This method starts from the equation 3.5 recalled below:

$$P \ mod \ (2^n + 1) = (P \ mod \ 2^n - P \ div \ 2^n) \ mod \ (2^n + 1). \tag{3.12}$$

Let us notice that the algorithms developed in this class deals with the numbers belonging to the set $\mathbb{Z}^*_{2^n+1}$ defined as $\mathbb{Z}^*_{2^n+1} = \{a \in \mathbb{Z}_{2^n+1} \mid gcd(a, 2^n + 1) = 1\}$ is a group of order $\phi(2^n + 1)$ under the operation of multiplication modulo $(2^n + 1)$, where ϕ denotes the *Euler's totient function* (for $a \geq 1$, $\phi(a)$ denotes the number of integers in {1,...,a} which are relatively prime to a). If $(2^n + 1)$ is a Fermat prime, the set $\mathbb{Z}^*_{2^n+1}$ contains $\phi(2^n + 1) = 2^n$ elements and the following bijection between $\mathbb{Z}_{2^n} = \{a \mid 0 \leq a \leq (2^n - 1)\}$ and $\mathbb{Z}^*_{2^n+1} = \{a \mid 1 \leq a \leq 2^n\}$ is considered. Consequently, the element 0 is replaced by 2^n. The original algorithm used in this technique was the "Low-High" algorithm which was described by the designer of the IDEA used to perform the modular multiplication [93]. This Low-High algorithm provides the programmer with a tool to perform modulo $2^n + 1$ multiplication (denoted by $\odot$) of two integers when they are in $\mathbb{Z}^*_{2^n+1}$. By noticing that the integer 2^n is congruent to (-1) modulo $(2^n + 1)$, equation 3.12 can be rewritten as

$$x \odot y = (c_L - c_H) \ mod \ (2^n + 1) = \begin{cases} (c_L - c_H) \ mod \ 2^n & \text{if } c_H \leq c_L \\ (c_L - c_H + 1) \ mod \ 2^n & \text{if } c_H > c_L, \end{cases} \tag{3.13}$$

where c_L and c_H are the lower and higher words respectively of the product $x \odot y$ defined as

$$c_L = \sum_{i=0}^{n-1} p_i 2^i \quad and \quad c_H = \sum_{i=0}^{n-1} p_{n+i} 2^i.$$

This method implements the $\odot$ operator when x and y are nonzero. However, the cases where $x = 0$ or $y = 0$ must be treated separately. To remedy this problem, Lai in [91] slightly modified c_L and c_H:

$$c'_L = \begin{cases} (2^n + 1 - x) \ mod \ 2^n & \text{if } y{=}0, \\ (2^n + 1 - y) \ mod \ 2^n & \text{if } x{=}0, \\ xy \ mod \ 2^n & \text{otherwise,} \end{cases} \quad and \quad c'_H = \begin{cases} 0 & \text{if } y{=}0, \\ 0 & \text{if } x{=}0, \\ xy \ div \ 2^n & \text{otherwise.} \end{cases} \tag{3.14}$$

Consequently, $x \odot y$ can be performed according to

$$x \ \odot \ y = \begin{cases} (c'_L - c'_H) \ mod \ 2^n & \text{if } c'_H \leq c'_L, \\ (c'_L - c'_H + 1) \ mod \ 2^n & \text{if } c'_H > c'_L, \end{cases} \tag{3.15}$$

The direct hardware implementation of equation 3.15 requires an unsigned $n \times n$-bit multiplier, two subtractors and a multiplexer. The modulo $(2^n + 1)$ reduction involves a comparator, a subtracter and an adder ([93] Fig.1.a). To reduce the area of the $\odot$ operator (by removing the two subtracters and the comparator), [93] has suggested to redefine c''_L and c''_H as

$$c''_L = \begin{cases} 0 & \text{if } x \neq 0 \text{ and } y = 0, \\ 0 & \text{if } x = 0 \text{ and } y \neq 0, \\ 1 & \text{if } x = 0 \text{ and } y = 0, \\ xy \ mod \ 2^n & \text{otherwise,} \end{cases} \quad and \quad c''_H = \begin{cases} 0 & \text{if } x \neq 0 \text{ and } y = 0, \\ 0 & \text{if } x = 0 \text{ and } y \neq 0, \\ 1 & \text{if } x = 0 \text{ and } y = 0, \\ xy \ div \ 2^n & \text{otherwise,} \end{cases} \tag{3.16}$$

and by noting that

$$(c''_L - c''_H) \ mod \ 2^n = (c''_L - c''_H + 2^n) \ mod \ 2^n = (c''_L + \overline{c''_H} + 1),$$

$$(c''_L - c''_H + 1) \ mod \ 2^n = (c''_L + \overline{c''_H} + 2) \ mod \ 2^n, \tag{3.17}$$

the test on c''_L and c''_H can be rewritten as:

$$c''_L \geq c''_H \Leftrightarrow c''_L + 2^n - c''_H \geq 2^n \Leftrightarrow c''_L + \overline{c''_H} + 1 \geq 2^n. \tag{3.18}$$

Consequently, the comparison between c''_L and c''_H can be transformed to a simple check of the carry-out of the sum $c''_L + \overline{c''_H} + 1$. Then, Equation 3.15 can be rewritten as

$$x \ \odot \ y = \begin{cases} (c''_L + \overline{c''_H} + 1) \ mod \ 2^n & \text{if } c''_L + \overline{c''_H} + 1 \geq 2^n, \\ (c''_L - c''_H + 1) \ mod \ 2^n & \text{if } c''_L + \overline{c''_H} + 1 < 2^n. \end{cases} \tag{3.19}$$

With equation 3.19, one still need to test the four cases of the input data (x and y) expressed by the Equation 3.16. To simplify these tests, Curiger [92] proposed an

architecture that uses an $(n+1) \times (n+1)$-bit multiplier (instead of $n \times n$-bit multiplier) to treat the special cases $x = 2^n$ and/or $y = 2^n$ correctly. The modular reduction is performed by implementing a subtracter, modulo 2^n adder and an OR gate to set the carry-in of the modulo 2^n adder. However, Beuchat [93] showed that the architecture proposed by Curiger suffers from an incorrect result when $x = 1$ and $y = 1$. To remedy the problem, Beuchat suggested, by exploiting the most significant bit of the product $P = xy$, to decompose P as

$$P = P_{2n}2^{2n} + 2^n \sum_{i=0}^{n-1} p_{n+i}2^i + \sum_{i=0}^{n-1} p_i 2^i,$$

$$x \odot y = \begin{cases} (c_L''' + \overline{c_H'''} + 2) \ mod \ 2^n & \text{if } P_{2n} = 1 \text{ or } (P_{2n} = 0 \text{ and } c_L''' + \overline{c_H'''} + 1 < 2^n), \\ (c_L''' + \overline{c_H'''} + 1) \ mod \ 2^n & \text{otherwise,} \end{cases} \quad (3.20)$$

where $c_L''' = \sum_{i=0}^{n-1} p_i 2^i$ and $c_H''' = \sum_{i=0}^{n-1} p_{n+i}2^i$. The hardware architecture implementing the equation 3.20 is depicted in Fig. 3.11.

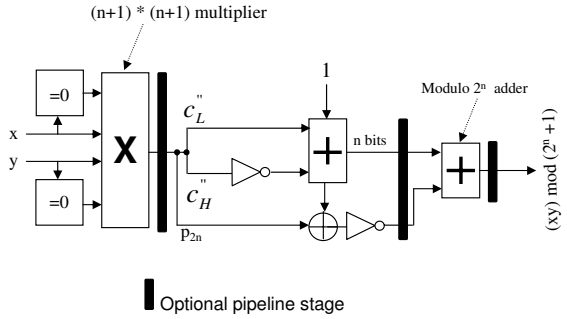

Figure 3.11: The modulo $2^n + 1$ multiplier [93]

• **Multiplication based on Carry-Save Adders:** In this method, the modulo $(2^n + 1)$ reduction is performed at the level of each partial product $x_i y$ (instead of being done at the end of the multiplication). In [92], Curiger proposed a modulo 2^n+1 multiplier architecture with Booth-recoded partial products and concurrent modulo reduction with carry-save addition. This architecture was improved in [96]. In [89], a modulo $(2^n + 1)$ multipliers based on diminished-one number representation with highly regular modulo carry-save adder arrays and trees were realized. In [85], the architecture proposed by [89] was improved by the precomputation of a correction term Z, using a faster final adder and using a normal number representation. The corresponding algorithm is based on the following equation

$$x \ \odot \ y = (\sum_{i=0}^{n-1} (x_i \ y \ 2^i) \ mod \ (2^n + 1)) \ mod \ (2^n + 1) \quad (3.21)$$

that, by [85], can be rewritten as

$$x \odot y = (n + 2 + \sum_{i=0}^{n-1} PP_i) \ mod \ (2^n + 1), \tag{3.22}$$

where

$$PP_i = x_i.y_{n-i-1}...y_0\overline{y}_{n-1}...\overline{y}_{n-i} + \overline{x}_i.0...0 \ 1...1.$$

Here, "0...0 1...1" denotes the number with $n - i$ '0' and i '1'. The corresponding circuit involves a modulo partial product generator, a modulo $(2^n + 1)$ carry-save adder, a correction unit, a multiplexer and a final modulo $(2^n + 1)$ adder.

3.2.3.2 The proposed reconfigurable multiplier

In this subsection, we will discuss the last two classes of modulo 2^n+1 multipliers previously described. In [92], Curiger compared the architecture based on the $(n + 1) * (n + 1)$-bit multiplier to the two architectures based on the adders summing the intermediate products. The authors concluded that the first architecture offers an excellent time/complexity trade-off compared to other architectures.

In [93], Beuchat compared the same two classes of multipliers by considering the implementation of the architecture based on the $(n + 1) * (n + 1)$-bit multiplier and the carry-save adder based architecture on FPGAs devices from Xilinx family. As the author noticed, the first architecture allows a significant gain in terms of area and delay compared to the second architecture when the implementation on Virtex-II is considered. The implementation on Virtex-E showed that this gain in terms of area decreases while the delay worsens when stage pipelines are not considered. The algorithms based on the entire multipliers are adapted to the FPGAs implementation and particularly in the case of devices embedding hardwired multipliers. For this, the $(n + 1) \times (n + 1)$-bit multiplier presented an important gain in terms of area and delay when implemented on Virtex-II where embedded multipliers can be used. This gain decreases in the case of Virtex-E where no multipliers are embedded.

Remembering that the modulo multiplier we aim to design should treat the numbers in $\mathbb{Z}_{2^n+1}$ rather than in $\mathbb{Z}^*_{2^n+1}$ and will be implemented onto the complex multiplier where entire real multipliers are available. Taking into account these considerations and the results of discussion drawn above, we have proposed to adopt the modulo 2^n+1 multiplier based on the $(n+1)*(n+1)$-bit multiplier. By slightly modifying the algorithm described in [93], we designed our modulo $2^n + 1$ multiplier (Fig. 3.12) whose modular addition is realized according to the following equation:

$$x \odot y = \begin{cases} (c_L + \overline{c_H} + 2) \ mod \ 2^n & \text{if } c_L + \overline{c_H} + 1 < 2^n \\ (c_L + \overline{c_H} + 1) \ mod \ 2^n & \text{otherwise,} \end{cases} \tag{3.23}$$

where x and y are elements from $\mathbb{Z}_{2^n+1} = \{0, \ 1, \ ..., \ 2^n\}$.

In this architecture, we have eliminated the two blocks dedicated to test the nil values of operands x and y since these nil values will be included in the modular computations. The other simplification is the elimination of the OR gate as the most significant bit (P_{2n}) is not necessary for the multiplication range required to perform the FNT transform. This simplification comes from the fact that the second operand will be represented by the powers of the primitive element α^i, $i \in \{0, \ 1, \ ..., \ \frac{F_t-1}{2} - 1\}$ with a range that does

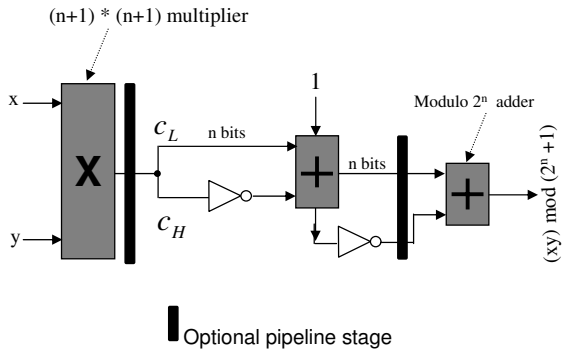

Figure 3.12: Proposed modulo $(2^n + 1)$ multiplier

not exceed $F_t - 2$ since $\alpha^{\frac{Ft-1}{2}} = F_t - 1 = -1 \; mod \; (F_t)$. The modulo $2^n + 1$ multiplier is now realized and therefore by incorporating it onto the complex multiplier, we obtain the reconfigurable multiplier illustrated in Fig. 3.13. This operator, having two operating modes, is capable of performing complex and modulo 2^n+1 multiplications. The switching from one mode to the other can be done by a simple adjustment of the control signal "DM". This signal pilots the multiplexors that are capable to change the connections between the operators. In the modular mode, as per Fig. 3.13, only the highlighted blocks will be activated to perform the modulo $2^n + 1$ multiplication. However, the whole architecture will be fully utilized when a complex multiplication is solicitated.

In our scenario, the signal DM is set to 1, 0 in the complex and modular mode respectively. As for the operators size "n_c", representing the wordlength, it is fixed according to the complex FFT architecture and the number of bits allocated to represent the elements of $GF(F_t)$ is equal to $2^t + 1$. Only a part $n + 1 = (2^t + 1)$ of the total bits n_c will be used in the modular computation as shown in Fig. 3.13. Considering the modular mode, the signal enters through the real input B_r, multiplies by the coefficients α^i (selected by the mux 1) where it then endures modular reduction by the subtracter and the adder to be driven out through the real output P_r. As for the complex mode, the data enter through the two inputs B_r and B_i, both cosine (W_r) and sine (W_i) memories are selected to undergo with the input data the complex operations and get ejected through the output signal P_r and P_i. The above discussion on the operator functioning is expressed in the following equation

$$\begin{aligned} P_r &= \begin{cases} (B_r \; W_r - B_i \; W_i) & \text{if DM=1} \\ (B_r \odot \alpha^i), \quad i = \{0, 1, ..., \frac{F_t-1}{2} - 1\} & \text{if DM=0} \end{cases} \\ P_i &= (B_r \; W_i + B_i \; W_r). \end{aligned} \tag{3.24}$$

3.2.4 The reconfigurable butterfly

The three arithmetic reconfigurable operators are designed, connected and pipelined to build the hardware structure of the reconfigurable butterfly described in Fig. 3.14. This butterfly receives two n_c-bit input signals and a signal control DM. When DM is 1, the

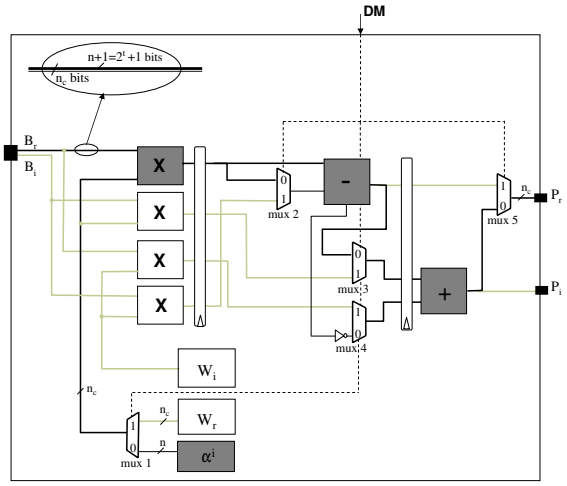

Figure 3.13: Proposed reconfigurable multiplier

butterfly operates in the complex mode and the size of operators (i.e. the wordlength) is fixed according to the complex FFT requirement (precision, area complexity). In this mode, all the arithmetic resources available in the architecture are used and the mux 1 selects the output of the ROM containing the cosine-coefficients W_r. The flow of the data through the architecture is controlled by the signal DM which commands the different multiplexors. When DM passes to 0, the butterfly operates in $GF(F_t)$ and the arithmetic operations are performed modulo F_t. The input data is driven through the two real inputs A_r and B_r, the two other inputs (A_i and B_i) are unused. In this mode, one of the four real multipliers is used and the others are deactivated. The wordlength is $n+1 = 2^t + 1$ bits that corresponds to the lower bits of the complex words represented by n_c bits. The ROM selected in this case is the one containing the powers of α (α^i). The output data is driven out on the two real ouputs P_r^1 and P_r^2 where only their lower n-bits are used. As shown in the Fig. 3.14, two pipeline stages are used to reduce the critical path. The emplacement of these stages is based on the implementation results discussed in the next chapter. Now, the heart of the reconfigurable FFT operator is designed which will allow us to design a global dual mode FFT operator ($DMFFT$) capable of carrying out, in a optimal way, two different transforms.

3.3 The Dual Mode FFT operator: DMFFT

By using the reconfigurable butterfly presented in the previous section we will realize the reconfigurable FFT operator called DMFFT. This realization of the Dual Mode FFT will be based on the realization techniques of the complex FFT while taking into account the considerations of reconfiguration aspects.

Considering the implementation of the complex FFT, apart from the arithmetic resources (multipliers and adders) requirements, two principal issues should be taken into account. The first one is the word-length or the size of the arithmetic operators where two possible number representations can be used: fixed-point and floating-point. This issue

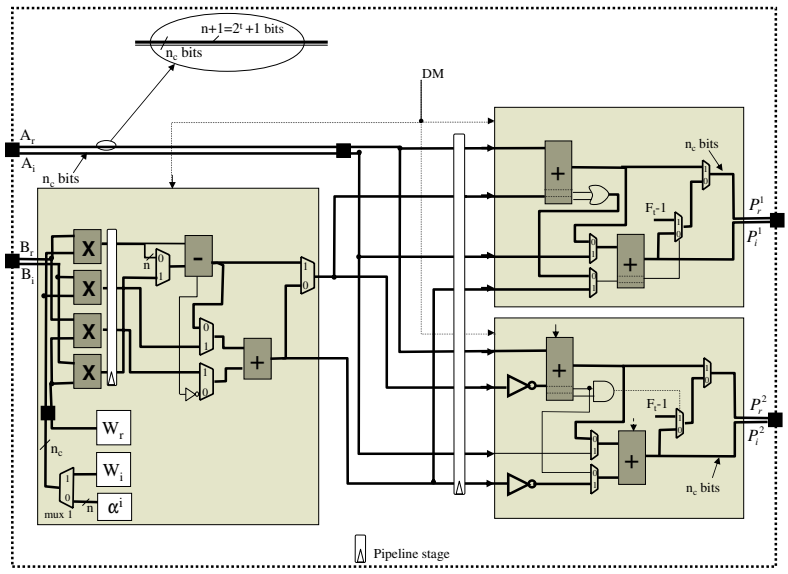

Figure 3.14: The reconfigurable butterfly architecture

affects precision, quantization errors and hardware complexity. Increased word-length of data and twiddle factors increase the precision and reduce the quantization error at the cost of area (and power). Conversely, to maintain a lower hardware cost, a shorter word-length can be chosen at the sacrifice of precision. In this work we consider the fixed-point arithmetic for the FFT computations. This matter will be discussed in section 3.4.

The second one is the memory requirement in which two memory types are required: RAM to store the intermediate data between two consecutive stages of computation, and ROM to store the twiddle factors and the powers of α.

There are many different ways of implementing an FFT operator. Among these different ways, there are two methods that can be seen as two extremes: the first one consists in performing the transform computations by using only a single memory unit and one arithmetic unit. This method leads to a very simple circuit in terms of area, but the penalty is the very high computation time required to execute the transform. The second one consists in implementing the entire FFT structure composed of log N stages and $\frac{N}{2}$ arithmetic units in each stage, where N is the transform length. For this method, no RAM blocks are needed between two consecutive stages since the entire sequence is processed in parallel. This method leads to a very high speed computation at a very large thus expensive area. As we will see later, this method is not practically feasible since it consumes the totality of a FPGA device. Between these two extremes, one can find a method that can constitute an important trade-off between area complexity and computation time. This way of implementing the FFT consists in implementing log$N$ stages where each stage contains one Processing Element (PE), some memory blocks and a control unit needed to control the data stream.

Based on this latest method and on the complex FFT structure, the DMFFT architecture we propose consists of a GCU (Global Control Unit) and log N stages as shown in Fig. 3.15.

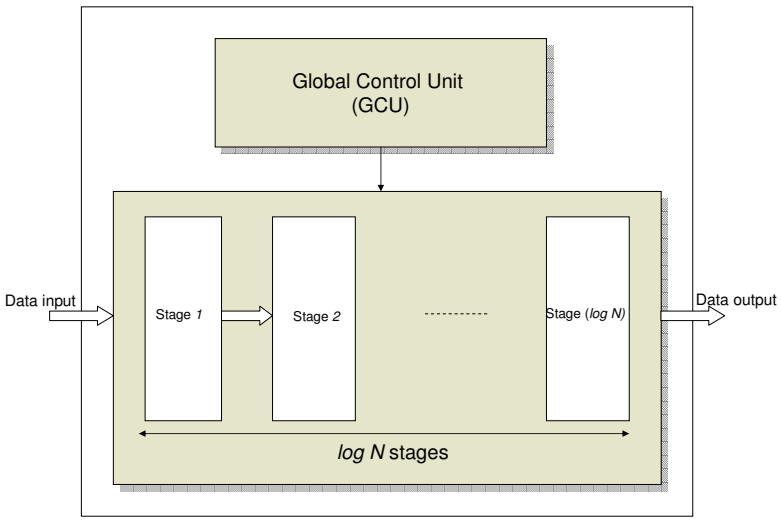

Figure 3.15: The DMFFT architecture

The uppermost GCU is composed of the following individual circuits:

1. *FFT/FNT operation selection*: there is a control signal DM which determines the operating mode. The values 1, 0 of DM indicate that the complex Fourier transform and Fermat transform are performed respectively.

2. *Transform size selection*: the DMFFT operator is designed to perform various lengths of FFT and FNT computations. The parameter m determines the number of stages to be implemented and then the transform size $N = 2^m$.

3. *Word-length size*: to provide the system designer with maximum flexibility, the input/output data and the twiddle factor's word-length has been designed in a way to vary by a simple adjustment of corresponding parameters. A parameter n_c determines the complex word-length, n_w determines the twiddle factor word-length and t determines the desired Fermat number and subsequently the length $n = 2^t + 1$ of the $GF(F_t)$ valued symbols.

This GCU is also responsible for the initialization of the entire DMFFT architecture and for the control of the data input and data output and the synchronization between two consecutive input frames.

3.3.1 Stage architecture

The proposed FFT architecture is a pipeline architecture where a pipeline level is employed between two consecutive stages. This allows a reduced critical path and then a high throughput rate. The internal structure of each stage except stage 1 is illustrated in Fig. 3.16. As shown, the stage architecture is composed of the following design units: SCU (Stage Control Unit), AGU (Address Generating Unit), memory blocks (RAM and ROM) and reconfigurable butterfly called RPE (Reconfigurable Processing Element). As

for stage 1, its architecture is similar to the one presented in Fig. 3.16 with some modifications. In this stage, no multipliers nor ROM blocks are needed since the corresponding twiddle factors are equal to 1. Consequently, the RPE of Fig. 3.16 is replaced by two operators: reconfigurable adder and reconfigurable subtracter which are described in subsections 3.2.1.3 and 3.2.2 respectively.

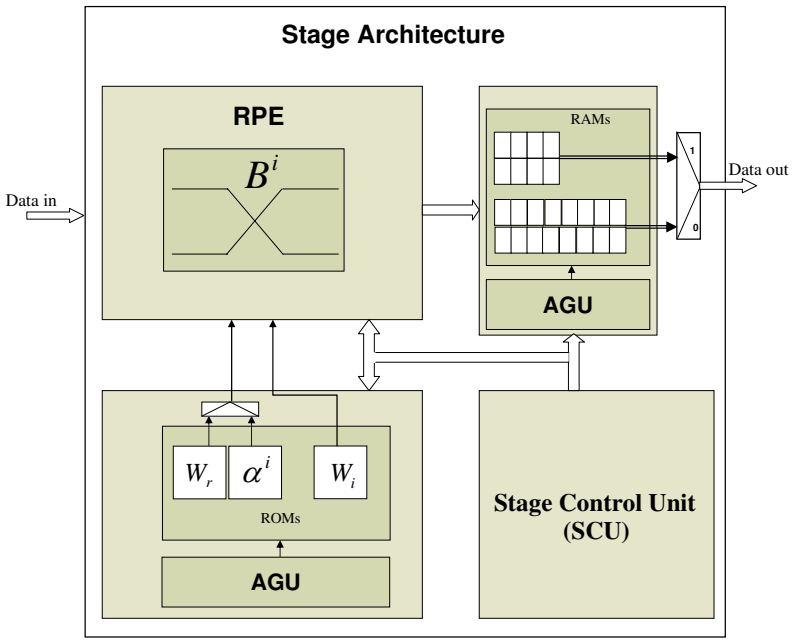

Figure 3.16: The general architecture of a DMFFT stage

3.3.2 Stage Control Unit (SCU)

Each stage in the DMFFT architecture contains a Stage Control Unit (SCU). This SCU handles the data storage and the data output in the actual stage. The data output is controlled by two signals *enable out* and *start out* connected directly to the next stage to indicate when the data is ready to be transferred. These two signals represent, for the next stage, the two signals *enable in* and *start in* that manage the data input. Then, the input data of stage i is controlled by the SCU of stage $i-1$. For stage 1, the input data is operated by the GCU. The final output data of the DMFFT is handled by the SCU of the final stage (stage $log\ N$).

The SCU is also responsible for handling the read/write operations from and to the memories (RAMs and ROMs).

3.3.3 Address Generator Units (AGUs)

The purpose of the AGUs is to provide the input/ouput RAMs and ROMs with the correct addresses. The ROM addresses of each stage are generated with the aid of virtual counter that synchronizes the output of twiddle factors (W_r, W_i and α^i) with the data stream. These twiddle factors are provided while taking into account the bit-reversal of the input data.

There is another virtual counter to control the read/write operations from and to the RAMs storage blocks. The initialization of all these virtual counters is done thanks to the control signals provided by the SCU.

3.3.4 Memory Blocks

Altera provides parameterizable megafunctions that can from one hand help to save valuable design time and from another hand allow the use of efficient and optimized functions from delay and complexity point of view. Among the resources available in the Altera's devices, there are ROM and RAM storage functions.

ROM megafunctions

The ROM megafunctions available in Altera's devices are single-port ROM with separate input and output ports. These storage functions support both synchronous and asynchronous modes of operation. The input and output ports are controlled by *input clock* and *output clock* respectively. An enable signal is provided that allows the optional use of enabling or disabling the output port. When the memory is disabled, the output port is high-impedance. The ROM initialization can be done using memory initialization file or HEX file. We have created the ROM blocks using the MegaWizard Plug-In Manager [(1)]. To initialize these ROMs, we computed the twiddle factor's coefficients for different word-lengths and various transform lengths by using MATLAB software to generate the files (.mif) and then by using Quartus II, we converted the files.mif to hexadecimal files (.hex).

RAM megafunctions

There are three types of RAM megafuntions available in Altera's devices (STRATIX II):

- **RAM: 1-PORT**
- **RAM: 2-PORT**
- **RAM: 3-PORT**

The **RAM-1-PORT** implements a single-port RAM that can operate with two modes: single clock mode and dual clock mode. In single clock mode, the read and write operations are synchronous with the same clock. Dual clock mode operates with two independent clocks: *input clock* for write operation and *output clock* for read operation.

The **RAM-2-PORT** implements a dual-port RAM function and offers the possibility of simultaneous read and write access to memory cells. The two clock modes are also available in this RAM function. In addition, for this dual port RAM, one can specify either one of the two-dual port modes: a simple dual-port mode (one read port and one write port) or a true dual-port mode (two read ports and two write ports).

[(1)] MegaWizard Plug-In Manager is a tool integrated in Quartus II software helping to create or modify design files that contain custom megafunctions variations.

The **RAM-3-PORT** implements a tri-port RAM function: one write port and two read ports. Single and dual clock modes are available in this RAM function. The tri-port RAM supports applications that require parallel data transfer in which two independent clock ports use different access rates for read and write operations.

Let us discuss the use of RAM for the DMFFT implementation. Firstly, let us describe the operating principle of a traditional FFT by considering the FFT butterfly-structure of the FFT-16 illustrated in Fig. 3.17. This architecture represents the whole FFT structure where the maximum number of RPE is implemented.

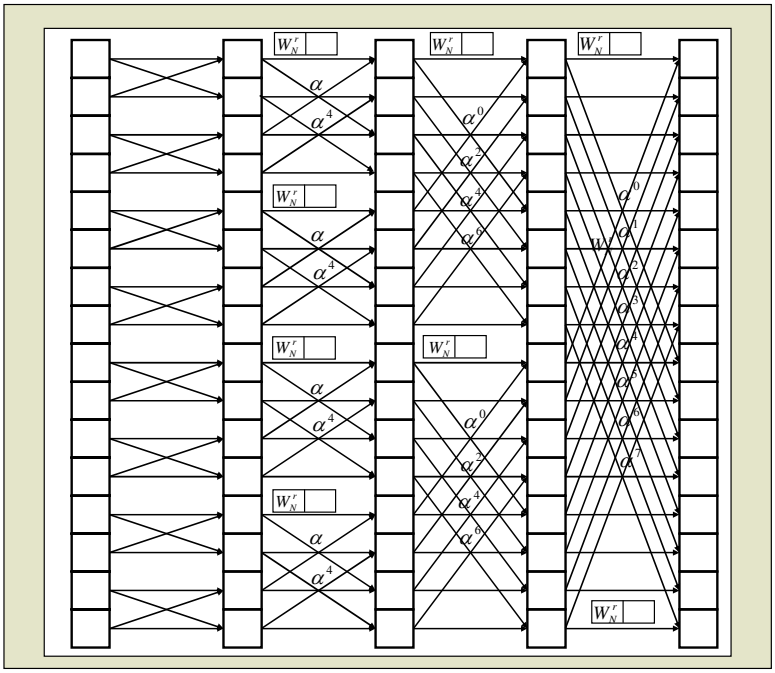

Figure 3.17: The FFT-like butterfly architecture

In this figure, only the wired butterflies with twiddle factors are shown. The purpose is to illustrate how the butterflies are routed into the FFT structure to thereafter better understand the scheduling of the butterfly processing. As discussed earlier in this section, performing the FFT computation with this architecture seems to be the most expensive solution. The alternative is to perform the FFT computation by implementing one RPE in each stage where this RPE must perform all the butterfly operations required in the given stage. But the point is that in which order the butterfly operations should be executed. Indeed, there are two methods to process these operations. The first one consists in performing the butterfly operations successively as shown in Fig. 3.18 where B_j^i represents the jth butterfly operation in the ith stage.

Due to the nature of the butterfly routing of the FFT structure shown in Fig. 3.17, there are some latencies between the start of the processing of any two consecutive stages. In Fig. 3.18, the scheduling of the butterfly processing is based on the fact that the outputs $B_{j,1}^i$ and $B_{j,2}^i$ of stage i must be consumed consecutively by the two consecutive butterfly operations of stage $i+1$. This method leads to a non FIFO buffering of the

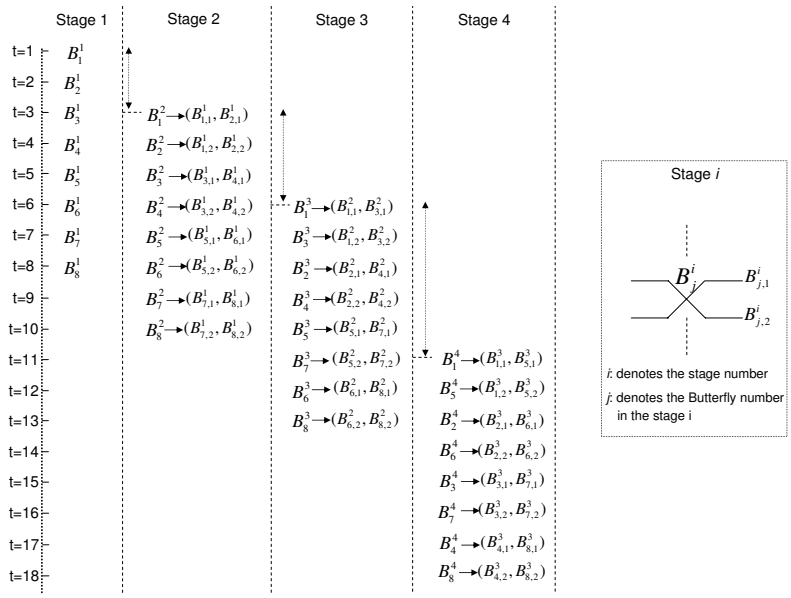

Figure 3.18: The scheduling of butterfly processing (non FIFO buffering)

data. Let us consider stage 2 and stage 3 of Fig. 3.18. The outputs ($B^2_{1,1}$ and $B^2_{1,2}$) of the butterfly B^2_1 should be consumed by the first two butterfly operations B^3_1 and B^3_3. But, B^3_1 and B^3_3 require the pair ($B^2_{3,1}$,$B^2_{3,2}$) to perform their computations correctly and since the pair ($B^2_{2,1}$,$B^2_{2,2}$) is computed and stored before ($B^2_{3,1}$,$B^2_{3,2}$), it leads to a non FIFO memory. To be processed continuously, the butterfly operations of stage 3, should start their computations after 5 clock cycles from the computation start of stage 1. Taking into account the latency of each stage, the global latency is 10 clock cycles and the first frequency sequence will be computed after 18 clock cycles.

The second method of scheduling the butterfly operations is illustrated in Fig. 3.19. This method is based on the fact that the butterfly operations of the stage i that need, for their computations, the components $B^{i-1}_{j,1}$ of stage $i-1$ to be scheduled and performed firstly. Once the available $B^{i-1}_{j,1}$ are processed, the butterfly operations employed to treat the components $B^{i-1}_{j,2}$ will be performed. This scheduling of the butterfly operations leads to FIFO buffering of the intermediate computations between any two consecutive stages. The size of the RAM needed to store the components $B^i_{j,2}$ is twice the size of that needed to store the components $B^i_{j,1}$ (as shown in Fig. 3.16). We note that the latency and the memory requirements are the same in the two methods.

According to the butterfly operation scheduling and to the principle of treatment of the data of stage i by the butterfly operations of stage $i+1$, we note the following. By using RAM-1-PORT blocks, the throughput rate of the DMFFT will be one symbol by clock cycle since the butterfly operation needs two symbols for its computation which are stored in the same memory. To provide two symbols per clock cycle, we need to use RAM-3-PORT blocks (one write port and two read ports). As shown in Fig. 3.16, for complex FFT, 4 RAM blocks are needed to store the two complex components of the butterfly (two blocks for the real components and two for the imaginary ones). In the case of finite field

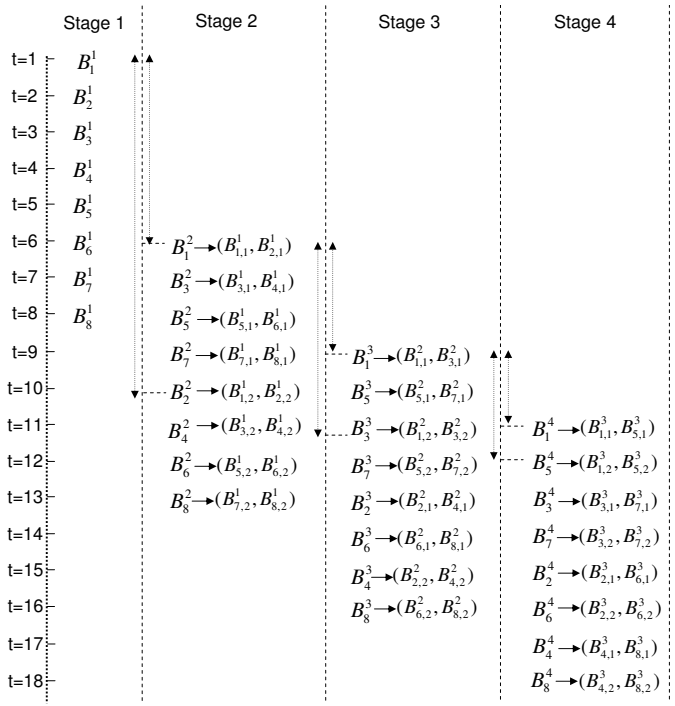

Figure 3.19: The scheduling of butterfly processing (FIFO buffering)

transform or FNT, the RAM blocks dedicated to store the real components will be reused to store momentarily the intermediate $GF(F_t)$ valued computations.

3.3.5 Reconfigurable Processing Element (RPE)

The reconfigurable butterfly designed in subsection 3.14 represents the core of the DMFFT operator. This butterfly called RPE is implemented in each stage and its operating mode is controlled by the GCU. The data stream acquisition of a RPE employed in stage i is controlled by the SCU of stage $i-1$. The RPE acquires the data and applies the multiplication of the twiddle factors with the corresponding symbols and the intermediate results continue the way to the adders and subtractors to undergo the corresponding operations. The storage of the RPE computation results in the RAM blocks is handled by the SCU.

3.4 FPGA implementation and complexity study

In chapter 1, we have brought up two approaches of designing a multi-standard or multi-mode system: the velcro approach and the reconfigurable approach, and we have suggested that this latest approach is more efficient than the former. In this section, we will justify this suggestion by implementing these two approaches and giving some evaluating figures. Embodiment of this issue by comparing the DMFFT operator performances to those of the alternative Velcro operator seems to be an appropriate solution. By Velcro operator,

we mean here a global operator that constitutes the implementation of two self-contained operators: FFT and FNT.

During the last decade, the FPGA devices have demonstrated the ever-growing ability to handle in hardware a number of complex functions which would have been developed in software. Therefore, actual FPGA-based flexible computing systems have real potentials of merging the flexibility of general-purpose computing systems and the high performance of custom designed hardware. These FPGA devices embed some fundamental arithmetic operators (multipliers, adders, etc.) that are at the heart of all computing systems. Their cost and speed performance have a direct influence on the performance of a computing system built from them. In this work, we have designed the DMFFT operator based on the processing element **RPE** which in turn is based on the fundamental arithmetic operators. Then, implementing this DMFFT on FPGA can take advantage of the embedded arithmetic resources and then provide a potential and high performance operator.

The FPGA implementation of the DMFFT and the Velcro operator will be then investigated. A complexity study of these two operators will be drawn by evaluating their performance-to-cost ratio defined in [97]. This ratio called η is expressed as $\frac{1}{TC} * 10^6$, where C is the number of logic blocks related to the cost of a FPGA-based circuit and T the execution time in nanoseconds (ns). As η increases, a better tradeoff between execution time and cost is obtained.

Usually, the mapping of a circuit design onto an FPGA device is accomplished by a software development tool provided by the FPGA manufacturer. These software tools have their predetermined cost and timing measures. The software used for our designs is the Quartus II v.6 and the device target is the Stratix II family from ALTERA.

3.4.1 The Stratix II family

This subsection gives a brief overview of useful features of Stratix II devices for this work. Stratix II presents an efficient logic structure that maximizes the performance and enables device densities approaching 180,000 equivalent Logic Elements (LEs). Stratix II devices offer up to 9 Mbits of on chip, TriMatrix memory and has up to 96 DSP (Digital Signal Processing) blocks with up to 384 ($18-bit \times 18-bit$) multipliers for efficient implementation of high performance filters and other DSP functions. Depending on the device, up to 1,170 I/O user pins can be supported.

The architecture of Stratix II devices is based on two-dimensional row and column that provides signal interconnects between Logic Array Blocks (LAB), memory block structures and DSP blocks. Each LAB consists of eight Adaptive Logic Modules (ALMs), carry chains, LAB control signals, local interconnects and register connection chain lines. An ALM is the Stratix II device family's building block of logic providing efficient implementation of user logic functions. The local interconnect transfers signals between ALMs in the same LAB. Register chain connections transfer the output of an ALM register to the adjacent ALM register in an LAB.

One ALM contains a variety of Look Up Tables (LUT)-based resources that can be divided between two Adaptive LUTs (ALUTs) as shown in Fig. 3.20. With up to eight inputs to the combinatorial logic, one ALM can implement various combinations of two functions. In addition to the adaptive LUT-based ressources, each ALM contains two programmable registers, two dedicated full adders, a carry chain, a shared arithmetic chain and a register chain. The ALUT is the cell used in the Quartus II software for logic

synthesis. Thus, the number of ALUTs will be used throughout this work as a metric to evaluate the FPGA-based circuit complexity and then the parameter η.

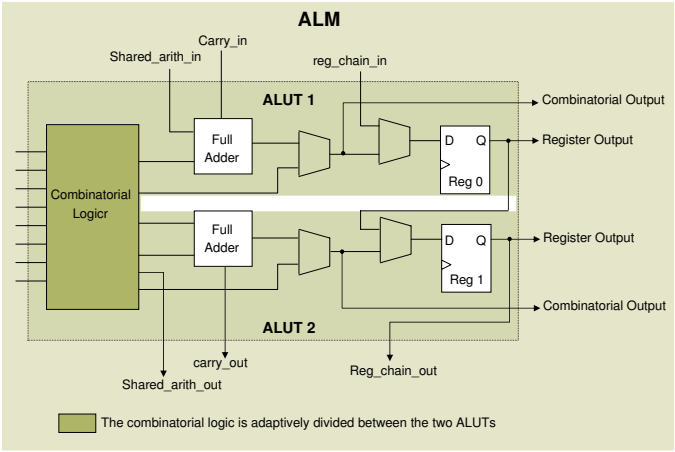

Figure 3.20: High-Level block diagram of the Stratix II ALM and its relationship with the ALUTs

We will start the complexity study by evaluating the performances of our proposed modulo $2^n + 1$ adder previously discussed in subsection 3.2.1.3 and comparing it to the adder proposed in [90]. Then, a complexity comparison between the Velcro and the reconfigurable adder will be given. This complexity comparison, Velcro vs reconfigurable approach, will continue by considering the RPE basic element to be ended with the DMFFT operator.

3.4.2 The modulo (F_t) adder complexity

We have written a synthetizable VHDL code[(2)] of the different basic elements of the DMFFT. The first experiment compares our proposed adder (Fig. 3.9) to the one proposed in [90]. The two operators have been implemented for three different adder sizes ($n = 5, 9, 17$) linked to the practical Fermat numbers F_t=17, 257 and 65537. Table 3.1 summarizes the implementation results. By analyzing the measures of the two key parameters (cost in term of ALUTs and critical path delay), we observe that the two adders have almost the same performances with a slight difference in the number of ALUTs required for large n, where some additional ALUTs are needed for our adder. But in view of the fact that our proposed adder returns exactly the desired addition result, it is still more advantageous since the adder presented in [90] needs an additional correction circuit while it returns the desired result incremented by one.

3.4.3 The reconfigurable adder complexity

By a second experiment, we have implemented a Velcro and a reconfigurable adders. The Velcro adder contains a complex and a modulo (F_t) adder implemented separately and the

[(2)] All experiments described in this paper were performed on a PC (pentium 4, 3 GHz, 1 GB of memory) running Windows XP. The VHDL code was synthesized using Quartus II version 6 and implemented on STRATIX II, EP2S15F484C3 Device with the option "Standard Fit" as the level of the Fitter's effort.

Table 3.1: Comparison between the two modulo (F_t) adders on a STRATIX II, EP2S15F484C3 device ($n = 2^t + 1$)

circuit	n=5	n=9	n=17
Fig1.k [90] (without pipeline)	16 ALUTs 2.369 ns	28 ALUTs 2.75 ns	52 ALUTs 3.48 ns
Fig.3.9a (without pipeline)	16 ALUTs 2.88 ns	28 ALUTs 3.43 ns	60 ALUTs 4.25 ns
Fig1.k [90] (with pipeline)	21 ALUTs 2.369 ns	38 ALUTs 2.369 ns	69 ALUTs 2.369 ns
Fig.3.9a (with pipeline)	22 ALUTs 2.369 ns	40 ALUTs 2.369 ns	86 ALUTs 2.369 ns

reconfigurable one contains a flexible modulo (F_t) adder able to perform both additions over $GF(F_t)$ and $\mathbf{C}$ fields. Table 3.2 summarizes their performances: η_V for the Velcro adder and η_R for the reconfigurable adder.

Table 3.2: Comparison between the reconfigurable and the Velcro adders on a STRATIX II, EP2S15F484C3 device ($n = 2^t + 1$).

circuit	n=5	n=9	n=17
Complex adder	30 ALUTs 2.369 ns	54 ALUTs 2.369 ns	102 ALUTs 2.369 ns
Modulo (F_t) adder	21 ALUTs 2.369 ns	40 ALUTs 2.369 ns	86 ALUTs 2.369 ns
Velcro adder	51 ALUTs 2.369 ns	94 ALUTs 2.369 ns	188 ALUTS 2.369 ns
reconfigurable adder	37 ALUTs 2.369 ns	67 ALUTs 2.369 ns	121 ALUTs 2.416 ns
$\eta = \frac{1}{TC} * 10^6$	$\eta_V = 8276$ $\eta_R = 10832$	$\eta_V = 4490$ $\eta_R = 6300$	$\eta_V = 2245$ $\eta_R = 3420$

The first two rows of Table 3.2, gives some figures of the implementation of each complex and modulo F_t adder implemented independently. The third row, gives the implementation measures of the Velcro operator containing the two adders implemented in single circuit. Concerning the comparison between the Velcro and reconfigurable adders, it is seen from Table 3.2 that the reconfigurable operator has the highest performance-to-cost ratio compared to the Velcro circuit. This is the expected result as the reconfigurable approach seems to be the more efficient approach.

3.4.4 The reconfigurable butterfly complexity

In the beginning of this section we have reported that the dynamic range is an important issue to be taken into account in any hardware implementation while it affects the computation precision as well as the circuit complexity. Now, we will discuss in details this issue by considering the quantification methods applied to the fixed-point FFT computations. In subsection 3.4.5, we give some figures illustrating the relationship between the wordlength

and the precision for a fixed-point FFT computations while taking into account the gain in terms of ALUTs. This can help to select the best tradeoff precision-circuit complexity. We should point out that this issue concerns the complex FFT while the wordlength required for the FNT computations is fixed by the given Galois field ($GF(F_t)$).

3.4.4.1 Quantification error analysis of the FFT

When the FFT is computed in a fixed-point arithmetic, some truncation and round operations are needed to maintain the same wordlength of the input and output data. These operations introduce some errors called *quantification errors* that can affect the accuracy of the FFT computations. Lots of researches have analyzed the effect of these quantification errors [98], [99], [100] where the output Signal-to-Quantizaion-Noise Ratio (SQNR) is evaluated. These analysis assume fixed point-arithmetic with n_c-bit wordlength where two quantification operations are considered: truncation of the multiplication result by W^P and scaling by a factor 1/2 of the addition results.

Let us consider the butterfly computations of the DIT-FFT algorithm described in the following equations:

$$B_{j,1}^{i+1} = B_{j,1}^{i} + W_N^r \; B_{j,2}^{i}$$
$$B_{j,2}^{i+1} = B_{j,1}^{i} - W_N^r \; B_{j,2}^{i},$$

where $B_{j,1}^{i+1}$ and $B_{j,2}^{i+1}$ represent the outputs of the jth butterfly operation at the (i+1)th stage. These equations show that the magnitude of computation results increases at each stage. In [100], it is shown that the maximum magnitude of the butterfly outputs at each stage is usually inferior or equals to 2:

$$max(|B_{j,1}^{i}|, |B_{j,2}^{i}|) \leq max(|B_{j,1}^{i+1}|, |B_{j,2}^{i+1}|) \leq 2 \; max(|B_{j,1}^{i}|, |B_{j,2}^{i}|).$$

Then, by applying a scaling factor $\frac{1}{2}$ to the butterfly outputs we can prevent the overflow. This quantification can be performed by a simple 1-bit right-shifting of the binary words. The variance of the resulting error is about $\sigma^2 \approx \frac{2^{-2b}}{16}$ [100], where b is the number of bits to the right of radix point after truncation.

The second quantification is done after the multiplication by the twiddle factors. After each real multiplication of two b-bit numbers, the $2b$-bit product is truncated to be b-bit number. The variance of this error truncation is $\sigma^2 \approx \frac{2^{-2b}}{12}$.

In [101], the author studied the effect of the emplacement of these two quantifications in the butterfly. After various computer simulations, the author showed that the best solution is to employ the truncation of b bits right after the real multiplier and the scaling (right bit-shifting) right after the real adder (and subtracter) as shown in Fig. 3.21.

This butterfly represents the best complexity-precision tradeoff. We note here that there are other ways of quantification that lead to better precision but need wider size of arithmetic operators.

For our implementation, since we consider an FFT/FNT operator, an additional issue should be taken into account that is the relationship between the wordlength of complex data and $GF(F_t)$ symbols. That means, before we fix the wordlength to be used in the FFT computations, we should predetermine which $GF(F_t)$ should be considered, i.e. which FNT length should be performed. In general, the dynamic range used for the fixed-point FFT implementation is between 10 and 16 bits [101]. This dynamic range allows the

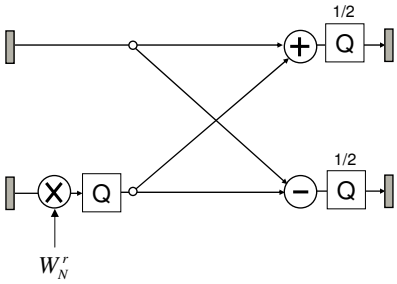

Figure 3.21: The emplacement of the quantifications in the butterfly

construction of $GF(F_t)$ for four Fermat prime numbers F_t: $t = 0, 1, 2, 3$. In practice, the most interesting $GF(F_t)$ used among these Galois fields is $GF(F_t = 257)$. It allows the various FNT lengths (from 16 up to 256) and consequently the building of various RS codes. Next subsection discusses the implementation of a dual mode butterfly realizing the FNT over $GF(257)$ and the FFT for various wordlengths. The case of $F_4 = 65537$ is also tested.

3.4.4.2 FPGA implementation of the reconfigurable and Velcro butterfly

Table 3.3 shows some measures of the implementation on Stratix II of the reconfigurable and Velcro butterflies for different n_c and n values that represent the $\mathbb{C}$ and $GF(F_t)$ symbol wordlengths respectively. The values $n = 9$ and $n = 17$ are the required bits number to map the symbols in $GF(F_t = 2^8 + 1 = 257)$ and $GF(F_t = 2^{16} + 1 = 65537)$ respectively.

Now let us discuss the figures given in Table 3.3. For the reconfigurable butterfly, there is a small delay excess of around 2% compared to the Velcro solution. This time penalty is widely balanced by complexity gain values (in term of ALUTs) around 18% in favor of the common butterfly operator. Moreover, the performance-to-cost ratio η_R has always the highest value compared to η_V. The associated gains are 20.8, 19.8 and 20.2 % for n_c equal to 9, 12 and 17 respectively.

These performance figures clearly justify once again the interest of using a common and reconfigurable operator instead of Velcro operator.

3.4.5 The DMFFT complexity study

Now we arrive to the implementation of the DMFFT. As a test, we have implemented the entire structure of the DMFFT-16 containing 4×8 butterflies. The implemented circuit consumed 60 % of the Stratix II, EP2S15F484C3 Device. Consequently, a larger length of DMFFT cannot be implemented on such FPGA device, from where comes the need to adopt a more simple structure such the structure previously illustrated in Fig. 3.15.

To evaluate the performances of this DMFFT architecture, we have implemented it on Stratix II (Fig. 3.22) and compared its performances to those of a Velcro FFT/FNT operator implemented on the same device (Fig. 3.23). The RAM blocks used in this implementation are RAM single port.

Table 3.4 shows the implementation measures for the DMFFT-64 implemented for different wordlengths. The Fourier/Fermat transforms that can be performed in this same architecture have $N = 64$ as transform length. Regarding the Fourier transforms, the

Table 3.3: Comparison between the reconfigurable butterfly and the Velcro butterflies on a STRATIX II, EP2S15F484C3 device.

circuit	n_c=9 n=9	n_c=12 n=9	n_c=17 n=17
Velcro butterfly	403 ALUTs 4.20 ns	629 ALUTs 5.07 ns	1062 ALUTs 5.60 ns
Re-configurable butterfly	326 ALUTs 4.3 ns	514 ALUTs 5.18 ns	875 ALUTs 5.64 ns
ALUT's gain	19.1 %	18.2 %	17.6 %
Delay's excess	2.18 %	2.16 %	0.7 %
$\eta = \frac{1}{TC} * 10^6$	$\eta_V = 590$ $\eta_R = 713$	$\eta_V = 313$ $\eta_R = 375$	$\eta_V = 168$ $\eta_R = 202$
Performance-to -cost ratio gain	**20.8%**	**19.8%**	**20.2%**

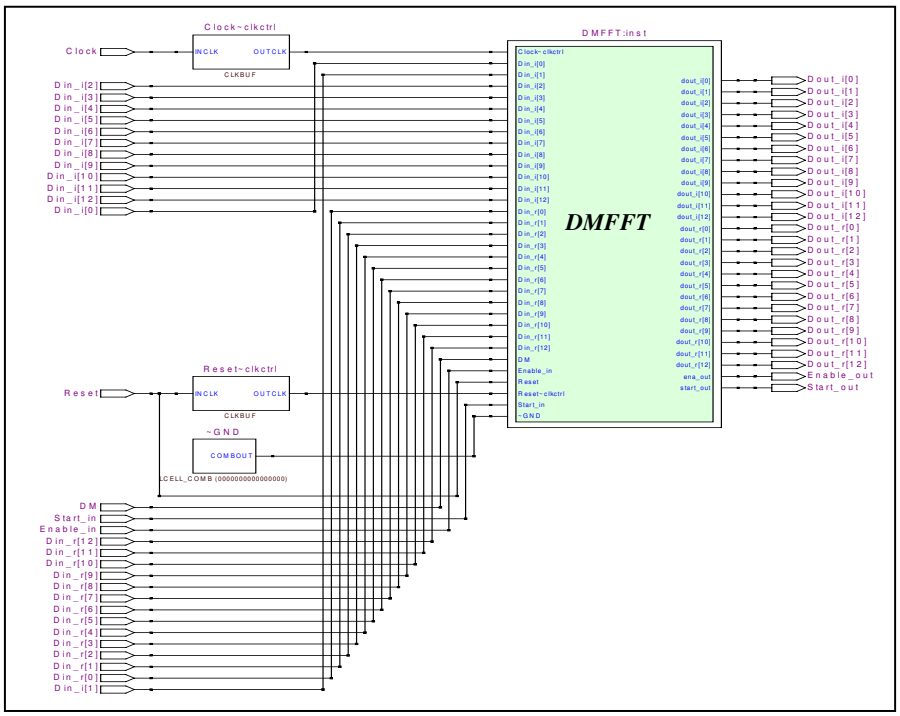

Figure 3.22: Layout of the DMFFT-64 implemented on a Stratix II, EP2S15F484C3

wordlength plays an important role on the accuracy of the computations. This accuracy increases as the wordlength increases. As for the Fermat transforms, they are defined over $GF(257)$ whose symbols are represented by 9 bits.

The measures given in Table 3.4 represent the cost in terms of $ALUTs$, the critical path delay in ns, the memory saving, the gain in terms of $ALUTs$ and the performance-to-cost ratio gain that the DMFFT presents compared to the Veclro operator. This latest measure represents the percentage gain between η_R and η_V. According to these measures,

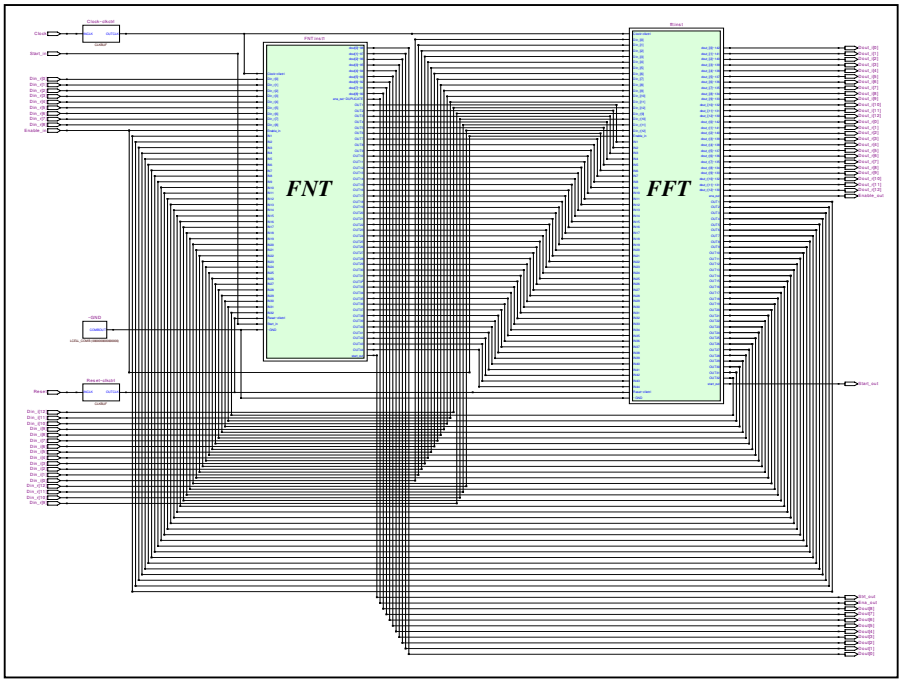

Figure 3.23: Layout of the Velcro FFT/FNT-64 implemented on a Stratix II, EP2S15F484C3

Table 3.4: Comparison between the DMFFT-64 and the Velcro FFT/FNT-64 operator on a Stratix II, EP2S15F484C3

n_c	9	10	11	12	13	14	15	16
Velcro operator	4205 ALUTs 4.86 ns	4768 ALUTs 5.27 ns	5156 ALUTs 5.08 ns	5831 ALUTs 5.46 ns	6064 ALUTs 5.74 ns	6844 ALUTs 5.81 ns	7302 ALUTs 5.79 ns	8143 ALUTs 6.62 ns
DMFFT	3109 ALUTs 4.78 ns	3744 ALUTs 4.97 ns	4112 ALUTs 5.0 ns	4857 ALUTs 5.45 ns	5182 ALUTs 5.76 ns	5975 ALUTs 5.85 ns	6469 ALUTs 5.86 ns	7387 ALUTs 6.65 ns
Memory saving	33 %	31 %	29 %	27.2 %	25.7 %	24.3 %	23 %	21.9 %
ALUT's gain	26 %	21.4 %	20.2 %	16.7 %	14.5 %	12.7 %	11.4 %	9.2 %
η's gain	37.4 %	35 %	27.5 %	20 %	16.7 %	13.9 %	11.5 %	9.7 %

we remark that depending on the wordlength, the DMFFT presents a memory saving around between 20 and 30 %, a gain in ALUTs and performance-to-cost ratio gain that go from 9.2 % up to 26 % and from 9.7 % up to 37.4 % respectively. These gains have their maximum values for $n_c = n = 9$, i.e. when the FNT mode exploits the maximum number of the resources dedicated to the FFT mode. These gains have their minimum

values when $n_c = 16$. In this case, there are 7 bits unused by the FNT mode while in the Velcro operator, 9-bits wordlength is sufficient to implement the FNT.

Table 3.5 represents the implementation measures of the DMFFT-256 and the Velcro FFT/FNT-256 operator. The memory saving is practically the same and the other gains evolve in the same manner of that of DMFFT-64 but with lower values. This is due to the fact that the DMFFT complexity is dominated by that of the FFT and when the transform length increases, the FFT complexity increases rapidly while the FNT complexity remains moderate.

Table 3.5: Comparison between the DMFFT-256 and the Velcro FFT/FNT-256 operator on a Stratix II, EP2S15F484C3

n_c	9	10	11	12	13	14	15	16
Velcro operator	5327 ALUTs 5.02 ns	6079 ALUTs 4.95 ns	6566 ALUTs 5.13 ns	7518 ALUTs 5.45 ns	7885 ALUTs 5.58 ns	8966 ALUTs 5.84 ns	9513 ALUTs 6.1 ns	10770 ALUTs 6.55 ns
DMFFT	4466 ALUTs 4.86 ns	5365 ALUTs 4.9 ns	5911 ALUTs 5.0 ns	6819 ALUTs 5.5 ns	7336 ALUTs 5.63 ns	8546 ALUTs 5.9 ns	9124 ALUTs 6.11 ns	10389 ALUTs 6.58 ns
Memory saving	33 %	31 %	29 %	27 %	25.5 %	24 %	22.8 %	21.9 %
ALUT's gain	16.2 %	11.7 %	9.97 %	9.29 %	7 %	4.68 %	4 %	3.5 %
η's gain	24 %	15.1 %	12.1 %	9.4 %	6.6 %	4.2 %	4.06 %	3.54 %

In order to select the wordlength that can provide the best tradeoff precision-cost, it was necessary to evaluate the influence of the wordlength on the precision or on the accuracy of FFT computations and then illustrate the corresponding result vs the gain in ALUTs. To perform this simulation, we have evaluated by software the FFT using floating-point arithmetic and have compared the results with those obtained with the DMFFT as shown in Fig. 3.24.

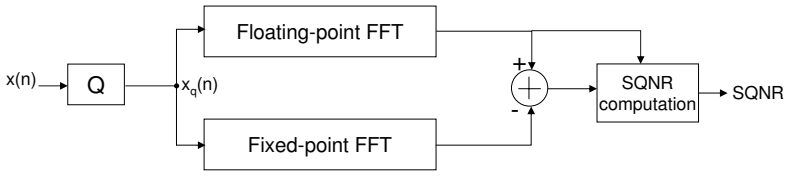

Figure 3.24: Block diagram of the simulation analysis

The output of the conceptual block diagram of Fig. 3.24 represents the Signal-to-Quantization-Noise Ratio

$$SQNR = 10\ log(\frac{E[|S(k)^2|]}{E[|N(k)^2|]}), \tag{3.25}$$

where $E[|S(k)^2|]$ and $E[|N(k)^2|]$ represent the root-mean-square of the useful signal (provided by the floating-point FFT) and the quantization-noise respectively. During simulation, random patterns are generated, quantified and fed into the two FFT blocks. Fig. 3.25 illustrates (from left to right) the SQNR evaluations for FFT-64 and FFT-256 vs the complexity gains for the same transform lengths previously given in Figs. 3.4 and 3.5. As we remark in Fig. 3.25, when the number of bits increases, the SQNR increases whereas the complexity gain in terms of ALUTs decreases.

As a tradeoff between precision and complexity, 13-bit wordlength seems to be the best solution.

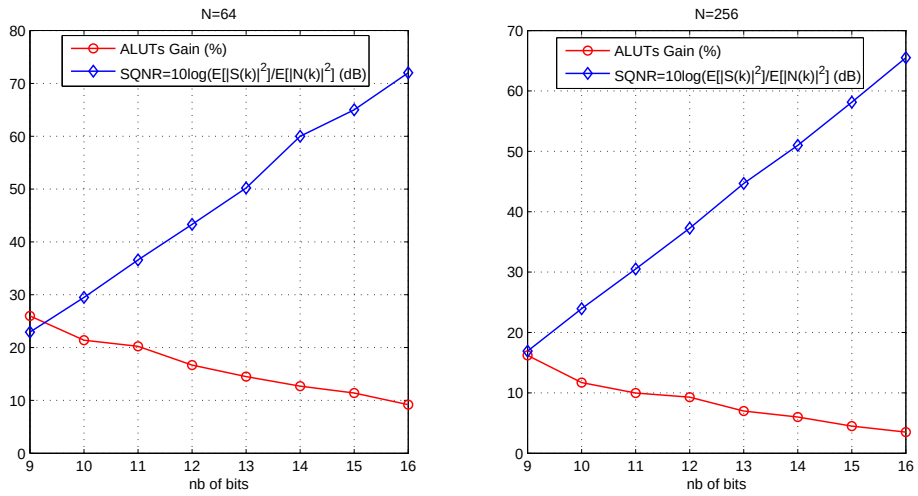

Figure 3.25: The relationship between the wordlength and the gain in ALUTs for N=64 and N=256

3.5 Conclusions

This chapter has investigated the design and FPGA implementation of a reconfigurable DMFFT operator. This design was based on the realization of reconfigurable arithmetical operators (multiplier, adder and subtracter) leading to the DMFFT operator. This operator provides two functionalities: FFT and FNT. During the experimentations, it was highlighted that the implementation of the whole FFT structure for large transform length is not a practical solution with the actual FPGA devices while an FFT-16 consumes 65 % of the STRATIX II devices. Then, another implementation strategy was adopted. The best computing time-area complexity tradeoff was obtained by implementing, for transform length N, $log\ N$ computation stages where each stage is composed of one processing element (known as butterfly), a stage control unit and some memory blocks. With this implementation strategy, the DMFFT and the Velcro FFT/FNT operators were implemented and compared. The implementation results showed, in favor of the DMFFT operator, an important memory saving and gains in terms of ALUTs and in terms of performance-to-cost ratio which vary with the wordlength and the transform length. The DMFFT operator that was built constitutes a CO candidate to integrate a SDR system intending to support

several standards where the complex Fourier transform and the RS coding are required. The RS codes that can be teated with this operator are the RS codes defined over $GF(F_t)$.

The next step is to extend the study towards the classical RS codes defined over $GF(2^m)$ and design an operator which, besides the two functionalities FFT and FNT, provides also the finite field Fourier transform defined over $GF(2^m)$. Next chapter will discuss the evolution of the Dual mode operator DMFFT towards a triple mode operator.

Chapter 4

DMFFT and FFT over $GF(2^m)$: from a dual to a triple mode FFT operator

Contents

4.1 Introduction

The analysis presented in Chapter 2 has suggested the use of RS codes defined over $GF(F_t)$ by taking advantages of the use of FNT to perform the longest steps of the decoding process. This FNT functionality is provided by the DMFFT operator as seen in Chapter 3. Nevertheless, RS codes implemented in the various actual standards are codes defined over $GF(2^m)$. Then, the inclusion of the finite field Fourier transform defined over $GF(2^m)$ as an extra functionality to be provided by our designed CO is a necessity to be on actual standardization's trend.

From error correcting capability point of view, we have shown that the performances of the specific RS codes defined over $GF(F_t)$ are as good as those of RS codes defined over $GF(2^m)$. Starting from this hypothesis and from the main objective aiming to insert the

complex FFT in the channel coding, in Chapter 3 we have designed a common operator able to provide two functionalities: complex Fourier transform and Fermat transform. The resulting operator can perform any communication task requiring the complex Fourier transform and contributes efficiently in the decoding process of RS codes defined over $GF(F_t)$ thanks to the Fermat transform.

The hardware design realized in Chapter 3 constitutes a serious and concrete attempt to design a reconfigurable operator by exploiting the hardware resources of complex FFT. This attempt showed the feasibility of such an operator and led to a dual mode FFT whose performances are more efficient than those of the Velcro FFT/FNT.

From hardware complexity point of view, we should point out that the mathematical structures used in $GF(2^m)$ are still simpler than those used in $GF(F_t)$. That is, a modulo 2 addition (i.e. XOR operation) is simpler than a modulo F_t addition. Then, a RS encoder/decoder over $GF(2^m)$ compared to a RS encoder/decoder over $GF(F_t)$ should be less complex. However, the use of RS codes over $GF(F_t)$ presents an advantage in the presence of a pre-implemented complex FFT while the FFT and FNT structures are very similar.

Starting from these considerations, we have investigated the realization of an operator able from one hand to take advantage of the complexity gain offered by the DMFFT and from the other hand to meet the requirements of the current standards. For this, we proposed the realization of a Triple Mode Velcro FFT operator (TMVFFT) able to provide three functionalities: complex Fourier transform (FFT-$\mathbb{C}$), Fermat Number Transform (FNT) and finite field Fourier transform over $GF(2^m)$ (FFT-GF2). This operator, able to support three processing contexts via a simple parameter adjustment, could be a CO for a SWR system intended to receive several communication standards.

This chapter begins by presenting the general architecture of the TMVFFT operator with its Velcro approach and thereafter gives some suggestions to upgrade this operator towards a single and reconfigurable Triple Mode FFT operator (TMFFT).

4.2 TMVFFT operator architecture

In the light of the above discussion, we present in this section the architecture of the TMVFFT. This architecture seems to be obvious, but the idea is to start from this architecture and reach a more efficient architecture.

Fig. 4.1 shows the Velcro approach of the triple mode FFT where TM is a control signal used to select the functioning mode to be performed. This architecture consists of two self-contained operators: DMFFT and FFT-GF2. This approach will generally provide the best performance but at a high circuit complexity. We can minimize the circuit complexity by going towards a reconfigurable approach where the two architectures are combined and therefore the same hardware resources can be reused for both architectures. Nevertheless, this reusability of the hardware resources should not affect the system's performance. Thus, we need to find the right level of complexity at which we reach a performance level acceptable for the set of practical applications involved in the SWR system functionality. This is the level that gives the optimum performance-cost trade-off.

Starting from these considerations, we propose in the next section two scenarios allowing the the evolution of the TMVFFT.

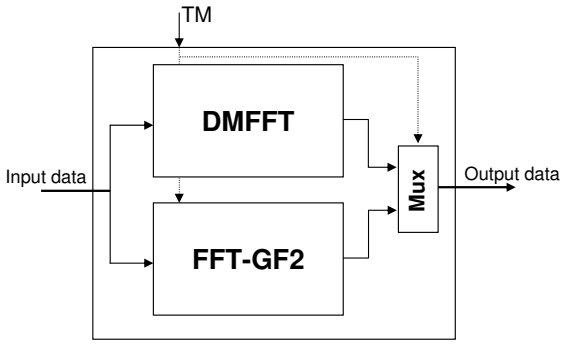

Figure 4.1: The TMVFFT architecture

4.3 Scenarios for the evolution of the TMVFFT architecture

In this section, we describe the two scenarios we propose. The first one leads to an optimal use of the TMVFFT while the second serves as a road-map to progress from the TMVFFT of Fig. 4.1 towards a Triple Mode FFT (TMFFT) operator shown in Fig. 4.2. The practical realization of the first scenario is described in the following sections. As for the second scenario, we give some guidelines that could aid the achievement of this scenario envisaged as a perspective of this work.

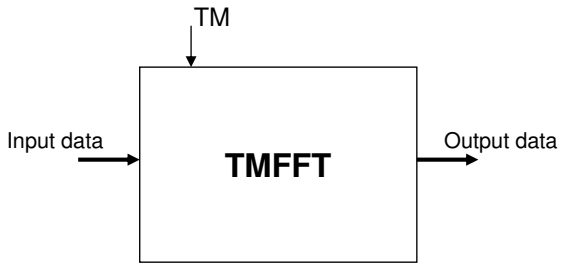

Figure 4.2: The TMFFT architecture

- **Scenario 1: Speeding up the computing time**. This scenario supposes that an operator can be reused during a given time slot if its computing time is at least twice faster than the existing alternative operators. With this scenario, the architecture of the triple mode operator is still a Velcro architecture.
- **Scenario 2: Combination of the two operators (DMFFT and FFT-GF2).** This scenario is based on the combination of the DMFFT and FFT-GF2 operators. The aim is to implement the FFT-GF2 using the arithmetical resources of the DMFFT. The resulting operator is a reconfigurable triple mode operator (TMFFT) able to provide transforms over three different domains: $\mathbb{C}$, $GF(F_t)$ and $GF(2^m)$. The switching from one mode to another can be performed by simple parameter adjustments.

4.4 Scenario 1: optimal use of the TMVFFT via the upgrade of the DMFFT

In general, a CO intending to replace two operators (operator 1 and operator 2) by providing their functionalities, could efficiently achieve this mission if it is capable to perform the task of each operator in half the time needed by each one of the two operators. Then, such use of the CO does not affect the performances of the concerned system. Fig. 4.3 illustrates this principle. As shown in this figure, the CO is considered as twice faster than each of the two operators. Consequently, if operator 1 task (Op1 task) and operator 2 task (Op2 task) are initially performed with T_c each, the CO is able to perform both Op1 task and Op2 task in T_c.

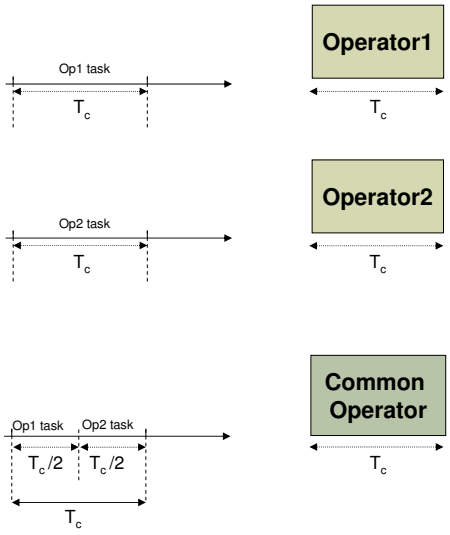

Figure 4.3: Task diagram of Common Operator

In this section, we consider the evolution of the TMVFFT architecture by upgrading the functioning efficiency of the DMFFT as shown in Fig. 4.3. As discussed in the previous chapter (section 3.3.4), the type of RAM plays an important role in the processing time of the RPE. Let us start with the DMFFT architecture of Fig. 4.4. This figure shows only a part of the architecture that represents the RPE, RAM blocks and the data routing buses.

The architecture of Fig. 4.4 employs RAM-1-port blocks (one write port and one read port). Let us consider n the number of clock cycles needed to perform the FFT (or the FNT) functionality. The corresponding computing time to provide a transform sequence is equal to $t = n\ T_c$. By employing a RAM-3-Port (one write port and two read ports), the architecture of Fig. 4.4 evolves towards the architecture of Fig. 4.5.

This architecture is more efficient in terms of computing time which in this case is equal to $t' = (n/2)\ T_c$. By this way, during a time t, the DMFFT operator is able to provide both FFT and FNT functionalities.

For example, with the architecture of Fig. 4.5, the DMFFT operator can compute two sequences of Fermat transform during a time t. This allows the DMFFT, used in the RS decoding process, to replace the syndrome computation block and the Chien search circuit

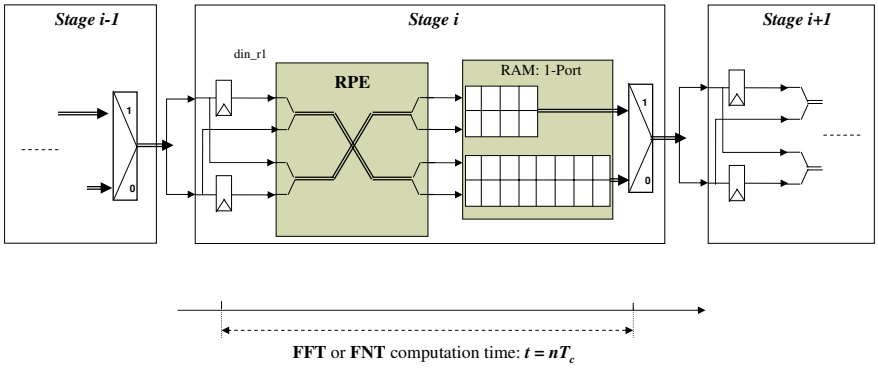

Figure 4.4: The DMFFT architecture with RAM 1-Port

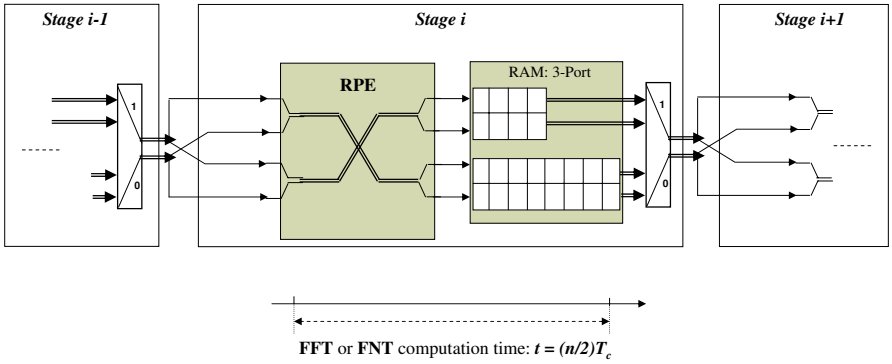

Figure 4.5: The DMFFT architecture with RAM 3-Port

and provide their functionalities during the same time t and without affecting the original time of the decoding process.

This approach can be extended and a computing time $t'' = (n/4)\ T_c$ can be attained by implementing two RPEs per stage, i.e. duplication of the architecture of Fig. 4.5 within the global DMFFT architecture previously shown in chapter 3 (Fig. 3.15). The order of this evolution can be determined according to the need of the DMFFT functionalities required in the SWR system. We should then predetermine the right need of the DMFFT functionalities in SWR system and base the DMFFT structure on that need while taking into account the hardware complexity that should not exceed certain limits.

4.5 Scenario 1: optimal use of the TMVFFT via the upgrade of the FFT-GF2

In this section, based on the principle illustrated in Fig. 4.3, the design of a new FFT-GF2 architecture leading to an efficient use of FFT-GF2 implemented in Fig. 4.1 will be described.

In the $GF(2^m)$ domain, the best algorithm to compute the finite field Fourier transform has been introduced by Wang and Zhu [102]. This algorithm allows the reduction of a number of multiplications from $O(n^2)$ to $O(\frac{1}{4}n(log\ n)^2)$, where n is the transform length. A VLSI chip was designed by the authors for the simple case of $GF(2^4)$ by using the technology $\lambda = 1,5\mu m$ CMOS. The required computing time is $(n+1)T_c$ where T_c is the time required for doing one multiplication over the underlying field. This long computing time restricts the use of the FFT circuit that should be allotted to only one task during a given time slot.

Starting from SWR considerations, the idea is then to design a FFT-GF2 with performances that can evolve this operator into a CO. To reach this objective, we proceed in a way which allows us to speed up the transform computing time while taking into account the hardware complexity. The solution is the use of cyclotomic FFT. This solution is based on representing a polynomial as a sum of linearized polynomials. This approach was firstly suggested in [103] and then generalized in [104]. Based on this approach, an attractive method to compute Fourier transform over $GF(2^m)$ is proposed in [105]. In that paper, the authors compared their proposed method to main related works existing in the literature (as Horner's method and Goertzel's algorithm and some others algorithms) and showed, throughout a complexity study, that their suggested method requires a lower number of multiplication and addition operations. As mentioned, their suggested method is more efficient for short FFT lengths ($n \leq 255$). For large transform lengths, more efficient FFT algorithms are preferred [102] [106] [107].

The method proposed in [105] is based on the factorization of the transform matrix into a product of binary circulant matrix and a diagonal circulant matrix. This method takes advantages of the cyclotomic decomposition of an original polynomial into a sum of linearized polynomials and then provides a matrix form of the transform computations. By carefully studying this method, we have noticed that the cyclotomic decomposition of polynomials used to reduce the number of operations (multiplications and additions) can be equally exploited to get a set by set treatment of the input sequence. This means that instead of sequential processing (symbol by symbol) by the FFT circuit of the input data used in [102], we propose to design a FFT circuit taking advantages of the cyclotomic decomposition of the input sequence to process the data set by set according to this decomposition. This leads to a flexible FFT-GF2 able to perform its transform computation with a time $t' \leq \frac{t}{2}$, where t refers to the computing time required by the algorithm [102] which will be used in the rest of this section. According to [102], t is expressed as $t = (n+1)T_c$.

The architecture we aim to design, which is based on the method proposed by [105], could make the FFT-GF2 a CO thanks to the reduced computing time at which this CO will operate. This leads to get an optimal exploitation of resources and find a best way to re-use some hardware modules without affecting the system's performances.

In [108], the application of the cyclotomic FFT to the RS decoding is discussed where a reduction complexity of the syndrome evaluation is shown. In [104], an improved algorithm for finding roots of polynomials over finite fields is presented.

In this work, we consider the use of the FFT to perform both steps: syndrome computation and Chien search in the RS decoding process. This architecture can be efficiently used to implement the Gao algorithm [109] whose main operations can be performed using the Fourier transform and the method proposed in [110]. It can also be used in any other

application necessitating the use of FFT-GF2 and specifically when fast computations and high throughput rate are required.

For the description of the method introduced in [105] and its implementation we will proceed as follows. In section 4.5.1 we summarize the method by defining some of its key concepts. For more details on some basic definitions and computation developments, the lecturer will be referred to Appendix C. Section 4.5.2 gives the hardware interpretation of the method and section 4.5.2.4 describes the proposed hardware architecture. The performance evaluation of the designed architecture is done by considering its implementation on FPGA devices and comparing it with the VLSI implementation of the algorithm proposed in [102]. Some implementation figures (i.e. critical path delay and hardware complexity) are given in section 4.5.2.5. Finally, a conclusion ends this chapter.

4.5.1 Cyclotomic algorithm for the finite field transform computation

Definition 4.2.6. A polynomial L(y) over $GF(2^m)$, where m is an integer, is a linearized polynomial if

$$L(y) = \sum_i L_i y^{2^i}, \qquad L_i \in GF(2^m). \tag{4.1}$$

It can be easily shown that $L(y)$ satisfies $L(a+b) = L(a) + L(b)$. This property leads to the following theorem, presented here in a slightly modified form with respect to that defined in [57].

Theorem.1. *Let $x \in GF(2^m)$ and let $\beta = (\beta_0, \beta_1, ..., \beta_{m-1})$ be a basis of the field. If*

$$x = \sum_{i=0}^{m-1} x_i \beta_i, \quad \textit{then} \quad L(x) = \sum_{i=0}^{m-1} x_i L(\beta_i), \tag{4.2}$$

where $x_i \in GF(2)$. Let us consider the cyclotomic cosets C_{k_s} modulo $n = 2^m - 1$ over $GF(2)$:

$$\begin{aligned} C_0 &= \{0\}, \\ C_{k_1} &= \{k_1, k_1 2, k_1 2^2, ..., k_1 2^{m_1 - 1}\}, \\ &\vdots \\ C_{k_l} &= \{k_l, k_l 2, k_l 2^2, ..., k_l 2^{m_l - 1}\}, \end{aligned}$$

where $k_s \equiv k_s 2^{m_s} \mod n$. Then any polynomial

$$f(x) = \sum_{i=0}^{n-1} f_i x_i, \qquad f_i \in GF(2^m)$$

can be decomposed as

$$f(x) = \sum_{i=0}^{l} L_i(x^{k_i}), \tag{4.3}$$

where

$$L_i(y) = \sum_{j=0}^{m_i - 1} f_{k_i 2^j \bmod n}\ y^{2^j}.$$

In fact, equation (4.3) represents a way of grouping indices $0 \leq i < n$ of $f(x)$ terms into cyclotomic cosets : $i \equiv k_s 2^j \ mod\ n$. Obviously, this decomposition is always possible. Note that the term f_0 can be represented as $L_0(x^0)$, where $L_0(y) = f_0 y$.

The cyclotomic FFT algorithm described in [105] is based on representing the polynomial $f(x)$ as a sum of linearized polynomials (cyclotomic decomposition of the polynomial) as we will explain in the following.

Let us consider $f = f_0, f_1, ..., f_{n-1}$, $n \mid (2^m - 1)$, a vector of n $GF(2^m)$ elements. The Fourier transform of f is denoted by F_j with

$$F_j = \sum_{i=0}^{n-1} f_i \alpha^{ij}, \qquad j = 0, ..., n-1, \tag{4.4}$$

and α is the primitive element of order n in the field $GF(2^m)$. The vector $f(x)$ can be expressed by a polynomial

$$f(x) = \sum_{i=0}^{n-1} f_i x^i, \qquad i = 0, ..., n-1. \tag{4.5}$$

According to equation 4.3, $f(x)$ can be expressed as

$$f(\alpha^j) = \sum_{i=0}^{l} L_i(\alpha^{jk_i}). \tag{4.6}$$

It is known [57], that α^{k_i} is a root of a minimal polynomial of degree m_i ($m_i | m$) and thus belongs to a subfield $GF(2^{m_i})$. Thus all the values $(\alpha^{k_i})^j$ lie in $GF(2^{m_i})$ and so they can be decomposed in some basis $\beta_i = (\beta_{i,0}, ..., \beta_{i,m_i-1})$ of the subfield and represented as:

$$\alpha^{jk_i} = \sum_{s=0}^{m_i-1} a_{ijs} \beta_{i,s}, \quad a_{ijs} \in GF(2). \tag{4.7}$$

Then each linearized polynomial can be evaluated at the basis points of the corresponding subfield by the formula

$$L_i(\beta_{i,s}) = \sum_{p=0}^{m_i-1} \beta_{i,s}^{2^p} f_{k_i 2^p}, \tag{4.8}$$

and according to equation 4.3,

$$\begin{aligned} F_j = f(\alpha^j) &= \sum_{i=0}^{l} \sum_{s=0}^{m_i-1} a_{ijs} L_i(\beta_{i,s}) \\ &= \sum_{i=0}^{l} \sum_{s=0}^{m_i-1} a_{ijs} \Big(\sum_{p=0}^{m_i-1} \beta_{i,s}^{2^p} f_{k_i 2^p} \Big), \end{aligned} \tag{4.9}$$

where $j \in [0, n-1]$.

This equation can be represented in matrix form as

$$\mathbf{F=ALf}, \tag{4.10}$$

where $F = (F_0, F_1, ..., F_{n-1})^T$, $f = (f_0, f_{k_1}, f_{k_1 2}, f_{k_1 2^2}, ..., f_{k_1 2^{m_1-1}}, ..., f_{k_l}, f_{k_l 2}, ..., f_{k_l 2^{m_l-1}})^T$ are vectors consisting of some permutations of elements F_j and f_i, respectively, A is a matrix with elements $a_{ijs} \in GF(2)$ and L is a block diagonal matrix with elements $\beta_{i,s}^{2^p}$.

It is possible to choose the same basis for all linearized polynomials of the same degree m_i in equation (4.3) to obtain a very small amount of different blocks in the matrix L and consequently simplify the problem of constructing a fast algorithm for multiplication of the matrix L by a vector f over $GF(2^m)$. Moreover, if the normal basis β_i is used in equation (4.9), all the blocks of the matrix L are circulant matrices. Then, the multiplication by this matrix can be performed by a cyclic convolution of degree m_i. Indeed, there are efficient algorithms which can be applied to perform the cyclic convolution efficiently [43] and then reduce the computation complexity of the FFT.

The cyclotomic FFT algorithm can be considered as two principal stages of multiplications: (i) multiplication of the block diagonal matrix L by the original vector f and (ii) multiplication of the binary matrix A by the resultant vector $S = Lf$. The matrix L can be decomposed into $(l+1)$ circulant matrices according to the cyclotomic decomposition of f, where $(l+1)$ represents the number of cyclotomic cosets. Then, the mutiplication Lf can be performed by a set of cyclic convolutions. The development of this step is given in Appendix C. Taking into account the results of such a development, Equation 4.10 can be rewritten as

$$\mathbf{F=AQ(B.(PF))}, \tag{4.11}$$

where Q is the binary block diagonal matrix of combined post-additions for $l+1$ cyclic convolutions, B is the combined vector of constants, and P is the binary block diagonal matrix of combined pre-additions. Further details can be found in Appendix C.

4.5.2 Hardware interpretation

By turning the algorithm into a hardware language we design an architecture consisting of four stages with a parameterized and a modular architecture for the fourth stage. It is this last stage which determines computing time and the throughput rate of the global architecture.

In this section we reformulate the algorithm presented in [105] to be adapted to hardware implementation. Firstly, we give a cyclotomic decomposition of various $GF(2^m)$ and we show the influence of this decomposition on the FFT structure and then on the computing time and global throughput rate.

Secondly, we introduce different steps of computation which will be adapted to the hardware interpretation of the FFT algorithm, and to better understand the procedure we give some examples in $GF(2^4)$. Next, we present how the mathematical and matrix equations can be expressed as logic gates (AND, XOR) and some multipliers over $GF(2^m)$.

4.5.2.1 Cyclotomic decomposition

The elements of $GF(2^m)$ can be grouped into cyclotomic cosets C_{k_s}. These cosets do not all have the same size, so the processing blocks that the data have to cross will not be the same. That is, the cosets of the same size will be processed by the same processing blocks independently of the others which, in turn will be processed by other computational blocks. The structure of the FFT circuit is then based on the cyclotomic decomposition of the $GF(2^m)$ symbols constituting the time sequence. We have written a Matlab program to calculate the different cyclotomic cosets for several GF. Table 4.1 shows these cosets with their corresponding GF.

As shown in Table 4.1, each GF consists of three classes of cosets : (i) a coset that contains only one element (ii) cosets which contain more than one element and (iii) cosets with the highest possible number of elements. We observe that the number of cosets containing the highest number of elements are always more large as compared to other cosets. Consequently, the number of cycles needed to perform the FFT computation is determined by this latest class of cosets.

Table 4.1: The cyclotomic cosets for different GF.

GF	Cyclotomic cosets
$GF(2^3)$	1 $C\{1\}$, 2 $C\{3\}^*$
$GF(2^4)$	1 $C\{1\}$, 1 $C\{2\}$, 3 $C\{4\}$
$GF(2^5)$	1 $C\{1\}$, 6 $C\{5\}$
$GF(2^6)$	1 $C\{1\}$, 1 $C\{2\}$, 2 $C\{3\}$, 9 $C\{6\}$
$GF(2^7)$	1 $C\{1\}$, 18 $C\{7\}$
$GF(2^8)$	1 $C\{1\}$, 1 $C\{2\}$, 3 $C\{4\}$, 30 $C\{8\}$

* n $C\{m\}$: n sets of m elements

4.5.2.2 The different steps for the FFT algorithm

Here, we divide the cyclotomic FFT algorithm into four steps that will help the designer to adequate the theoretical approach to the hardware architecture.

Step 1. Determine the cyclotomic cosets (their number and the size of each one) of the given GF, choose the basis of each class of cosets and decompose each element of $GF(2^m)$ with respect to the normal basis. For example in $GF(2^4)$, we have five cyclotomic cosets:

$C_0 = \{f_0\}$,
$C_1 = \{f_1, f_2, f_4, f_8\}$,
$C_3 = \{f_3, f_6, f_{12}, f_9\}$,
$C_5 = \{f_5, f_{10}\}$,
$C_7 = \{f_7, f_{14}, f_{13}, f_{11}\}$,

where f_i represents the data to be transformed. The basis for C_1, C_3, C_7 is $(\beta, \beta^2, \beta^4, \beta^8)$, where $\beta = \alpha^3$ and α an element of $GF(2^4)$ which can be decomposed as $\alpha = \beta + \beta^8$. For C_5, we can choose as basis (γ, γ^2) where $\gamma = \alpha^5$.

Step 2. Develop $f(\alpha^i)$, $i = 0, ..., n-1$, according to $f(\alpha^j) = \sum_{i=0}^{l} L_i(\alpha^{jk_i})$ and deduce the coefficients "a_{ijs}" of the matrix A. For example, in $GF(2^4)$ l=4, k_0=0, k_1=1, k_2=3, k_3=5, k_4=7 and $f(\alpha^1) = L_0(\alpha^0) + L_1(\alpha) + L_2(\alpha^3) + L_3(\alpha^5) + L_4(\alpha^7)$
$= L_0(1) + L_1(\beta) + L_1(\beta^8) + L_2(\beta) + L_3(\gamma) + L_4(\beta) + L_4(\beta^2) + L_4(\beta^4)$,
where we can deduce the coefficients "a_{ijs}"=(1 1 0 0 1 1 0 0 0 1 0 1 1 1 0). The same procedure applied to the others $f(\alpha^i)$ leads to obtain the matrix A shown in Appendix C.

Step 3. Develop $L_i(y)$, $i = 0, ..., l$, according to (5). As shown in [105], $L_i(y)$ represents a cyclic convolution between the normal basis and the f_i.

Example: in $GF(2^4)$, by applying the algorithm for four-point cyclic convolution described in ([43], chapter 11) we can write the matrix product $S = Lf$ as

$$S = Lf = Q\Big(\Big(R\ \beta_i^T\Big).\Big(P\ f_i\Big)\Big) =$$

$$\begin{bmatrix} 1&0&1&1&0&0&0&0&1 \\ 1&0&1&0&1&0&0&1&0 \\ 1&1&0&1&0&0&1&0&0 \\ 1&1&0&0&1&1&0&0&0 \end{bmatrix} \left(\begin{bmatrix} 1&1&1&1 \\ 1&1&0&0 \\ 0&0&1&1 \\ 0&1&0&1 \\ 1&0&1&0 \\ 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \begin{bmatrix} \beta \\ \beta^8 \\ \beta^4 \\ \beta^2 \end{bmatrix} . \begin{bmatrix} 1&0&0&0 \\ 1&0&1&0 \\ 1&0&1&0 \\ 1&0&0&1 \\ 1&1&0&0 \\ 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_4 \\ f_8 \end{bmatrix} \right) \tag{4.12}$$

Step 4. Apply the same interpretation done in step 3 to the coset C_5. For this coset, we obtain a two-point cyclic convolution.

At the end of step 4 we have all the elements necessary to implement the equation $F = ALf = AS$. The hardware implementation of these matrices product is described in the next subsection.

4.5.2.3 Interpretation

Here, we consider the hardware interpretation of the different steps described above to subsequently present the implementation of 15-point FFT over $GF(2^4)$. As previously mentioned, this field consists of five cyclotomic cosets $(C_0, C_1, C_3, C_5, C_7)$. The same principle can be applied to other FFT lengths. The architecture we propose is based on the cyclotomic decomposition and is composed of two units. The first one is the principal unit dedicated to process, set by set, the data symbols which belong to the third class of cosets (C_1, C_3, C_7). It is the processing of data by group which allows to decrease the computing time and leads to a high throughput rate. The second one is the additional unit

used to process the data symbols which belong to the second class of cosets C_5. The only element of the first class of cosets which represents the first symbol of the "time-domain" data is processed at the last stage of computation.

We start the hardware interpretation of the algorithm from step 2 of the previous subsection. The inputs of the FFT circuit are the f_i which are grouped according to the cyclotomic decomposition and processed consecutively (C_1,C_3 and then C_7). So, taking these f_i's as inputs we tackle the implementation of equation 4.11 from right to left. Then, the first stage of the architecture which implements the matrix product $V_1 = Pf_i$ (V_1 is a column vector) is composed of XOR gates that provide the eight modulo 2 additions of the coset elements. The result of the second matrix product $V_2 = (R\beta_i^T)$ is also a column vector whose elements belong to $GF(2^4)$ which subsequently will be multiplied by V_1. Indeed, V_2 can be predetermined and then implemented as the multiplicands of the multipliers which will perform the vector multiplication $[V] = [V_1].[V_2]$. At this second stage only eight multipliers are needed since the first element of V_2 is always equal to 1 (it is the sum of the normal basis). The third stage consists of the multiplication of the matrix Q by the vector V. This multiplication can be implemented by XOR gates which perform the four (number of rows of the matrix Q) combinations of the V elements.

Once the vector S is computed, we can tackle the implementation of the last stage (stage 4) which represents the most important task. Stage 4 will perform the following multiplication :

$F = A\big(\big[C_{p0}\ C_{p1}\ C_{p3}\ C_{p5}\ C_{p7}\big]^T\big)$, where $C_{p0} = f_0$ and C_{p1}, C_{p3}, C_{p5}, C_{p7} are the processed cosets corresponding to C_1, C_3, C_5, C_7 respectively. Vector F is composed of the 15 frequency components. Let us consider, as an example, the computation of the component F_1. Bearing in mind that the Cpi (i=1,3,7) are generated consecutively, each one of these cosets will be multiplied by the corresponding set of elements of each one of the 15 rows of the matrix A. Firstly, we consider that the coset C_{p5} is processed in parallel by another block. Then F_1 is produced after 3 clock cycles where each cycle corresponds to 4 XOR additions and 15*4 AND-multiplication in parallel of each processed coset by its corresponding set in matrix A. After each multiplication the result will be stored in a register to be added to the next multiplication (and additions) result. Once the third multiplication is achieved, the total is added to the result of the processing of C_5 (C_{p5}) and then the sum is added to f_0 to finally obtain the component F_1.

In this approach we have considered the implementation of 15 cells (in stage 4) that process the data in parallel what reduces the computing time and produces a high throughput rate. We can reduce the complexity by executing repeatedly the functionality of the implemented cells. This issue will be discussed in more details in the next section. In Appendix C, we describe in details the development of theoretical computations of cyclotomic FFT-15.

4.5.2.4 Hardware architecture

Starting from the decomposition of the FFT computation into four steps and their hardware interpretation described in the previous section, now we present the FFT architecture and we discuss the different ways to implement the last stage of the proposed architecture. Fig. 4.6 shows the FFT-15 architecture that operates at the minimum computing time and produces the highest throughput rate over $GF(2^4)$. Fig. 4.7 shows in details

the internal architecture where the wired logic circuits, the ROM and the control unit of *stage* 4 are exhibited. As expected, this architecture consists of four stages. The first stage is composed of five 4-bit-XOR gates. These gates are interconnected according to the matrix S. The second stage is composed of eight multipliers whose multiplicands are predetermined and implemented. The third stage consists of ten XOR gates implemented according to the matrix Q.

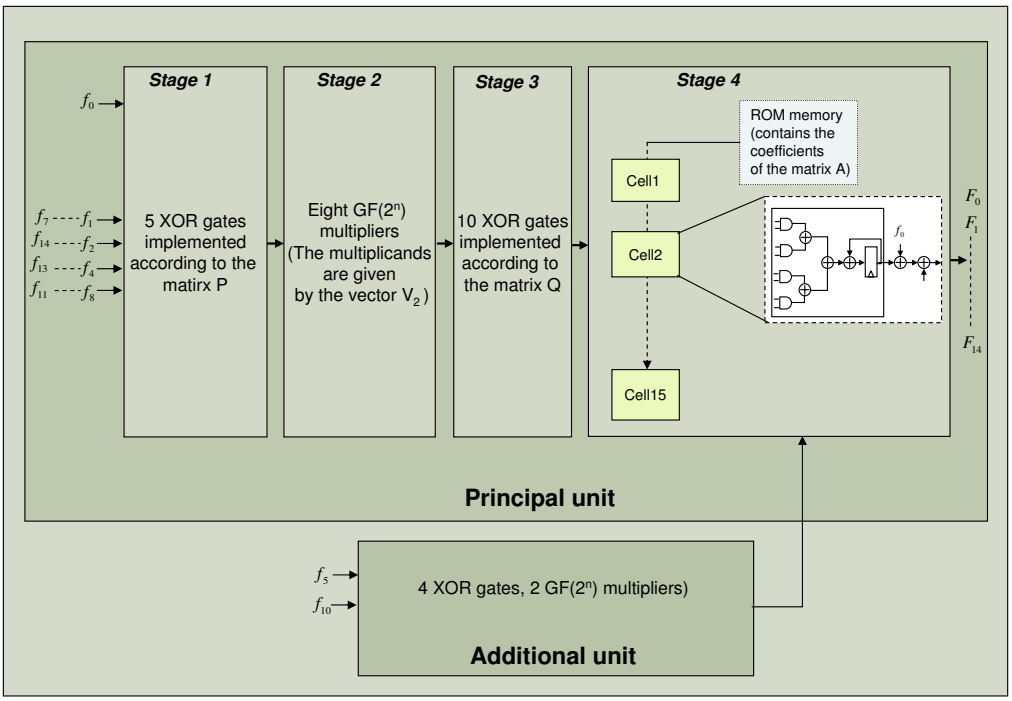

Figure 4.6: The cyclotomic FFT architecture

According to the cyclotomic decomposition of the data in a given GF, there are several choices of implementation of the last stage that consequently lead to different computing time values. In the architecture of Fig. 4.6, the last stage is composed of the maximum number of cells and the symbols of the coset C_5 are processed by an additional unit. We can choose to process this latest coset with the principal unit but this will be to the detriment of the increase of the clock cycle number (needed to compute the transform), i.e. the decrease of the throughput rate. In this case the multiplicands of the multipliers of the second stage have to be updated with the corresponding values.

As shown in Fig. 4.7, each cell of stage 4 consists of 6 XOR gates, 4 AND gates and a shift register. The AND gates are piloted by the coefficients of the matrix A. These coefficients are stored in ROM memories and generated synchronically with the processing of the different cyclotomic cosets.

Indeed, the various possibilities of the fourth stage implementation are directly related to the cyclotomic decomposition previously shown in Table 4.1. Depending on the underlying field, the computing time t' can be expressed as $t' =^{(1)} \text{ceil}(n\frac{N_s}{N_c}).T_c$, where N_c

[1] y=ceil(x) rounds x to the nearest integer greater than or equal to x

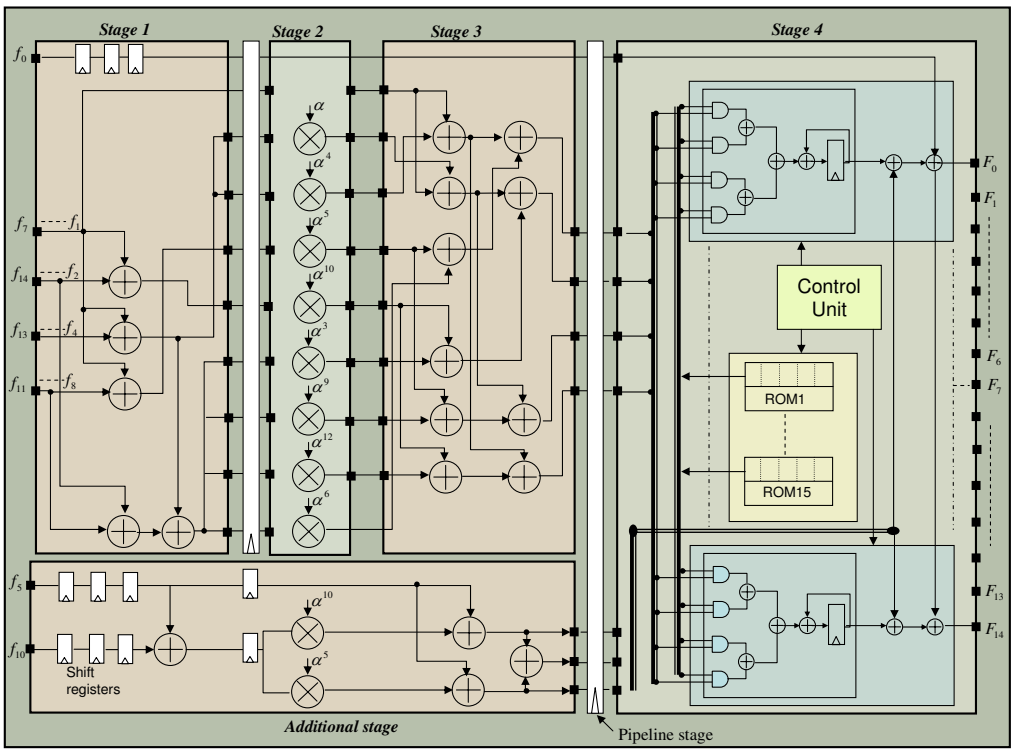

Figure 4.7: The cyclotomic FFT internal architecture

represents the number of cells implemented in stage 4, and N_s is the number of cosets that have the higher possible number of elements. The throughput rate (R) can be expressed as $R = \frac{n}{t'} f_c$, where $f_c = \frac{1}{T_c}$ is the system clock frequency.

Table 4.2 shows the different computing time with the corresponding amount of cells. In fact, according to the importance given to one of the two metrics (hardware complexity and execution speed), the designer can select his appropriate structure among different choices.

As shown in Table 4.2, we have a FFT architecture which at least operates twice as fast and can go up to 8 times faster than the architecture presented in [102]. Obviously, at this very high speed, there will be a price to pay, that is the increase of the amount of logic blocks necessitated by the realization of this architecture. Then, a comparison between the number of multipliers and adders (XOR gates) required by the two architectures is necessary. Indeed, in [102], the author shows that over $GF(2^m)$, $\frac{1}{2}m(m+1)$ multipliers and the same number of adders are required to implement the algorithm. As for our proposed architecture, the number of multipliers required is equal to $2m + 2(m_{C2} - 1)$, where m_{C2} is the maximum number of elements of the cosets which belong to the second class (for example in $GF(2^8)$, $m_{C2} = 4$).

The number of XOR gates can be estimated as the number of XOR and AND gates required by stage 4 while stage 1 and stage 3 have a very simple structure compared to stage 4. Then this number of gates is proportional to $N_c.n_{cg}$, where n_{cg} is the number of gates necessitated by one cell.

Let us consider the case of $GF(2^4)$ to compare the required amount of logic gates. For our architecture and at a computing time equal to $3T_c$, 10 ($2m+2(m_{C2}-1) = 8+2 = 10$) multipliers and 60 AND gates + 109 XOR gates are required. The number of AND and XOR gates can be reduced to 32 AND + 67 XOR if we choose to run out the transform at $t' = 6\ T_c$. The architecture proposed in [102] requires 10 multipliers, 10 XOR and a control block which is needed in each stage to test the degree of polynomial. The conclusion we can draw from the above comparison is that our architecture needs $2m+2(m_{C2}-1)$ multipliers, that is in some $GF(2^m)$ inferior to $\frac{1}{2}m(m+1)$, a large amount of XOR and AND gates and a very small ROM memory but without any control blocks. A counter modulo m is only needed in stage 4 to initialize the memory addresses. The architecture presented in [102] needs less XOR gates but needs some control blocks and operates at a fixed execution time $t = (n+1)T_c$ while our architecture operates at various speeds allowing very fast computing time which can down to $t' = \frac{t}{8}$. Note that T_c is always considered as the time required to perform a $GF(2^m)$ multiplication. The $GF(2^m)$ multiplier implemented in this work is the one proposed by [111].

Table 4.2: Computing time table

GF	**Computing time** (architecture of [102])	nb. of cosets N_s	nb. of cells N_c	**Computing time** (proposed architecture)
$GF(2^4)$	16 T_c	3	8	6 T_c
			15	3 T_c
$GF(2^5)$	32 T_c	6	16	12 T_c
			31	6 T_c
$GF(2^6)$	64 T_c	9	32	18 T_c
			63	9 T_c
$GF(2^7)$	128 T_c	18	64	36 T_c
			127	18 T_c
$GF(2^8)$	256 T_c	30	64	120 T_c
			128	60 T_c
			255	30 T_c

4.5.2.5 FPGA implementation

To evaluate the complexity of the proposed FFT architecture, we consider its implementation on Altera's STRATIX-II FPGA. The performance of this FPGA-based architecture, is determined by evaluating its area complexity and critical path delay. The area complexity can be represented by the number of logic units necessitated by the FPGA implementation while the critical path delay is represented by the clock cycle delay ($T_c = \frac{1}{f_c}$).

Table 4.3: FFT-15 (over $GF(2^4)$) implementation results on STRATIX II, EP2S15F484C3 Device

Computing time	N_c	Area	T_c
$t' = 3\ T_c$	15	343 ALUTs	2 ns
$t' = 6\ T_c$	8	203 ALUTs	2 ns

We have written a synthetizable VHDL code[(2)] of the different stages of the architecture. The pipeline stages are employed as shown in Fig. 4.7. Table 4.3 summarizes some implementation results at two different computing time values. The results illustrated in Table 4.3 show that this architecture consumes a very small amount of ALUTs (343 ALUTs $< 1\%$ of the total amount available in the STRATIX II device) at computing time $t' = 3\ T_c$ where T_c is restricted to 2 ns. The area can be reduced to 203 ALUTs by reducing the number of cells implemented in stage 4. In this case the computing time increases to $t' = 6\ T_c$.

Until now, scenario 1 was dedicated to optimize the computing time of each operator included in the TMVFFT. Nevertheless, having a Velcro design, the TMVFFT operator architecture is supposed to evolute toward a more efficient design based on some reconfigurability aspects what the next section will deal with.

4.6 Scenario 2: toward a combined TMFFT operator

This scenario aims to combine the FFT-GF2 structure with the DMFFT structure on the same die area to obtain a combined and reconfigurable operator TMFFT. This combination can be achieved if the following two steps can be realized.

1. Providing the $GF(2^m)$ operations, mainly the multiplications, with the binary multipliers implemented in the DMFFT operator.

2. Incorporation of the FFT-GF2 physical structure into the DMFFT structure.

We begin the discussion with the first step where the approach for a combined multiplier exploits the fact that the partial product generations of both $GF(2^m)$ multiplier and standard binary multiplier can be performed using the same array and interconnections between cells.

[(2)] All experiments described in this work were performed on a PC (pentium 4, 3 GHz, 1 GB of memory) running Windows XP. The VHDL code was synthesized using Quartus II version 6 and implemented on STRATIX II, EP2S15F484C3 Device with the option "Standard Fit" as the level of the Fitter's effort.

4.6.1 Basic binary multiplication

Let us consider the basic principle of the standard binary arithmetic multiplier. In 1964, Wallace [112] introduced a notion of a carry-save tree constructed from one-bit full adders as a way of reducing the number of partial product bits in a fast and efficient way. Later, Dadda [113] refined Wallace's method by defining a cell placement strategy that minimizes the number of full-adders and half adders, at the cost of a larger carry propagate adder. For our multiplier design, we will consider the Wallace's method since it is structurally more regular.

Wallace'method is based on parallel counters and the multiplication of two binary numbers is performed in the following sequence.

- Form all partial products in parallel with an array of AND gates.
- Reduce the partial products through a series of reduction stages to two numbers by strategically applying (3,2) and (2,2) counters. Architectures of counter (3,2) (or full adder) denoted by "W_3" and (2,2) counter (or half adder) denoted by "W_2" are illustrated in Fig. 4.8.
- Sum the two numbers produced in step 2 using a fast carry-propagate adder to generate the final product.

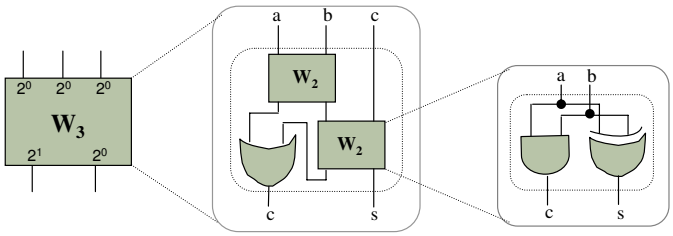

Figure 4.8: The W3 and W2 architectures

4.6.2 $GF(2^m)$ multiplication

As for the $GF(2^m)$ multiplication, it is viewed as a polynomial multiplication modulo $f(x)$, where $f(x)$ is the irreducible polynomial characterizing the field $GF(2^m)$.

Let us consider the multiplication of two $GF(2^m)$ numbers A and B. A and B can be represented by means of the vector basis $\{\alpha^{m-1}, ..., \alpha^1, \alpha^0\}$, where α is a primitive element of the field, as

$$A = a_{m-1}\alpha^{m-1} + ... + a_1\alpha^1 + a_0\alpha^{m-1},$$
$$B = b_{m-1}\alpha^{m-1} + ... + b_1\alpha^1 + b_0\alpha^{m-1},$$

where a_i, b_i are coefficients in $GF(2)$.

The polynomial multiplication $C(x) = A(x)B(x) \bmod f(x)$ can be calculated by firstly building the partial products

$$P_i(x) = A(x)b_i, \qquad for \;\; i = 0, ..., m-1.$$

Secondly, since the polynomial's coefficients belong to $GF(2)$, all partial products $P_i(x)$ have to be added modulo 2. A partial result $C_p(x)$ can be obtained with

$$C_p(x) = \sum Pi(x) \ mod \ 2$$

$$C_p(x) = c_0 + c_1 x + ... + c_{2m-2} x^{2m-2},$$

which is obviously not an element of the field $GF(2^m)$ and should be reduced modulo $f(x)$.

A $GF(2^m)$ multiplier then performs two basic operations. The product of two elements and the modulo $f(x)$ correction. The first operation can be performed by ANDing the corresponding a_i and b_i, for $i = 0, ..., m-1$, and subsequently adding the partial products modulo 2. These partial products must be arranged in rows, with each row shifted i positions to the left as shown in Fig. 4.9. This step is similar to the standard binary product with the only difference that the sum of partial products in this case is done modulo 2. Thus, an opportunity to exploit cells and the interconnection structure of partial product generator of a typical binary multiplier unit is possible. The modulo correction should be performed separately.

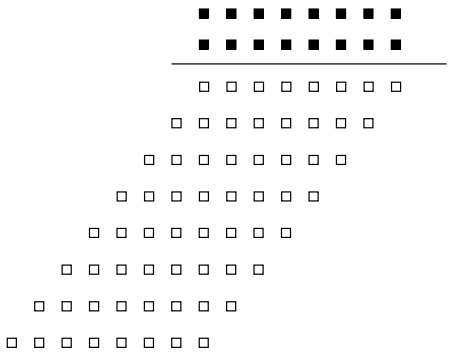

Figure 4.9: Partial product matrix

Early designs of $GF(2^m)$ multipliers used a serial approach. Although serial $GF(2^m)$ multipliers have low hardware requirements, they are very slow. Consequently, several parallel designs have been proposed in the literature [114] [115] [116] [117]. In [118] [119], the idea to combine $GF(2^m)$ with a standard binary multiplier on DSP processor was investigated. In [118], the proposed design is based on a Wallace tree multiplier which has been modified to perform either conventional binary or $GF(2^m)$ multiplication. Their polynomial reduction (or modulo correction) introduces a linear delay. In [119], a new wiring scheme, to avoid adding carries of partial product reduction, is proposed and a parallel polynomial reduction is used. The authors designed a multiplier capable of performing either 16-bit two's complement or unsigned multiplication, or two independent 8-bit $GF(2^8)$ multiplications. In the following, we propose a general approach allowing the implementation of any $GF(2^m)$ multiplier, for $6 \leq m \leq 8$, in the reconfigurable multiplier proposed in chapter 3. We start here from the standard binary structure of the multiplier shown in Fig. 4.10. The partial product generation or bitwise ANDing of $a_i b_i$

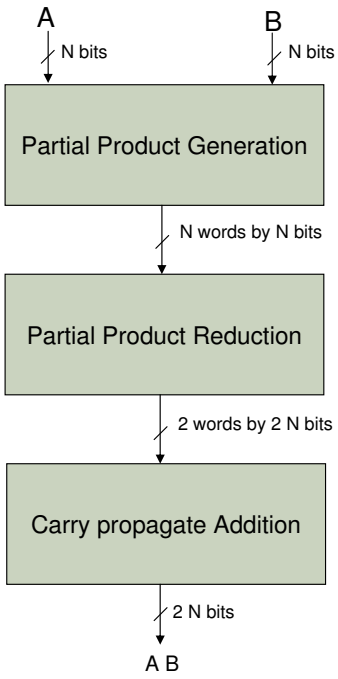

Figure 4.10: Steps for N by N multiplication

is performed regardless of the type of multiplication ($GF(2^m)$ or standard binary multiplication). The partial product reduction should be redesigned in such a way to avoid the carries propagation if the $GF(2^m)$ multiplication is to be performed. This can be realized by reconfiguring the wire connects of the "W3" cells. The new wiring scheme is illustrated in Fig. 4.11. This wiring scheme can be easily realized in a reconfigurable way on FPGA devices using a reprogrammable LUTs. Arriving at the last step (the polynomial reduc-

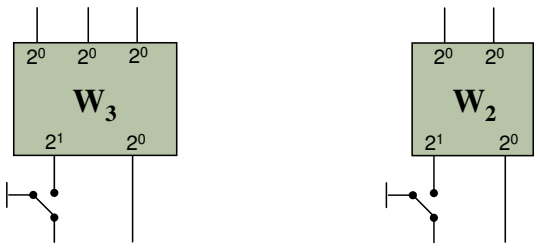

Figure 4.11: The (3,2) and (2,2) counters in $GF(2^m)$ multiplication

tion), it can be performed in parallel by using the method considered in [116]. Let $P(\alpha)$ be the extended result before the polynomial reduction. $P(\alpha)$ can be expressed as

$$P(\alpha) = \sum_{i=m}^{2m-2} p_i \alpha^i + \sum_{i=0}^{m-1} p_i \alpha^i, \tag{4.13}$$

where α^i for $m \leq i \leq 2m-2$ can be substituted by

$$A_i(\alpha) = \sum_{j=0}^{m-1} a_{i,j} \alpha^j.$$

The $A_i(\alpha)$ are the canonical representations in the field's base of the field elements α^i that appear in the first summation in Equation 4.13.

Let us consider an example of multiplication of two elements A and B in $GF(2^4)$. Let $f(x) = x^4 + x + 1$ be the primitive polynomial and

$$A = \alpha^{10} = 0.\alpha^3 + \alpha^2 + \alpha + 1 = (0111),$$

$$B = \alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1 = (1111).$$

The multiplication operation is described in Fig. 4.12. The classical method of multiplication shown in the upper left side of Fig. 4.12 is expressed by hardware circuit which is based on Wallace tree. This tree, originally designed to perform the standard binary multiplication, is modified in such a way to avoid the carries propagation between the cells constituting the tree. In this approach we have considered the disconnection of the carry's outputs and the corresponding neighboring cell inputs that are set to zero. The disconnected wires are represented by black dots. This wiring scheme can be realized by means of reprogrammable LUT1s that connect the carry outputs of each A^iB^i unit to the corresponding entries of the $W3$ cells. A LUT1 maintain the carry propagation (by connecting the input to the output) in the case of standard binary multiplication and set its output to zero in the case of $GF(2^m)$ multiplication.

Fig. 4.12 shows an example of $GF(2^4)$ multiplier whose whole architecture, except the polynomial reduction, is implemented on the standard binary multiplier. This implementation can be extended for any $GF(2^m)$ multiplier provided that the size of the original binary multiplier can support the size of the $GF(2^m)$ multiplier to be implemented.

In the next subsection, we propose a reconfigurable architecture of a combined multiplier that can support either a binary multiplication or $GF(2^m)$ multiplication for m=6, 7 and 8.

4.6.3 Combined multiplier

In this subsection we consider the combination of $GF(2^m)$ multiplier, for $m = 6, 7, 8$, with the standard binary multiplier. Fig. 4.13 shows a block diagram of the combined multiplier 8×8 bits. In this design, we restrict the sizes of GF multipliers according to the code lengths of RS codes used in the practical applications.

The architecture we propose is based on the Wallace's tree for the partial product reduction. A tree of size 8×8 is sufficient to perform $GF(2^8)$ multiplication. However, the tree size is fixed according to the desired precision for the complex Fourier transform. As seen in chapter 3, a 13 bit-wordlength represents a good complexity-precision tradeoff. Thus, for $GF(2^m)$ multiplication, the Wallace's tree can process any two words for $m \leq 13$

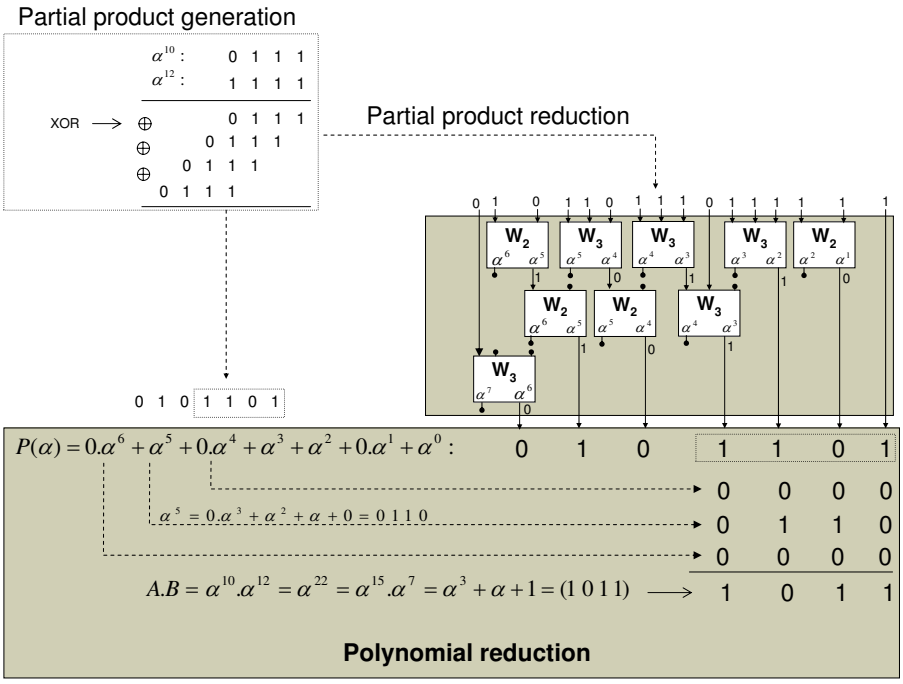

Figure 4.12: Multiplication of two elements in $GF(2^4)$

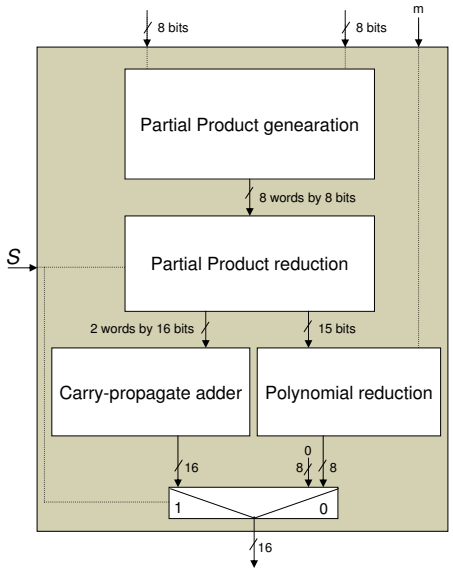

Figure 4.13: Block diagram of the combined multiplier

but a further attention should be taken into account for the realization of the polynomial reduction. For this, in the following we restrict the values of m to the more practical ones. The multiplier receives a control signal S to pilot the switching from the standard binary

addition to modulo 2 addition and to manage the configuration of the interconnection between the neighboring cells according to the model previously shown in Fig. 4.12. The multiplier performs the partial product reduction regardless of the size of the operands. If the size is smaller than 13, the operands can be concatenated by "0". The rest of the architecture is the carry-propagate adder (for the standard binary multiplication) and the polynomial reduction unit (for the $GF(2^m)$ multiplication).

The part which will be influenced by the variable length of the GF multiplier is the polynomial reduction unit since the reduction is based on the binary representation of the powers of α which varies with the order of the Galois field. For this, we propose a reconfigurable polynomial reduction unit as shown in Fig. 4.14.

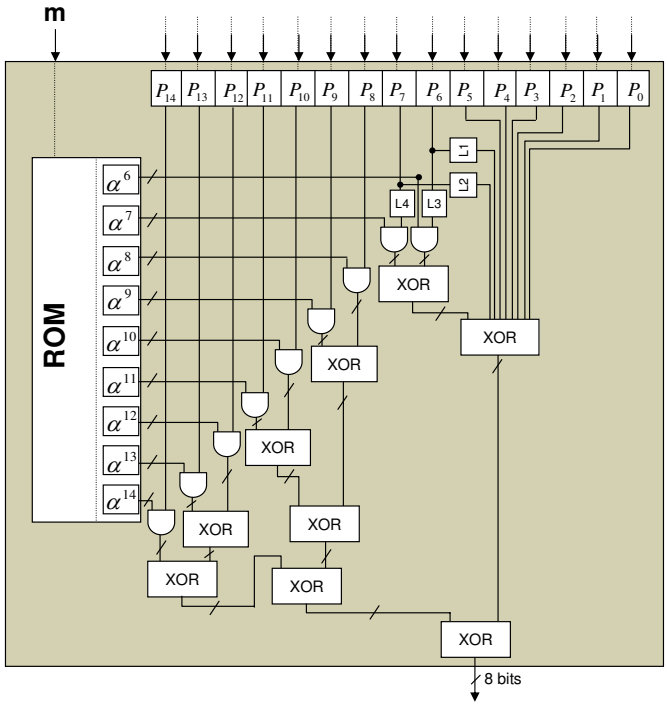

Figure 4.14: Parallel polynomial reduction for $m = 6$, 7 and 8.

This polynomial reduction unit receives 15 bits from the partial product reduction block and a parameter m which defines the size of $GF(2^m)$ multiplication to be performed. According to the chosen GF, the corresponding powers of α are selected from the ROM blocks. By ANDing these powers of α with the corresponding P_i, for $m \leq i \leq 2m-2$, and XORing their results with the word $\{P_{m-1}$ ... P_1 $P_0\}$, the $GF(2^m)$ multiplication is provided on 8 bits. Four LUT1s ($L1$, $L2$, $L3$, $L4$) are needed to select the corresponding entries of the two first XOR gates depending on the $GF(2^m)$. These LUT1s are configured to provide their outputs according to Table 4.4. If $m < 8$, the most significant bits of P_is are pre-initialized to zero and the rest of bits will contain the useful result. The latency of the polynomial reduction unit is equivalent to the time needed to perform one 8-bit AND and four 8-bit XOR.

Table 4.4: Configuration of the LUT1s

multiplier size	L1	L2	L3	L4
$m = 6$	0	0	P_6	P_7
$m = 7$	P_6	0	0	P_7
$m = 8$	P_6	P_7	0	0

The combined multiplier shown in Fig. 4.13 can be easily integrated within the reconfigurable multiplier designed in Chapter 3 leading to triple mode multiplier (Fig. 4.15) able to perform three different multiplications: (i) conventional binary multiplication (ii) multiplication over $GF(F_t)$ and (iii) multiplication over $GF(2^m)$.

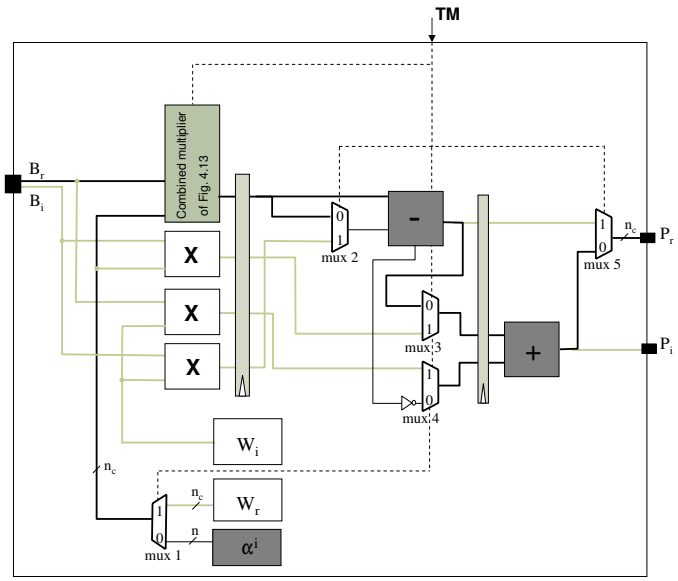

Figure 4.15: Triple mode multiplier

With this multiplier and by taking into account that a modulo 2 addition can be realized by means of XOR gates, the arithmetical resources required to design the TMFFT will be available. Thus, step 1 of scenario 2 is practically accomplished.

To achieve scenario 2, step 2 should be realized. Further studies beyond this thesis is needed for this step which aims to find a way allowing the structure of the $GF(2^m)$-FFT to be integrated within the DMFFT operator.

4.7 Conclusions

In this chapter, we have presented a TMVFFT operator consisting of two self-contained operators: DMFFT and FFT over $GF(2^m)$. This triple mode operator is able to provide three different types of transforms: FFT over $\mathbf{C}$, FNT over $GF(F_t)$ and FFT over $GF(2^m)$. To optimize the reuse and the concept of this operator, we have proposed two scenarios allowing its evolution. Scenario 1 allows the maximization of reuse of each functionality provided by the TMVFFT by reducing the computing time. In this context, we have discussed the evolution of the DMFFT and we have designed a FFT-GF2 based on cyclotomic decomposition of polynomials representing the input data. A very fast computing speed is attained which makes the FFT-GF2 able to produce its functionality at least twice faster than the existing solutions.

Scenario 2 investigated some opportunities to combine the FFT-GF2 with the DMFFT. This can lead to a new operator architecture called TMFFT that performs the same three functionalities in a economical way, i.e. by exploiting the same logical cells at the gate level with a low complexity and a marginal delay overhead. This second scenario consists of two steps: step 1 aiming to merge the arithmetical operators and step 2 aiming to integrate the FFT-GF2 within the DMFFT. Step 1 was developed in this work by designing a combined and reconfigurable multiplier able to perform a standard binary multiplication and $GF(2^m)$ multiplications for $m = 6, 7$ and 8. The designed structure can be easily implemented onto the reconfigurable multiplier realized in Chapter 3 to obtain a triple mode multiplier which will constitute the core of the TMFFT operator. As for the step 2, further studies are needed to develop it and achieve the intended TMFFT.

The TMFFT architecture will serve as the basis for a CO able to be implicated in OFDM modulation and channel coding particulary the RS codes.

Conclusions and prospects

This thesis has addressed the problem of parametrization technique under the CO approach where it is in line with the optimal design of a SWR system intended to support several communication standards. In this context, we investigate the frequency processing of cyclic codes particularly RS codes with the aim to insert the classical FFT operator, initially used for complex Fourier transform, in the encoding and decoding processes of RS codes. By examining the characteristics of the Fourier transform FFT-GF2 used to process the encoding and some decoding processes of classical RS codes defined over $GF(2^m)$, we found that the adequacy of its structure to the structure of FFT defined over $\mathbb{C}$ (FFT-$\mathbb{C}$) is challenged by the transform length. That is, the most efficient algorithms applied to a transform of length 2^m in the case of FFT-$\mathbb{C}$ computations cannot be applied to compute FFT-GF2 since its transform length is of the form $2^m - 1$. For this, we thought to seek out a transform satisfying the FFT-$\mathbb{C}$ criteria while keeping in mind to find a way to include the FFT-GF2, used in the RS coding defined over $GF(2^m)$, in the intended common and reconfigurable FFT structure.

To be able to exploit the whole FFT-$\mathbb{C}$ structure we explored finite field transforms having a highly composite length. The candidate transforms were the Fermat transforms defined over $GF(F_t)$. These transforms led us to revive a specific class of RS codes defined over $GF(F_t)$. These codes were studied in 1976 where the authors have emphasized the importance of their codeword length allowing the use of efficient algorithms to compute their associated Fermat transforms. These codes were also recommended for the use in spacecraft communications where the RS(256,224) over $GF(257)$ was studied to be used together with a convolutional code.

From arithmetical structure point of view, we have noticed that the arithmetical operations defined over $GF(2^m)$ are simpler than those defined over $GF(F_t)$. However, our choice to study these specific RS codes was motivated by the fact that the FNT transform used in the processing of encoding and decoding processes of these codes will be based on the pre-implemented FFT-$\mathbb{C}$. We have studied and simulated these codes and we have shown that their performances are almost similar to those of RS codes defined over $GF(2^m)$. According to the simulations results, we have shown that the frequency encoding of these codes is not the suitable solution since the representation of the symbol "$F_t - 1$" affects all the spectral components of the codeword. Then, we proposed to use time domain encoding and represent the information symbols by 2^t bits and the check parity symbols by $2^t + 1$ bits.

As a first contribution, we have investigated the redesign of the FFT-$\mathbb{C}$ in such a way to be able to provide two functionalities: complex Fourier transform and Fermat transform. Conceptually, the design of such a dual mode operator implies the design of arithmetical operators capable to operate over the two domains: C and $GF(F_t)$. We have proposed

reconfigurable architectures for the multiplier, adder and subtracter. These operators implemented on the complex butterfly (available in the FFT-$\mathbb{C}$) led to a reconfigurable butterfly that will constitute the core of the dual mode operator. Based on the FFT-$\mathbb{C}$ structural strategy, we have designed the architecture of the DMFFT operator.

To evaluate the complexity and speed performances of this operator, we have considered its implementation on ALTERA's FPGA devices. Compared to a Velcro FFT/FNT operator, the DMFFT presented an important gain in terms of ALUTs and memory saving. We have shown that for a transform length $N = 64$ implemented with different wordlengths ($9 \leq n_c \leq 16$), the DMFFT presents a memory saving between 20 and 30%, a gain in ALUTs and performance-to-cost ratio gain that go from 9.2 % up to 26 % and from 9.7 % up to 37.4 % respectively. This deviation of gain in ALUTs and in performance-to-cost ratio is directly related to the wordlength, so as n_c increases, these gains decrease. For $N = 256$, the DMFFT presents the same memory saving and the other gains evolve in the same manner of that of DMFFT-64 but with lower values. This is due to the fact that the complexity of the DMFFT architecture is mainly dominated by the FFT-$\mathbb{C}$ architecture complexity which increases with the increase of the wordlength and the transform length. The high n_c values are necessary to get a good FFT-$\mathbb{C}$ computations precision while in the Velcro FFT/FNT, a 9-bit wordlength is sufficient to implement the FNT. Then, for a transform length $N \leq 256$, the use of the DMFFT is largely efficient and allows to process RS codes which have codeword length $N \leq 256$ and whose principals are N=64, 128 and 256.

In order to treat the classical RS codes defined over $GF(2^m)$ used in the actual standards, we have proposed the design of a triple mode FFT operator able to provide three functionalities: FFT-$\mathbb{C}$, FNT and FFT-GF2. We started from a Velcro architecture consisting of two self-contained operators DMFFT and FFT-GF2 and we have proposed two scenarios to upgrade this Velcro architecture. The first scenario aimed to maximize the reusability of the TMVFFT by speeding up the computing time of each operator included in the TMVFFT. Based on cyclotomic decomposition, we proposed an efficient architecture of the FFT-GF2 allowing a reduced computing time and a very high throughput rate.

The second scenario aimed to combine the FFT-GF2 with the DMFFT by exploiting the same logical cells at the gate level to obtain a reconfigurable TMFFT operator. From this perspective, we have proposed a generic combined multiplier able to perform a standard binary multiplication and $GF(2^m)$ multiplications for m=6, 7 and 8. We have also showed that this multiplier can be easily implemented into the reconfigurable multiplier proposed in chapter 3 which leads to a triple mode multiplier able to operate over three different fields: $\mathbb{C}$, $GF(F_t)$ and $GF(2^m)$. Having this triple mode multiplier and taking into account that the addition in $GF(2^m)$ can be realized by XOR gates, the necessary arithmetical tools to implement the FFT-GF2 are available in the DMFFT operator. What remains to achieve scenario 2 is the integration of the FFT-GF2 structure on the DMFFT. Thus, further studies are needed to reach the TMFFT.

Prospects

This thesis provided a framework that naturally extends to investigate future research topics of interest among which we mention the following:

- Design of a wiring methodology of the FFT-GF2 cells allowing the incorporation of the FFT-GF2 within the DMFFT operator.
- Development of a dynamic reconfiguration that allows the elimination of some multiplexors and some control blocks leading to more efficient reconfigurable DMFFT and TMFFT operators. A related study is to define a task scheduling technique to manage the sharing of the reconfigurable operator considered as common between several processings.
- Study of the Gao algorithm for decoding the RS codes. This algorithm consists of the following main operations: interpolation, partial gcd, and long division of polynomials. As the author mentioned, all these operations can be implemented by the fast algorithms based on FFTs.
- The desirable continuation of this work would be the extension of the use of the reconfigurable FFT towards the convolutional codes and non-binary LDPC codes.

Publications

International Conferences

Ali Al Ghouwayel, Yves Louët, Amor Nafkha and Jacques Palicot, *On the FPGA Implementation of the Fourier Transform over Finite Fields $GF(2^m)$*, IEEE ISCIT'2007, Sydney, Australia, Oct 16-19, 2007.

Ali Al Ghouwayel, Yves Louët and Jacques Palicot, *Complexity Evaluation of a Re-Configurable Butterfly with FPGA for Software Radio Systems*, IEEE PIMRC'07, Athens, Greece, September 2007.

Ali Al Ghouwayel, Yves Louët and Jacques Palicot, *A Reconfigurable Architecture for the FFT Operator in a Software Radio Context*, IEEE ISCAS'2006, Island of Kos, Greece, May 2006.

Ali Al Ghouwayel, Yves Louët and Jacques Palicot, *A Reconfigurable Butterfly Architecture for Fourier and Fermat Transforms*, IEEE WSR'2006, Karlsrhue, Germany, March 2006.

National Conferences

Ali Al Ghouwayel, Yves Louët, Jacques Palicot, *Un opérateur reconfigurable dans un contexte Radio Logicielle : de la transformée de Fourier à la transformée de Fermat*, MajecStic'2006, Lorient (France), Novembre 2006.

Ali Al Ghouwayel, *La Radio Logicielle et la Paramétrisation*, Journée des doctorants en électronique de Bretagne, LESTER, Lorient (France), 4 Décembre 2004.

Oral Communications

Ali Al Ghouwayel, *FFT: Un opérateur reconfigurable dans un contexte Radio Logicielle*, Doctoriales de Bretagne 2006, Landerneau (France), Novembre 2006.

Ali Al Ghouwayel, *Traitement fréquentiel des codes cycliques et reconfiguration de l'opérateur FFT*, Séminaire TAMCIC-ENST de Bretagne, Brest (France), 8 juin 2006.

Ali Al Ghouwayel, *Traitement fréquentiel des codes cycliques et reconfiguration de l'opérateur FFT*, Séminaire SCEE, Supélec campus de Rennes, 16 Mars 2006.

Appendix

Appendix A

CRC and Reed-Solomon codes with MulDiv

This appendix presents how the MulDiv operator can be parameterized to be used in CRC and RS codes over $GF(2^m)$.

A.1 The CRC calculation with MulDiv

Cyclic redundancy check coding is an error-control coding technique for detecting errors that occur when a message is transmitted. Unlike block or convolutional codes, CRC codes do not have a built-in error correction capability. Instead, when an error is detected in a received message word, the receiver requests the sender to retransmit the message word. The principle of the CRC calculation is as follows: consider a binary message M whose polynomial representation is $M(x)$ and $G(x)$ a generator polynomial of degree d. The transmitted message $T(x)$ has a polynomial representation:

$$T(x) = x^d M(x) + R(\frac{x^d M(x)}{G(x)}), \tag{A.1}$$

where $R(.)$ represents the remainder of the Euclidian polynomial division.

This remainder can be calculated using a circuit similar to the one presented in Fig. 1.10. The parameters h_i and g_i have just to be parameterized in accordance with the CRC calculation.

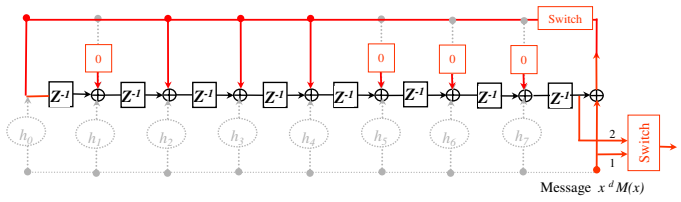

Figure A.1: CRC 8 calculation for the GSM standard with Muldiv operator

Fig. A.1 is an example of the CRC 8 calculation in the case of the GSM standard. All h_i coefficients are equal to zero as well as some of g_i. The generator polynomial has

expression $1 + D^2 + D^3 + D^4 + D^8$. The CRC calculation of the circuit of Fig. A.1 can be carried out as follows: the message is shifted into the registers from the right end while the upper switch is turned on and at the same time the lower switch is connected to the input "1" and shifts $M(x)$ to the output of the circuit. After the entire message $M(x)$ has been shifted into the registers, the contents of these registers form the remainder of the division of $x^d M(x)$ by $G(x)$. Then, the upper switch is turned off and the lower one is turned to the input "2" to get the remainder result located in all the shift registers. Starting from this example, the same structure is applied for the CRC 3 (see Fig. A.2).

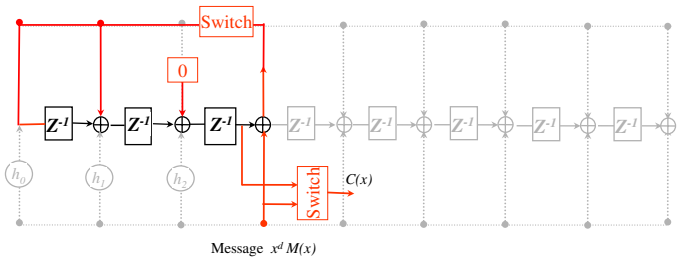

Figure A.2: CRC 3 calculation for the GSM standard with Muldiv operator

The generator polynomial is expressed as $1 + D + D^3$. As in Fig. A.1, the switches of Fig. A.2 have the same functions.

Tab. A.1 summarizes all the CRC polynomials for three different standards: IEEE 802.11.g, GSM and UMTS.

The next section will show how the MulDiv architecture can be derived for some channel coding/decoding operations. The identified codes are the well known error correcting cyclic codes and the Reed-Solomon codes.

Table A.1: CRC polynomials for GSM, UMTS and IEEE 802.11.a standards

Standards	CRC polynomial
IEEE 802.11.g	$G_{CRC-16}(D) = D^{16} + D^{12} + D^5 + 1$
GSM	$G_{CRC-3}(D) = D^3 + D + 1$ $G_{CRC-8}(D) = D^8 + D^4 + D^3 + D^2 + 1$ $G_{CRC-6}(D) = D^6 + D^5 + D^3 + D^2 + D + 1$ $G_{CRC-40}(D) = (D^{23} + 1) + (D^{17} + D^3 + 1)$
UMTS	$G_{CRC-24}(D) = D^{24} + D^{23} + D^6 + D^5 + D + 1$ $G_{CRC-16}(D) = D^{16} + D^{12} + D^5 + 1$ $G_{CRC-12}(D) = D^{12} + D^{11} + D^3 + D^2 + D + 1$ $G_{CRC-8}(D) = D^8 + D^7 + D^4 + D^3 + D + 1$

A.2 Error-correcting cyclic codes

The encoding circuit of error-correcting cyclic codes is similar to that of the CRC codes described in the previous section. The decoding of error-correcting cyclic codes consists of three steps: syndrome computation, association of the syndrome to an error pattern accomplished by an error-pattern detector, and error correction. The syndrome computation can be accomplished with MulDiv operator whose complexity is linearly proportional to the number of parity-check digits (i.e., $n-k$ where n, k are the code and message block-length respectively). The error-correction step constitutes simply adding (modulo 2) the error pattern to the received vector. This can be achieved with a single XOR gate if corrections are carried out in a serial manner (i.e. one digit at a time). The association of the syndrome to an error pattern can be completely specified by a decoding table. Fig. A.3 shows the decoding circuit for a (7,4) cyclic code. In this circuit, the syndrome is formed by shifting the entire received vector into the syndrome register. At the same time, the received vector is stored in the buffer register. The syndrome is read into the detector and is tested for the corresponding error pattern. The output of the detector is 1 if, and only if, the syndrome in the syndrome register corresponds to a correctable error pattern with an error at the highest-order position $x^n - 1$. In the circuit of Fig. A.3, the error pattern detector is made with a single three-input AND gate. In this decoder, the MulDiv operator forms a main circuit element, which accomplishes the syndrome computation. Having this reconfigurable operator, we can think of a decoder, which can decode cyclic codes generated by any generator polynomial $g(x)$. To do this, one also needs to reconfigure the error pattern detector. The reconfigurability of the error pattern decoder and of the MulDiv operator allows the creation of a decoder that can decode various block-length cyclic codes.

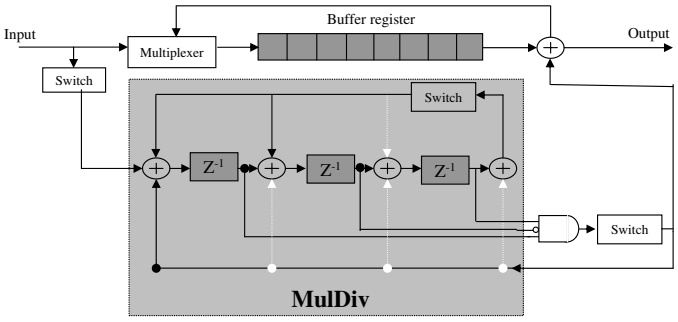

Figure A.3: Decoding principle for the (7,4) cyclic code generated by $1 + x + x^3$ using MulDiv operator

A.3 Reed Solomon codes with MulDiv operator

In this section, we show that the *Galois Field* (GF) basic operations as well as encoding and some decoding processes of RS codes can be performed with the aid of MulDiv.

A.3.1 Basic operations in Galois Field (GF)

It is interesting to show that the circuits that generate the GF symbols and the circuits that perform the multiplication of two GF elements are based on the MulDiv architecture. The aim is now to present MulDiv as an operator that can be parameterized to generate and multiply the GF elements.

Let us begin with the generation of GF elements. Fig. A.4 represents the circuit generating all the non zero elements of $GF(2^4)$. To perform this generation, the vector (1 0 0 0) which is the binary representation of $\alpha^0 = 1$ (bit at the top left is the least significant bit), where α is the primitive element of GF, is loaded into the register. Then, successive shifts of the register will generate the vector representations of the successive powers of α. This circuit can be also used to multiply an arbitrary element of $GF(2^4)$ by α.

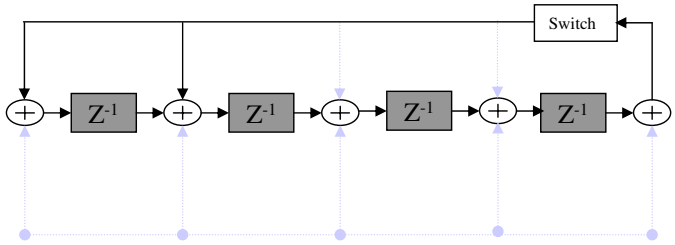

Figure A.4: Architecture for generating elements of $GF(2^4)$ with MulDiv operator

Now, let us consider the multiplication of two arbitrary elements A and B of $GF(2^4)$. By expressing these two elements in their polynomial form, $A = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3$ and $B = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$, then the product $A.B$ can be expressed by:

$$B.A = (((a_3B)\alpha + a_2B)\alpha + a_1B)\alpha + a_0B.$$

This calculation can be realized with the MulDiv operator, as shown in Fig. A.5.

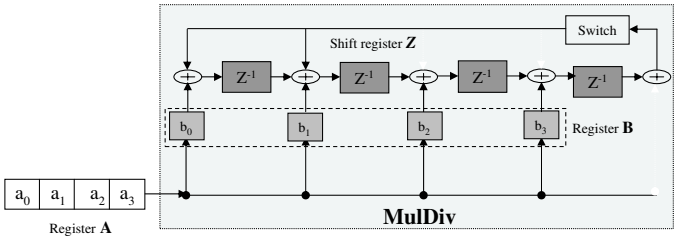

Figure A.5: Architecture for multiplying two elements of $GF(2^4)$ with MulDiv operator

With this architecture, the shift registers Z^{-1} are initially empty and (a_0, a_1, a_2, a_3) and (b_0,b_1,b_2,b_3), the vector representations of A and B, are respectively loaded into registers A and B. Then registers A and Z are shifted four times. At the end of the fourth shift, register Z contains the product $A.B$ in vector form.

A.3.2 RS coding and decoding algorithms

The RS encoding can be carried out using MulDiv operator with the same parametrization aspect previously discussed. Since the RS encoding is realized by means of polynomial divisions, it can be easily performed with the aid of MulDiv and in this case, the hardware resources: adder, multiplier and storage devices deal with elements from $GF(2^m)$ rather than from $GF(2)$.

The RS decoding process consists of the successive following steps: syndrome computation, erasure location polynomial, Berlekamp algorithm (to compute the error-locator polynomial $\Lambda(x)$), some polynomial computations, Chien search, polynomial evaluation and error-correction algorithm. Among these operations, there are four operations that can be realized using the MulDiv operator.

Let us consider the received word $R(x) = C(x) + S(x)$, where $S(x)$ denotes the syndrome polynomial and $C(x)$ the code word that can be expressed as the product of the polynomial generator: $C(x) = g(x)q(x)$. Then, the syndrome $S(x)$ defined as the remainder of division of $R(x)$ by $g(x)$ can be performed with MulDiv.

The polynomial computations imply the derivation of two polynomials: $\Omega(x)$ (the error-evaluator polynomial) and $\Lambda'(x)$ (the derivative of $\Lambda(x)$). The polynomial computation of $\Omega(x)$ can be expressed as follows:

$$\Omega(x) = S(x)\Lambda(x)(modx^{2t}),$$

where for $t = 2$:

$$S(x) = S_1 + S_2x + ... + S_4x^3,$$

and

$$\Lambda(x) = \Lambda_0 + \Lambda_1x + ... + \Lambda_3x^3.$$

Then the Ω_i can be expressed as the following equations:

$$\Omega_0 = \Lambda_0S_1,$$

$$\Omega_1 = \Lambda_0S_2 + \Lambda_1S_1,$$

$$\Omega_2 = \Lambda_0S_3 + \Lambda_1S_2 + \Lambda_2S_1,$$

and finally

$$\Omega_3 = \Lambda_0S_4 + \Lambda_1S_3 + \Lambda_2S_2 + \Lambda_3S_1.$$

These computations can be easily performed using MulDiv as shown in Fig. A.6, where the adder, multiplier and registers denote the devices that deals with elements from $GF(2^4)$.

The computation of $\Lambda'(x)$ can be implemented with a simple wired circuit where: $\Lambda'_{2i} = \Lambda_{2i+1}$, for $i = 0, ..., t-1$. The "Chien search" consists in finding the roots of the error locator polynomial $\Lambda(x)$ that can be realized with the circuit of Fig. A.7. These roots identify the error locations. Once an error location is found, the next step is to find the error values, what consists in evaluating the polynomials $\Omega(x)$ and $\Lambda'(x)$ for the finite fields elements $1, \alpha, \alpha^2, ..., \alpha^{n-1}$, where n is the the code block-length. This step is called the polynomial evaluation. This polynomial evaluation and the Chien search are performed by multiplying the polynomial coefficients by successive powers of α and summing the partial products. Each multiplication can be realized by MulDiv operator

where the polynomial coefficient can be loaded in register A, and register B contains the binary representations of α^i for $i = 1, ..., t$. Each elementary multiplication module of Fig. A.7 circuit is similar to the one presented in Fig. A.5. In that sense, the MulDiv operator is a basic element of the polynomial evaluation of RS decoding algorithms.

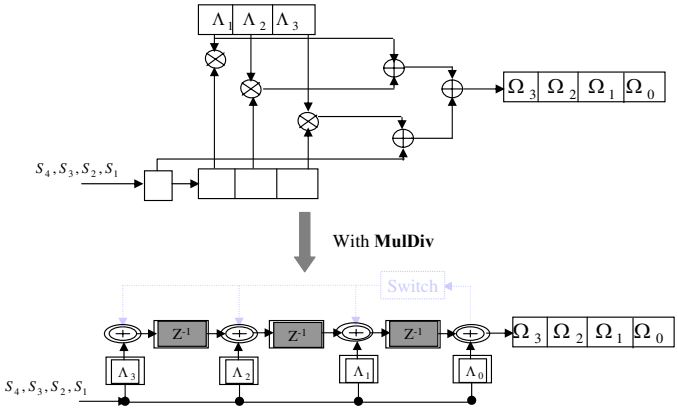

Figure A.6: Polynomial computation $\Omega(x)$

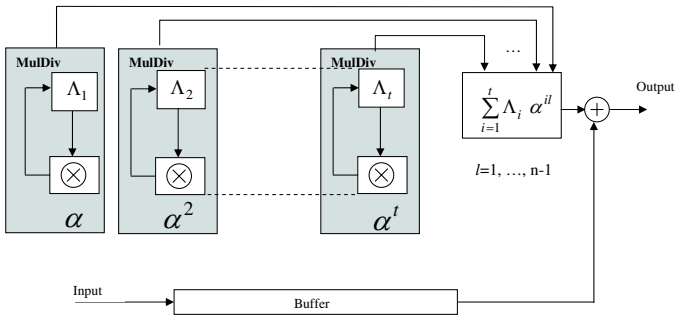

Figure A.7: Chien search computation

To conclude, encoding and some steps of the decoding process of RS codes can be realized with the MulDiv operator as shown in Fig. A.8. This operator can be parameterized with the appropriate weights h_i and g_i (explained in Fig. 1.9) so as to perform the corresponding operation.

A.4 The MulDiv hardware implementation: the MDI and MDIV operators

By considering the example of CRC calculation, we will show how the MulDiv operator can significantly reduce the complexity in terms of computation complexity. Let us consider the MulDiv hardwired pattern shown in Fig. A.9. This hardwired pattern, seen as a common operator, executes one task in one clock cycle. This structure is parameterized

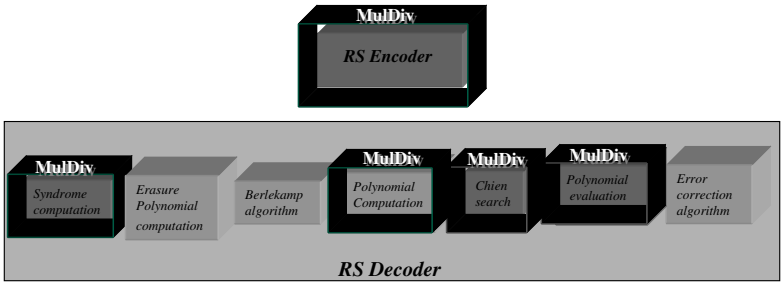

Figure A.8: RS encoder-decoder with MulDiv operator

(by the g_i and h_i coefficients) and will be called *MDI*. The MDI operator is the elementary structure of MulDiv.

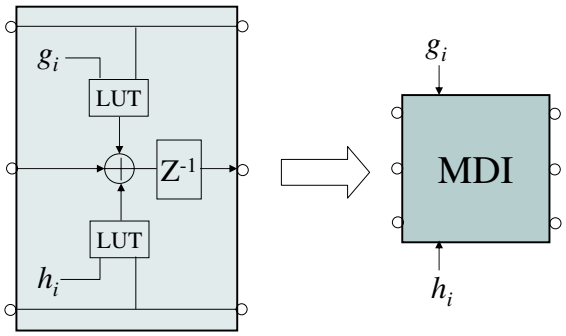

Figure A.9: The MDI operator

In this case, the coefficients g_i are associated to the CRC polynomials and h_i are settled to zero). By concatenating several MDI patterns, we get the general structure of the CRC previously shown in Fig. A.1 where the number of MDI cells is related to the degree of the CRC polynomial (eight in this example). Knowing that the degree of most CRC polynomials is multiple of four, a novel operator called "MDIV" can be defined (see Fig. A.10) and considered as a common operator able to executes one task in one clock cycle.

Table A.2 shows the benefit of using the MDIV operators (which includes MDI operator) instead of classical elements such as shift registers. By considering the standards IEEE 802.11.g, GSM and UMTS, we have reported the figures concerning the operator numbers per standard, for each CRC polynomial.

It can be seen that the use of a hardwired common operator, that executes one task in one clock cycle, leads to a significant reduction in the computation complexity.

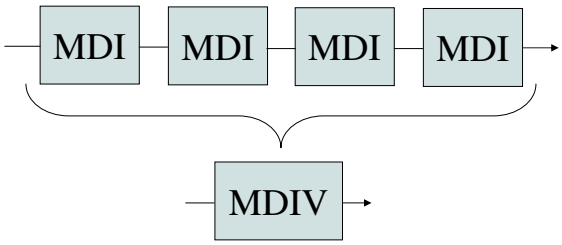

Figure A.10: The MDIV operator composed of 4 MDI operators

Table A.2: CRC calculation with MulDiv

Standard	CRC polynomial	Number of operators without MulDiv	Number of operators with MDIV and MDI
IEEE 802.11.g	$G_{CRC-16}(D) = D^{16} + D^{12} + D^5 + 1$	16 SR, 3 XOR	4 MDIV, 1 XOR
GSM	$G_{CRC-3}(D) = D^3 + D + 1$	3 SR, 2 XOR	3 MDI, 1 XOR
	$G_{CRC-8}(D) = D^8 + D^4 + D^3 + D^2 + 1$	8 SR, 4 XOR	2 MDIV, 1 XOR
	$G_{CRC-6}(D) = D^6 + D^5 + D^3 + D^2 + D + 1$	6 SR, 5 XOR	1 MDIV, 2 MDI, 1 XOR
	$G_{CRC-40}(D) = (D^{23} + 1) + (D^{17} + D^3 + 1)$	40 SR, 5 XOR	10 MDIV, 1 XOR
UMTS	$G_{CRC-24}(D) = D^{24} + D^{23} + D^6 + D^5 + D + 1$	24 SR, 5 XOR	6 MDIV, 1 XOR
	$G_{CRC-16}(D) = D^{16} + D^{12} + D^5 + 1$	16 SR, 3 XOR	4 MDIV, 1 XOR
	$G_{CRC-12}(D) = D^{12} + D^{11} + D^3 + D^2 + D + 1$	12 SR, 3 XOR	3 MDIV, 1 XOR
	$G_{CRC-8}(D) = D^8 + D^7 + D^4 + D^3 + D + 1$	8 SR, 5 XOR	4 MDIV, 1 XOR
Total number		168	48

Appendix B

Some elements for Galois fields construction

In this Appendix we give some principal algebraic tools helping to the construction of Galois fields.

B.1 Some prime polynomials for different $GF(2^m)$

Table B.1: Prime polynomials over $GF(2)$

m	Prime polynomial
2	$x^2 + x + 1$
3	$x^3 + x + 1$
4	$x^4 + x + 1$
5	$x^5 + x^2 + 1$
6	$x^6 + x + 1$
7	$x^7 + x^3 + 1$
8	$x^8 + x^4 + x^3 + x^2 + 1$
9	$x^9 + x^4 + 1$
10	$x^{10} + x^3 + 1$
11	$x^{11} + x^2 + 1$
12	$x^{12} + x^6 + x^4 + x + 1$
13	$x^{13} + x^4 + x^3 + x + 1$
14	$x^{14} + x^{10} + x^6 + x + 1$
15	$x^{15} + x + 1$

B.2 Construction of GF(16)

The field $GF(16)$ can be constructed with the polynomial $x^4 + x + 1$, and the element $\alpha = x$ of order 15 is a primitive element ($\alpha^0 = \alpha^{15} = 1$). The following table gives the different $GF(16)$ elements with the corresponding polynomial and binary representations.

Table B.2: The elements of $GF(16)$

Element	polynomial notation	Binary notation
0	0	0 0 0 0
α^0	1	0 0 0 1
α^1	x	0 0 1 0
α^2	x^2	0 1 0 0
α^3	x^3	1 0 0 0
α^4	$x+1$	0 0 1 1
α^5	x^2+x	0 1 1 0
α^6	x^3+x^2	1 1 0 0
α^7	x^3+x+1	1 0 1 1
α^8	x^2+1	0 1 0 1
α^9	x^3+x	1 0 1 0
α^{10}	x^2+x+1	0 1 1 1
α^{11}	x^3+x^2+x	1 1 1 0
α^{12}	x^3+x^2+x+1	1 1 1 1
α^{13}	x^3+x^2+1	1 1 0 1
α^{14}	x^3+1	1 0 0 1

Appendix C

FFT over $GF(2^m)$

In this Appendix we gives some basic definitions and we develop the computation of FFT-15 over $GF(2^4)$.

C.1 Basic notions and definitions

In this section, we will review some definitions and basic notions required for later developments of cyclotomic FFT algorithm. We also highlight some specific characteristics of the Galois Field of characteristic 2, $GF(2^m)$.

Let G be a group together with the operation denoted by $\bullet$ and let H be a subset of G. Then H is called a *subgroup* of G if H itself a group with respect to the restriction of $\bullet$ to H.

Definition 1. Let h, h_1, h_2, ..., h_m be a sequence of elements of H and choose h_1 to be the identity element. A matrix decomposition of G is called a *coset decomposition* when the matrix is constructed as follows:

1. The first row consists of the elements of H, with identity at the left and every other element of H appearing once.
2. Choose any element of G not appearing in the first row and multiply this element by each element of the sequence h_i , for $i = 1, ..., m$. The resulting vector constitutes the second row of the array.
3. Construct a third, fourth, ..., row similarly, each time choosing a previously unused group element for the element in the first column.
4. Stop when all the group elements appears somewhere in the array.

The array is

$$C = \begin{pmatrix} h_1 = 1 & h_2 & h_3 & h_4 & \cdots & h_m \\ g_2 \bullet h_1 = g_2 & g_2 \bullet h_2 & g_2 \bullet h_3 & g_2 \bullet h_4 & \cdots & g_2 \bullet h_m \\ g_3 \bullet h_1 = g_3 & g_3 \bullet h_2 & g_3 \bullet h_3 & g_3 \bullet h_4 & \cdots & g_3 \bullet h_m \\ g_4 \bullet h_1 = g_4 & g_4 \bullet h_2 & g_4 \bullet h_3 & g_4 \bullet h_4 & \cdots & g_4 \bullet h_m \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ g_n \bullet h_1 = g_n & g_n \bullet h_2 & g_n \bullet h_3 & g_n \bullet h_4 & \cdots & g_n \bullet h_m \end{pmatrix}$$

The first element on the left of each row is known as a coset leader. Each row in the array is called *left coset*, or simply as a *coset* when the group is abelian.

Definition 2. Let k_i be any integer, with $0 \leq k_i \leq p^m - 1$, and let m_i the smallest integer such that $k_i p^{m_i} \equiv k_i \bmod (p^m - 1)$. The *cyclotomic cosets* mod $p^m - 1$ are defined by

$$C_{k_i} = \{k_i,\ k_i p,\ k_i p^2, \ldots, k_i 2^{m_i - 1}\}, \qquad i = 1, \ldots, l, \tag{C.1}$$

where k_i is the coset leader, $i \in [1, l]$, l being the number of cyclotomic cosets modulo $p^m - 1$.

Examples:

1- The cyclotomic cosets mod 7 ($p = 2$, $m = 3$) are:

$$\begin{aligned} C_0 &= \{0\}, \\ C_1 &= \{1, 2, 4\}, \\ C_3 &= \{3, 6, 5\}. \end{aligned}$$

2- The cyclotomic cosets mod 15 ($p = 2$, $m = 4$) are:

$$\begin{aligned} C_0 &= \{0\}, \\ C_1 &= \{1, 2, 4, 8\}, \\ C_3 &= \{3, 6, 9, 12\}, \\ C_5 &= \{5, 10\}, \\ C_7 &= \{7, 11, 13, 14\}. \end{aligned}$$

Definition 3. Let $GF(2^m)$ be a Galois field of order $n = 2^m - 1$. Each element of $GF(2^m)$ can be represented as a linear combination: $a_0 + a_1\alpha + ... + a_{m-2}\alpha^{m-2} + a_{m-1}\alpha^{m-1}$ and the set $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$ is called the basis of the field.

Definition 4. Let $GF(q)$ be a field and $GF(Q)$ be an extension field of $GF(q)$. Let β be in $GF(Q)$. The prime polynomial $f(x)$ of smallest degree over $GF(q)$ with $f(\beta) = 0$ is called the minimal polynomial of β over $GF(q)$.

Definition 5. A *primitive polynomial p(x)* over $GF(q)$ is a prime polynomial over $GF(q)$ with the property that in the extension field constructed modulo $p(x)$, the field element represented by x is a primitive element.

Definition 6. A matrix whose each row is obtained from the preceding row by one left (right) cyclic shift is called *circulant matrix*. Circulant matrices can be square or rectangular matrices. If the entries of a circulant matrices are submatrices, the resultant matrix is referred to as a block circulant matrix. A circulant $n \times n$-matrix can be represented as

$$A = \begin{pmatrix} A_0 & A_1 & \cdots & A_{n-1} \\ A_1 & A_2 & \cdots & A_0 \\ \vdots & \vdots & \vdots & \vdots \\ A_{n-1} & A_0 & \cdots & A_{n-1} \end{pmatrix}.$$

C.2 Development of equation 4.11

Let us consider step by step the multiplication $F = ALf$. Starting from the right side, the multiplication Lf represents a set of cyclic convolutions of each block L_i by the the corresponding cyclotomic coset of the vector f. The matrix L is composed of $(l+1)$ blocks ($(l+1)$ represents the number of cyclotomic cosets of f) and can be represented as

$$L = \begin{pmatrix} L_0 & & & 0 \\ & L_1 & & \\ & & \ddots & \\ & & & L_{l-1} \\ & 0 & & & L_l \end{pmatrix}.$$

By using the normal basis of the $GF(2^m)$, the L_is can be expressed as

$$L_i = \begin{pmatrix} \gamma_i^{2^0} & \gamma_i^{2^1} & \cdots & \gamma_i^{2^{m_i-1}} \\ \gamma_i^{2^1} & \gamma_i^{2^2} & \cdots & \gamma_i^{2^0} \\ \vdots & \vdots & \vdots & \vdots \\ \gamma_i^{2^{m_i-1}} & \gamma_i^{2^0} & \cdots & \gamma_i^{2^{m_i-2}} \end{pmatrix}.$$

Then, the multiplication of L_i by the corresponding sub-vector or cyclotomic coset of f can be represented as

$$\begin{pmatrix} S_{i,0} \\ S_{i,1} \\ \vdots \\ S_{i,m_i-1} \end{pmatrix} = \begin{bmatrix} \gamma_i^{2^0} & \gamma_i^{2^1} & \cdots & \gamma_i^{2^{m_i-1}} \\ \gamma_i^{2^1} & \gamma_i^{2^2} & \cdots & \gamma_i^{2^0} \\ \vdots & \vdots & \vdots & \vdots \\ \gamma_i^{2^{m_i-1}} & \gamma_i^{2^0} & \cdots & \gamma_i^{2^{m_i-2}} \end{bmatrix} \begin{pmatrix} f_{i,0} \\ f_{i,1} \\ \vdots \\ f_{i,m_i-1} \end{pmatrix}, \tag{C.2}$$

where $(f_{i,0}, f_{i,1}, \cdots, f_{i,m_i-1})^T$ represent the ith cyclotomic coset.

As shown in [105], the computation of Equation C.3 can be performed by a cyclic convolution of each row of L_i by the subvector $f_i = (f_{i,0}, f_{i,1}, \cdots, f_{i,m_i-1})^T$, which can be represented by polynomial form as

$$\begin{aligned} S_i(x) &= S_{i,0} + S_{i,m_i-1}x + \ldots + S_{i,1}x^{m_i-1} \\ &= (\gamma_i + \gamma_i^{2^{m_i-1}}x + \ldots + \gamma_i^2 x^{m_i-1})(f_{i,0} + f_{i,1}x + \ldots + f_{i,m_i-1}x^{m_i-1}) \ mod \ (x^{m_i-1}). \end{aligned}$$

This latest Equation can be computed by applying an algorithm proposed by Berlekamp [57] for the cyclic convolution computation. By his algorithm, Berlekamp shows that every cyclic convolution can be decomposed as the product of some predetermined matrices:

$$\begin{pmatrix} S_{i,0} \\ S_{i,1} \\ \vdots \\ S_{i,m_i-1} \end{pmatrix} = Q_i \left(R_i \begin{pmatrix} \gamma_i \\ \gamma_i^{2^{m_i-1}} \\ \vdots \\ \gamma_i^2 \end{pmatrix} . P_i \begin{pmatrix} f_{i,0} \\ f_{i,1} \\ \vdots \\ f_{i,m_i-1} \end{pmatrix} \right), \tag{C.3}$$

where Q_i, R_i and P_i are binary matrices predetermined according to the length of the cyclic convolution. The symbol "." denotes componentwise multiplication of vectors.

Thus, if B_i denotes $R_i(\gamma_i\ \gamma_i^{2^{m_i-1}} \cdots \gamma_i^2)^T$, Equation 4.10 can be rewritten as

$$\mathbf{F{=}AQ(B.(PF))}. \tag{C.4}$$

C.3 Computation of cyclotomic FFT-15 over $GF(2^4)$

Let $f = \{f_i\}$, for $i = 0, ..., 2^4 - 2$, be the input sequence to be transformed. These f_i can be rearranged according to the cyclotomic decomposition modulo $2^4 - 1$ of the set $\{0, ..., 2^4 - 2\}$.

The integers $\{0, 1, ..., 14\}$ will be decomposed into cyclotomic cosets C_{k_s}, $0 \leq s < 15$, where each C_{k_s} consists of the set $\{k_s, k_s 2, k_s 2^2, ..., k_s 2^{m_l-1}\}$, k_s is the coset leader such that $0 \leq k_s \leq 14$ and m_l is the smallest integer such that $k_s 2^{m_l} \equiv k_s \bmod 15$. $l + 1$ represents the number of cyclotomic cosets.

Table C.1 shows the cyclotomic decomposition of f.

Table C.1: FFT-15 cyclotomic cosets

C_0 $(m_0 = 1,\ k_0 = 0)$	C_1 $(m_1 = 4,\ k_1 = 1)$	C_2 $(m_2 = 4,\ k_2 = 3)$	C_3 $(m_3 = 2,\ k_3 = 5)$	C_4 $(m_4 = 4,\ k_4 = 7)$
$\{f_0\}$	$\{f_1, f_2, f_4, f_8\}$	$\{f_3, f_6, f_{12}, f_9\}$	$\{f_5, f_{10}\}$	$\{f_7, f_{14}, f_{13}, f_{11}\}$

Let α be a root of the primitive polynomial x^4+x+1 and let $(\gamma_1, \gamma_1^2, \gamma_1^4, \gamma_1^8)$ be the basis of the field $GF(2^4)$, where $\gamma_1 = \alpha^3$. As for the fourth cyclotomic coset, the corresponding primitive polynomial of degree $m_3 = 2$ is $x^2 + x + 1$. α^5 is root of this latest polynomial and we can choose (γ_2, γ_2^2), where $\gamma_2 = \alpha^5$ as a basis of the linearized polynomial of degree $m_3 = 2$. These two basis satisfy the property $\gamma_i + \gamma_i^2 + ... + \gamma_i^{2^{mi-1}} = 1$.

The elements of $GF(16)$ can be expressed as the combination of the basis $(\gamma_1, \gamma_1^2, \gamma_1^4, \gamma_1^8)$ as following:

$$\begin{pmatrix} \alpha^0 \\ \alpha^1 \\ \alpha^2 \\ \alpha^3 \\ \alpha^4 \\ \alpha^5 \\ \alpha^6 \\ \alpha^7 \\ \alpha^8 \\ \alpha^9 \\ \alpha^{10} \\ \alpha^{11} \\ \alpha^{12} \\ \alpha^{13} \\ \alpha^{14} \end{pmatrix} = \begin{pmatrix} \gamma_1 + \gamma_1^2 + \gamma_1^4 + \gamma_1^8 \\ \gamma_1 + \gamma_1^8 \\ \gamma_1 + \gamma_1^2 \\ \gamma_1 \\ \gamma_1^2 + \gamma_1^4 \\ \gamma_1^2 + \gamma_1^8 \\ \gamma_1^2 \\ \gamma_1 + \gamma_1^2 + \gamma_1^4 \\ \gamma_1^4 + \gamma_1^8 \\ \gamma_1^8 \\ \gamma_1 + \gamma_1^4 \\ \gamma_1 + \gamma_1^2 + \gamma_1^8 \\ \gamma_1^4 \\ \gamma_1 + \gamma_1^4 + \gamma_1^8 \\ \gamma_1^2 + \gamma_1^4 + \gamma_1^8 \end{pmatrix}. \tag{C.5}$$

The decomposition of the polynomial $f(x)$ according to the following equation:

$$f(x) = \sum_{i=0}^{4} L_i(x^{k_i}),$$

and the substitution of x by α^i gives the frequency components F_j, for $i, j = 0, ..., 14$. Let us consider the development of some components.

$$
\begin{aligned}
f(\alpha^0) &= L_0(\alpha^0) + L_1(\alpha^0) + L_2(\alpha^0) + L_3(\alpha^0) + L_4(\alpha^0) \\
&= L_0(1) + L_1(\gamma_1) + L_1(\gamma_1^2) + L_1(\gamma_1^4) + L_1(\gamma_1^8) + L_2(\gamma_1) + L_2(\gamma_1^2) + L_2(\gamma_1^4) + L_2(\gamma_1^8) \\
&\quad + L_3(\gamma)_2 + L_3(\gamma_2^2) + L_4(\gamma_1) + L_4(\gamma_1^2) + L_4(\gamma_1^4) + L_4(\gamma_1^8), \\
f(\alpha^1) &= L_0(\alpha^0) + L_1(\alpha) + L_2(\alpha^3) + L_3(\alpha^5) + L_4(\alpha^7) \\
&= L_0(1) + L_1(\gamma_1) + L_1(\gamma_1^8) + L_2(\gamma_1) + L_3(\gamma_2) + L_4(\gamma_1) + L_4(\gamma_1^2) + L_4(\gamma_1^4), \\
f(\alpha^2) &= L_0(\alpha^0) + L_1(\alpha^2) + L_2(\alpha^6) + L_3(\alpha^{10}) + L_4(\alpha^{14}) \\
&= L_0(1) + L_1(\gamma_1) + L_1(\gamma_1^2) + L_2(\gamma_1^2) + L_3(\gamma_2^2) + L_4(\gamma_1^2) + L_4(\gamma_1^4) + L_4(\gamma_1^8), \\
f(\alpha^3) &= L_0(\alpha^0) + L_1(\alpha^3) + L_2(\alpha^9) + L_3(\alpha^{15}) + L_4(\alpha^6) \\
&= L_0(1) + L_1(\gamma_1) + L_2(\gamma_1^8) + L_3(1) + L_4(\gamma_1^2), \\
f(\alpha^4) &= L_0(\alpha^0) + L_1(\alpha^4) + L_2(\alpha^{12}) + L_3(\alpha^5) + L_4(\alpha^{13}) \\
&= L_0(1) + L_1(\gamma_1^2) + L_1(\gamma_1^4) + L_2(\gamma_1^4) + L_3(\gamma_2) + L_4(\gamma_1) + L_4(\gamma_1^4) + L_4(\gamma_1^8), \\
f(\alpha^5) &= L_0(\alpha^0) + L_1(\alpha^5) + L_2(\alpha^{15}) + L_3(\alpha^{10}) + L_4(\alpha^5) \\
&= L_0(1) + L_1(\gamma_1^2) + L_1(\gamma_1^8) + L_2(1) + L_3(\gamma_2^2) + L_4(\gamma_1^2) + L_4(\gamma_1^8), \\
f(\alpha^6) &= L_0(\alpha^0) + L_1(\alpha^6) + L_2(\alpha^3) + L_3(\alpha^{15}) + L_4(\alpha^{12}) \\
&= L_0(1) + L_1(\gamma_1^2) + L_2(\gamma_1) + L_3(1) + L_4(\gamma_1^4), \\
f(\alpha^7) &= L_0(\alpha^0) + L_1(\alpha^7) + L_2(\alpha^6) + L_3(\alpha^5) + L_4(\alpha^4) \\
&= L_0(1) + L_1(\gamma_1) + L_1(\gamma_1^2) + L_1(\gamma_1^4) + L_2(\gamma_1^2) + L_3(\gamma_2) + L_4(\gamma_1^2) + L_4(\gamma_1^4), \\
f(\alpha^8) &= L_0(\alpha^0) + L_1(\alpha^8) + L_2(\alpha^9) + L_3(\alpha^{10}) + L_4(\alpha^{11}) \\
&= L_0(1) + L_1(\gamma_1^4) + L_1(\gamma_1^8) + L_2(\gamma_1^8) + L_3(\gamma_2^2) + L_4(\gamma_1) + L_4(\gamma_1^2) + L_4(\gamma_1^8), \\
f(\alpha^9) &= L_0(\alpha^0) + L_1(\alpha^9) + L_2(\alpha^{12}) + L_3(\alpha^{15}) + L_4(\alpha^3) \\
&= L_0(1) + L_1(\gamma_1^8) + L_2(\gamma_1^4) + L_3(1) + L_4(\gamma_1), \\
f(\alpha^{10}) &= L_0(\alpha^0) + L_1(\alpha^{10}) + L_2(\alpha^{15}) + L_3(\alpha^5) + L_4(\alpha^{10}) \\
&= L_0(1) + L_1(\gamma_1) + L_1(\gamma_1^4) + L_2(1) + L_3(\gamma_2) + L_4(\gamma_1) + L_4(\gamma_1^4), \\
f(\alpha^{11}) &= L_0(\alpha^0) + L_1(\alpha^{11}) + L_2(\alpha^3) + L_3(\alpha^{10}) + L_4(\alpha^2) \\
&= L_0(1) + L_1(\gamma_1) + L_1(\gamma_1^2) + L_1(\gamma_1^8) + L_2(\gamma_1) + L_3(\gamma_2^2) + L_4(\gamma_1) + L_4(\gamma_1^2), \\
f(\alpha^{12}) &= L_0(\alpha^0) + L_1(\alpha^{12}) + L_2(\alpha^6) + L_3(\alpha^{15}) + L_4(\alpha^9) \\
&= L_0(1) + L_1(\gamma_1^4) + L_2(\gamma_1^2) + L_3(1) + L_4(\gamma_1^8), \\
f(\alpha^{13}) &= L_0(\alpha^0) + L_1(\alpha^{13}) + L_2(\alpha^9) + L_3(\alpha^5) + L_4(\alpha) \\
&= L_0(1) + L_1(\gamma_1) + L_1(\gamma_1^4) + L_1(\gamma_1^8) + L_2(\gamma_1^8) + L_3(\gamma_2) + L_4(\gamma_1) + L_4(\gamma_1^8), \\
f(\alpha^{14}) &= L_0(\alpha^0) + L_1(\alpha^{14}) + L_2(\alpha^{12}) + L_3(\alpha^{10}) + L_4(\alpha^8) \\
&= L_0(1) + L_1(\gamma_1^2) + L_1(\gamma_1^4) + L_1(\gamma_1^8) + L_2(\gamma_1^4) + L_3(\gamma_2^2) + L_4(\gamma_1^4) + L_4(\gamma_1^8),
\end{aligned}
$$

This system of equations can be written in matrix form as

$$F = \begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \\ F_{10} \\ F_{11} \\ F_{12} \\ F_{13} \\ F_{14} \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{pmatrix} L_0(1) \\ L_1(\gamma_1) \\ L_1(\gamma_1^2) \\ L_1(\gamma_1^4) \\ L_1(\gamma_1^8) \\ L_2(\gamma_1) \\ L_2(\gamma_1^2) \\ L_2(\gamma_1^4) \\ L_2(\gamma_1^8) \\ L_3(\gamma_2) \\ L_3(\gamma_2^2) \\ L_4(\gamma_1) \\ L_4(\gamma_1^2) \\ L_4(\gamma_1^4) \\ L_4(\gamma_1^8) \end{pmatrix}. \tag{C.6}$$

Each L_i constitutes a $(m_i \times m_i)$-matrix. By developing the L_is, Equation C.6 is equivalent to

$$F = A \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma_1^1 & \gamma_1^2 & \gamma_1^4 & \gamma_1^8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma_1^2 & \gamma_1^4 & \gamma_1^8 & \gamma_1^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma_1^4 & \gamma_1^8 & \gamma_1^1 & \gamma_1^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma_1^8 & \gamma_1^1 & \gamma_1^2 & \gamma_1^4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \gamma_1^1 & \gamma_1^2 & \gamma_1^4 & \gamma_1^8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \gamma_1^2 & \gamma_1^4 & \gamma_1^8 & \gamma_1^1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \gamma_1^4 & \gamma_1^8 & \gamma_1^1 & \gamma_1^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \gamma_1^8 & \gamma_1^1 & \gamma_1^2 & \gamma_1^4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \gamma_2^1 & \gamma_2^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \gamma_2^2 & \gamma_2^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \gamma_1^1 & \gamma_1^2 & \gamma_1^4 & \gamma_1^8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \gamma_1^2 & \gamma_1^4 & \gamma_1^8 & \gamma_1^1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \gamma_1^4 & \gamma_1^8 & \gamma_1^1 & \gamma_1^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \gamma_1^8 & \gamma_1^1 & \gamma_1^2 & \gamma_1^4 \end{bmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_4 \\ f_8 \\ f_3 \\ f_6 \\ f_{12} \\ f_9 \\ f_5 \\ f_{10} \\ f_7 \\ f_{14} \\ f_{13} \\ f_{11} \end{pmatrix},$$

which can be written as the following form:

$$F = ALf. \tag{C.7}$$

The multiplication of matrix L by matrix f is equivalent to four cyclic convolutions (three four-point cyclic convolutions and one two-point cyclic convolution) of L_i by the corresponding cyclotomic coset of f_i.

The four-point cyclic convolutions

$$\begin{pmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{pmatrix} = \begin{pmatrix} \gamma_1^1 & \gamma_1^2 & \gamma_1^4 & \gamma_1^8 \\ \gamma_1^2 & \gamma_1^4 & \gamma_1^8 & \gamma_1^1 \\ \gamma_1^4 & \gamma_1^8 & \gamma_1^1 & \gamma_1^2 \\ \gamma_1^8 & \gamma_1^1 & \gamma_1^2 & \gamma_1^4 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_4 \\ f_8 \end{pmatrix}, \quad \begin{pmatrix} S_5 \\ S_6 \\ S_7 \\ S_8 \end{pmatrix} = \begin{pmatrix} \gamma_1^1 & \gamma_1^2 & \gamma_1^4 & \gamma_1^8 \\ \gamma_1^2 & \gamma_1^4 & \gamma_1^8 & \gamma_1^1 \\ \gamma_1^4 & \gamma_1^8 & \gamma_1^1 & \gamma_1^2 \\ \gamma_1^8 & \gamma_1^1 & \gamma_1^2 & \gamma_1^4 \end{pmatrix} \begin{pmatrix} f_3 \\ f_6 \\ f_{12} \\ f_9 \end{pmatrix}$$

$$\textit{and} \qquad \begin{pmatrix} S_{11} \\ S_{12} \\ S_{13} \\ S_{14} \end{pmatrix} = \begin{pmatrix} \gamma_1^1 & \gamma_1^2 & \gamma_1^4 & \gamma_1^8 \\ \gamma_1^2 & \gamma_1^4 & \gamma_1^8 & \gamma_1^1 \\ \gamma_1^4 & \gamma_1^8 & \gamma_1^1 & \gamma_1^2 \\ \gamma_1^8 & \gamma_1^1 & \gamma_1^2 & \gamma_1^4 \end{pmatrix} \begin{pmatrix} f_7 \\ f_{14} \\ f_{13} \\ f_{11} \end{pmatrix},$$

can be computed using short convolution algorithms described in [43]. For example

$$\begin{pmatrix} S_1 \\ S_2 \\ S_4 \\ S_8 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} \gamma_1 \\ \gamma_1^8 \\ \gamma_1^4 \\ \gamma_1^2 \end{pmatrix} . \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_4 \\ f_8 \end{pmatrix} \right) \tag{C.8}$$

Similarly, the two-point cyclic convolution

$$\begin{pmatrix} S_9 \\ S_{10} \end{pmatrix} = \begin{pmatrix} \gamma_2^1 & \gamma_2^2 \\ \gamma_2^2 & \gamma_2^1 \end{pmatrix} \begin{pmatrix} f_5 \\ f_{10} \end{pmatrix}$$

can be computed using short two-point convolution algorithm.

$$\begin{pmatrix} S_9 \\ S_{10} \end{pmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \gamma_2 \\ \gamma_2^2 \end{bmatrix} . \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} f_5 \\ f_{10} \end{bmatrix} \right).$$

As previously shown, the FFT can be written with a simple form as

$$\mathbf{F{=}AQ(B.(PF))}, \tag{C.9}$$

where

$$Q = \begin{pmatrix} 1 & & & 0 & \\ & Q_1 & & & \\ & & Q_1 & & \\ & & & Q_2 & \\ & 0 & & & Q_1 \end{pmatrix}, \quad Q_1 = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

and

$$Q_2 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix},$$

$$B = \begin{pmatrix} 1 & & & 0 & \\ & B_1 & & & \\ & & B_1 & & \\ & & & B_2 & \\ & 0 & & & B_1 \end{pmatrix} \begin{pmatrix} 1 \\ \Gamma_1 \\ \Gamma_1 \\ \Gamma_2 \\ \Gamma_1 \end{pmatrix},$$

with,

$$B_1 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \textit{and} \quad B_2 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \Gamma_1 = \begin{pmatrix} \gamma_1 \\ \gamma_1^8 \\ \gamma_1^4 \\ \gamma_1^2 \end{pmatrix} \quad \textit{and} \quad \Gamma_2 = \begin{pmatrix} \gamma_2 \\ \gamma_2^2 \end{pmatrix},$$

$$Pf = \begin{pmatrix} 1 & & & 0 & \\ & P_1 & & & \\ & & P_1 & & \\ & & & P_2 & \\ & 0 & & & P_1 \end{pmatrix} \begin{pmatrix} f_0 \\ f_{C_1} \\ f_{C_2} \\ f_{C_3} \\ f_{C_4} \end{pmatrix}, \quad P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad \textit{and} \quad P_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

f_{C_1}, f_{C_2}, f_{C_3}, f_{C_4} are the transposed vector containing the elements of cyclotomic cosets C_1, C_2, C_3, C_4 respectively.

As shown in Chapter 4, the hardware implementation of the matrix computations of Equation C.9 can be tackled from right to left, that is, the multiplication Pf is performed firstly, then the computation of matrix B, the multiplication of the result by the matrix Q and finally the multiplication of the latest result by matrix A.

List of Figures

List of Tables

Bibliography

[1] J. Mitola, *Software Radio Architecture*, Wiley, 2000

[2] W. Tuttlebee, *Software Defined Radio: Enabling Technologies*, Wiley, 2002.

[3] J. H. Reed, *SoftWare Radio: A Modern Appoach to Radio Engineering,* Prentice Hall, 2002.

[4] R. H. Walden, *Performance Trends for Analog-to-Digital Converters*, IEEE Commun. Mag., Feb. 1999, pp. 96-101.

[5] H. Tsurumi, Y. Suzuki, *Broadband RF stage Architecture for Software-Defined Radio in Handheld Applications*, IEEE Communications Magazine, February 1999.

[6] T. Hentschell, M. Henker and G. Fettweis, *The Software Front-End of Software Radio Terminals* IEEE Personal Communications, August 1999.

[7] T. Hentschell, G. Fettweis and M. Bronzel, *Channelization and Sample Rate Adaptation in Software Radio Terminals*, 3rd ACTS Mobile Commun. Summit, pp. 121-126, Rhodos, Greece, June 1998.

[8] F. Jondral, *Parameterization-a technique for SDR Implementation*, Chapter 8 of "Software Defined Radio Enabling Technologies" edited by W.Tuttlebee, Wiley, 2002.

[9] R. L. Lackey and D. W. Upmal, *Speakeasy: The Mlitary Software Radio*, IEEE Communications Magazine, May 1995.

[10] P. G. Cook and W. Bonser, *Architectural Overview of the SPEAKeasy System*, IEEE Communications Magazine, April 1999.

[11] P. A. Eyermann and M. A. Powell, *Maturing the Software Communications Architecture for JTRS*, Proceedings of the IEEE Military Communications Conference, vol 1, 2001.

[12] B. Tarver, E. Christensen and A. Miller et al, *Digital Modular Radio (DMR) as a Maritime/FIxed Joint Tactical Radio System (JTRS)*, Proceedings of the IEEE Military Communications Conference, vol 1, 2001.

[13] J. Mitola III, *SDR Architecture Refinement for JTRS*, Proceedings of the IEEE Military Communications Conference, vol 1, 2001.

[14] Free Software Foundation, *GNU Radio, The GNU Software Radio*, http://www.gnu.org/software/gnuradio.

[15] *www.sdrforum.org.*

[16] W. Tuttlebee, *Software Radio Technology: A European Perspective*, IEEE Communications Magazine, February 1999.

[17] D. Ikonomou, J. M. Pereira and J. da Silva, *EU funded R-D on Re-configurable Radio Systems and Networks: The story so Far*, Infowin Thematic Issue Mobile Communications, ACTS, 2000.

[18] *http://e2r2.motlabs.com/.*

[19] Arnd-Ragnar Rhiemeier, *Benefits and limits of parameterized channel coding for software radio*, In Pro. 2^{nd} Workshop on Software Radio, Karlsrhue , Germany, March 2002.

[20] F. Jondral, A. Wiesler and R. Machauer, *A Software Defined Radio Structure for 2nd and 3rd Genaration Mobile Communications Standards*, IEEE 6th Int. Symp. On Spread-Spectrum Tech. and Appli., New Jersey, USA, Sept. 6-8. 2000.

[21] J. Palicot, C. Roland, *FFT: a basic Function for a Reconfigurable Receiver*, ICT'2003, Feb. 2003, Papeete, Tahiti.

[22] C. Moy, J. Palicot, V. Rodriguez and D. Giri, *Optimal Determination of Common Operators for Multi-Standards Software-Defined Radio*, IEEE WSR'2006, Karlsrhue, Germany, March 2006.

[23] S. Kirpatrick, C. D. Gelatt and M. P. Vecchi, *Optimization by Simulated Annealing*, Science, 220, 4598, 671-680, 1983.

[24] V. Rodriguez, C. Moy and J. Palicot, *Install or invoke?: The optimal trade-off between performance and cost in the design of multi-standard reconfigurable radios*, Wirel. Commun. Mob. Comput. 2006; 6:1-14.

[25] T. Magnanti and R. Wong, *Network design and transportation planning: models and algorithms*, Transportation Science, vol. 18, no. 1, pp. 1-56, 1984.

[26] S. Bilavarn, G. Gogniat, J-L. Philippe, *Fast Prototyping of Reconfigurable Architectures: An Estimation And Exploration Methodology from System-Level Specifications* Eleventh ACM International Symposium on Field-Programmable Gate Arrays Monterey, California, February 23-25 2003.

[27] J. Cavallaro and M. Vaya, *VITURBO : A Reconfigurable Architecture for Viterbi and Turbo Coding*, In IEEE International Conference on Acoustics, speech, and Signal Processsing (ICASSP), China, April 2003.

[28] A. Wiesler, F. Jondral, *A Software Radio for Second-and Third-Generation Mobiles Systems*, IEEE Transactions On Vehicular Technology, Vol. 51, NO. 4,July 2002.

[29] M. Ghozzi, *Intérêt des Techniques de Paramétrization pour des Archirtectures Radio Logicielle reconfigurables*, Master thesis of Rennes1 University/Supélec, June 2004.

[30] *www.analog.com.*

[31] L. Alaus, D. Noguet and J. Palicot, *A Reconfigurable Linear Feedback Shift Register Operator for Software Defined Radio Terminal*, ISWPC, May 2008, Santorini, Greece.

[32] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., Vol. 19. pp. 297-301, April 1965.

[33] E. R. Ferrara, C. F. N. Cowan and P. M. Grant, *Frequency Domain Adaptive Filtering*, Prentice-Hall, Jan. 1995.

[34] D. Mansour, A. H. Gray, *Unconstrained Frequency-Domain Adaptive Filter*, IEEE Trans. on ASSP-30, N°5, October 1982.

[35] T. A. Schirtzinger, X. Li and W. K. Jenkins, *A Comparison of three algorithms for blind equalization based on the constant modulus error criterion*, ICASSP'95, vol. 2, NY, USA, 1995.

[36] K. Beberidis, J. Palicot, *A block Quasi-Newton Algorithm implemented in the frequency domain*, EUSIPCO'96, Trieste, September 1995.

[37] K. Beberidis, J. Palicot, *A frequency domain decision feedback equalizer for multi path echo cancellation*, Globecom'95, Singapore, November 1995.

[38] K. Beberidis, A. Marava, P. Karaivazoglou and J. Palicot, *Robust and Fast Converging Decision Feedback Equalizer Based on a New Adaptive Semi Blind Estimation Algorithm*, Globecom'01, San Antonio, US, November 2001.

[39] D. Chase, *A Class of Algorithms for Decoding Block Codes with Channel Measurement Information*, IEEE Trans. on Information Theory, vol. 18, pp. 170-182, Jan. 1972.

[40] R. Pyndiah, A. Glavieux, A. Picart and S. Jacq, *Near optimum decoding of product codes*, Proc. IEEE Global Telecommunications Conference (GLOBECOM'94), Nov. 1994.

[41] M. Davey and D. J. C. Mackay, *Low-density parity check codes over GF (q)*, IEEE Commun. Lett., vol. 2, no. 6, pp. 165-167, June 1998.

[42] D. Declercq and M. Fossorier, *Decoding Algorithms for Nonbinary LDPC Codes over GF(q)*, IEEE. Trans. On Communications, Vol. 35, No. 4, April 2007.

[43] R. Blahut, *Theory and Practice of Error Control Codes*, Reading, Massachusetts : Addison-Wesley, 1983.

[44] *R. Blahut, Algebraic Codes for Data Transmission*, Cambridge University press, 2003.

[45] J. M. Pollard, *The fast Fourier transform in a finite field*, IEEE Trans. Comput., vol. 25, pp. 365-374, Apr. 1971.

[46] W. C. Gore, *Transmitting Binary Symbols with Reed-Solomon Codes*, Proceedings of Princeton Conference on Information Sciences and Systems, Princeton, NJ, 1973, pp. 495-497.

[47] A. Michelson, *A Fast Transform in Some Galois Field and an application to Decoding Reed-Solomon Codes*, IEEE International Symposium on Information Theory, Ronneby, Sweden, 1976, p. 49.

[48] A. Lempel and S. Winograd, *A New Approach of Erroe Correcting COdes*, IEEE Trans. Inf. Theory IT-23, 503-508, 1977.

[49] R. T. Chien and D. M. Choy, *Algebraic Generalization of BCH-Goppa-Helgert Codes*, IEEE Trans. Inf. Theory IT-21, 70-79, 1975.

[50] R. E. Blahut, *Transform Decoding without Transform*, presented at the Tenth IEEE Communication Theory Workshop, Cypress Gardens, FL, 1980.

[51] S. M. Reddy and J.P. Robinson, *Random Error and Burst Correction by Iterated Codes*, IEEE Trans. Inf. Theory, vol IT-18, p. 172-185, Jan. 1972.

[52] S. B. Wicker and V. K. Bhargava, *Reed-Solomon codes and their applications*, New York: IEEE Press, 1994.

[53] C. Basile et al., *The US HDTV standard the grand*, IEEE Spectrum, vol. 32, pp. 36-45, April 1995.

[54] W. W. Peterson, *Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes*, IEEE Trans. Inf. Theory, vol IT-6, p. 459-470, 1960.

[55] D. C. Gorenstein and N. Zierler, *A Class of Error-Correcting Codes in p^m Symbols*, Journal of the Society of Industrial and Applied Mathematics, vol. 9, p. 207-214, 1961.

[56] R. T. Chien, *Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes*, IEEE Trans. Info. Theory, Vol. IT-10, pp. 357-363, October 1964.

[57] E. R. Berlekamp, *Algebraic coding theory*, McGraw-Hill Book Co., Inc., New York, 1986.

[58] J. L. Massey, *Shift Register Synthesis and BCH decoding*, IEEE Trans. Inf. Theory, vol IT-15, p. 122-127, 1972.

[59] G. D. Forney, *On Decoding BCH Codes*, IEEE Trans. Info. Theory, Vol. IT-11, pp. 549-557, 1965.

[60] *R. E. Blahut, Transform Decoding without Transform*, presented at the Tenth IEEE Communication Theory Workshop, Cypress Gardens, FL, 1980.

[61] R. E. Blahut, *A universal Reed-Solomon decoder*, IEEE. Trans. Comput., vol.C-34,pp. 150-158. Mar. 1984.

[62] C. M. Rader, *The number theoretic DFT and exact discrete convolution*, IEEE Arden House Workshop on Digital Signal Processing, Harriman, N. Y., Jan. 11. 1972.

[63] C. M. Rader, *Discrete convolution via Mersenne transforms*, IEEE Trans. Comput., vol. C-21, pp. 1269-1273, Dec. 1972.

[64] R. C. Agarwal and C. S. Burrus, *Fast digital convolution using Fermat transforms*, in Southwest IEEE Conf. Rec., Houston, Tex., pp. 538-543, Apr. 1973.

[65] J. Justesen, *On the Complexity of Decoding Reed-Solomon Codes*, IEEE Trans. Inform. Therory, vol. IT-22, pp. 237-238 Mar. 1976.

[66] S. W. Golomb, *On the sum of the reciprocals of the Fermat numbers and related irrationalities*, Canad. J. Math. 15(1963),475-478.

[67] R. C. Agarwal and C. S. Burrus, *Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering*, IEEE Trans. on Acou. Spee. and Sig. Proces., vol. ASSP-22, no. 2, Apr. 1974.

[68] I. S. Reed, T. K. Truong and L.R. Welch, *The Fast Decoding of Reed-Solomon Codes Using Fermat Transforms*, IEEE Trans. Inform. Theory, vol. IT-24, no. 4, July 1978.

[69] M. R. Best; H. F. A. Roefs, "Technical assistance channel coding investigation (spacecraft Telemetry)" [Final Report] 1981.

[70] M. Frigo and S. G. Johnson, *FFTW: An Adaptive Software Architecture for the FFT*, ICASSP conference proceedings, 3:1381-1384, 1998.

[71] *Analog Devices DSP selection Guide 2002 Edition*, Analog Devices, 2002.

[72] *Motorola DSP 56600 16 bit DSP Family Datasheet*, Motorola Ltd., 2002.

[73] B. M. Baas, *A Low-Power, High-Performance 1024-point FFT Processor*, IEEE Journal of Solid State Circuits. pp. 380-387, 1999.

[74] *FFT Megacore Function User Guide*, Altera Ltd., 2006.

[75] *Xilinx 1024-point FFT/IFFT Core Datasheet*, Xilinx Ltd., 2002.

[76] C. Y. Wang, C. B. Kuo and J. Y. Jou, *Hybrid Wordlength Optimization Methods of Pipelined FFT Processors*, IEEE Transactions on Computers, vol. 56, no. 8, pp. 1105-1118, Aug., 2007

[77] *http://www.systemc.org.*

[78] *http://www.celoxica.com.*

[79] *http://www.fftw.org.*

[80] H. Sorensen, M. Heideman and C. Burrus, *On computing the split-radix FFT*, IEEE. Trans. Acoust., Speech, Signal Process., vol. 34, no. 1, pp. 152-156, Jan. 1986.

[81] M. A. Soderstrand, W. K. Jenkins, G. A. Gulien and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signla Processing*, IEEE Press, New York, 1986.

[82] K. M. Elleithy and M. A. Bayoumi, *Fast and Flexible Architecture for RNS Arithmetic Decoding*, IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing, vol. 39, pp. 191-201, Apr. 1977.

[83] R. Zimmermann, *Binary Adder Architecture for Cell-Based VLSI and their Synthesis*, PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, Hartung-Gorre Verlag, 1998.

[84] R. E. Ladner and M. J. Fischer, *Parallel Prefix Compuation*, J .ACM, Vol. 27, No. 4, pp. 831-838, Oct. 1980.

[85] R. Zimmermann, *Efficient VLSI Implementation of modulo* $(2^n \pm 1)$ *Addition and Multipication*, Proc. 14th IEEE Symp. Computer Arithmetic, pp. 158-167, Apr. 1999.

[86] H. T. Vergos, C. Efstathiou and D. Nikolos, *Diminished-One Modulo* $(2^n + 1)$ *Adder Design*, IEEE. Tarns. Computers, vol. 51, no. 12, pp. 1389-1399, Dec. 2002.

[87] C. Efstathiou, H. T. Vefos and Dimitris Nikolos, *Modulo* $(2^n \pm 1)$ *Adder Design Using Select-Prefix Blocks*, IEEE. Tarns. Computers, vol. 52, no. 11, pp. 1399-1406, Nov. 2003.

[88] L. Kalamboukas, D. Nkolos, C. Efsthathiou, H. T. Vergosand and J. Kalamatianos, *High-Speed Parallel-Prefix Modulo* $(2^n - 1)$ *Adders*, IEEE. Tarns. Computers, vol. 49, no. 7, pp. 673-680, July 2000.

[89] Z. Wang, G. A. Gulien and W. C. Miller, *An efficient tree architecture for modulo* $2^n + 1$ *multiplication*, J. VLSI Signal Processing Systems, 14(3): 241-248, Dec. 1996.

[90] Jean-Luc Beuchat, *Some Modular Adders and Multipliers for Field Programmable Gate Arrays*, Proceedings of the 17th International Parallel and Distributed Processing Symposium. IEEE Computer Society,2003.

[91] X. Lai and J. L. Massey, *A Proposal for a New Block Encryption Standard*, In Advances in Cryptology-EUROCRYPT'90, pages 389-404, Berlin, Germany: Springer-Verlag, 1990.

[92] A. V. Curiger, H. Bonnenberg and H. Kaeslin, *Regular VLSI Architecture for Multiplication Modulo* $(2^n + 1)$, IEEE J. of Solid-State Circuits, Vol. 26, No. 7, pp. 990-994, July 1991.

[93] Jean-Luc Beuchat, *Modular Multiplication for FPGA Implementation of the IDEA Block Cipher*, Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03).

[94] F. J. Taylor, *Large moduli multipliers for signal processin*, IEEE Trans. Circuits Syst., vol. C-28, pp. 731-736, July 1981.

[95] G. A. Julien, *Implementation of multiplication, modulo a prime number, with applications to number theoretic transforms*, IEEE Trans. Comp., vol. C-29, pp. 899-905, Oct. 1980.

[96] Y. Ma, *A Simplified Architecture for Modulo* $(2^n + 1)$ *Multiplication*, IEEE Transactions on Computers, 47(3) :333-337, 1998.

[97] William W.H. Yu and Shanzhen Xing *Fixed-Point Multiplier Evaluation and Design with FPGA*, Proceedings of SPIE, vol. 3844, September 1999.

[98] B. Liu, T. Thong, *Fixed-Point Fast Fourier Transform Error Analysis*, IEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP-24, pp. 563-573, December 1976.

[99] A. Elterich, W. Stammler, *Error Analysis and Resulting Structural Improvements For Fixed Point FFTs*, Proc. ICASSP, NY, pp. 1419-1422, 1988.

[100] A. V. Oppenheim and C. J. Weinstein, *Effects of Finite Register Length in Digital Filtering and The Fast Fourier Transform*, Proc. IEEE, vol. 60, pp. 957-976, August 1972.

[101] A. Jalali, *Study and design of a digital Modem devoted to Multicarrier Modulation*, University of Rennes 1, Ph.D. Thesis, April 1998.

[102] Y. Wang and X. Zhu, "A Fast Algorithm for the Fourier Transform Over Finite Fields and its VLSI Implementation," *IEEE. Journal on Selected Areas in Communications,* vol.6, no. 3, April 1988.

[103] T. K. Truong, J. H. Jeng and I. S. Reed, "Fast Algorithm for Computing the Roots of Error Locator Polynomials up to Degree 11 in Reed-Solomon Decoders," *IEEE. Trans. Commun.,* vol. 49, no. 5, pp. 779-783, 2001.

[104] S. V. Fedorenko and P. V. Trifonov, "Finding Roots of Polynomials over Finite Fields," *IEEE. Trans. Commun.,* vol. 50, no. 11, pp. 1709-1711, 2002.

[105] S. V. Fedorenko and P. V. Trifonov, "A method for Fast Computation of the Fast Fourier Transform over a Finite Field," *Problems of Information Transmission,* 39(3):231-238, July-September 2003. Translation of Problemy Peredachi Informatisii.

[106] V. B. Afanasyev, "On Complexity of FFT over Finite Field," *in Proc. 6th Joint Swedish-Russian Int.Workshop on Information Theory,* Mölle, Sweden, 1993, pp. 315-319.

[107] T. K. Truong, P. D. Chen, L. J. Wang, I. S. Reed and Y. Chang, "Fast Prime factor, discrete Fourier transform algorithms over $GF(2^m)$ for $8 \leq m \leq 10$," *Inf. Sci.,* vol. 176, no. 1, pp. 1-26, January 2006.

[108] Elena. Costa, S. V. Fedorenko and P. V. Trifonov, "On Computing the Syndrome Polynomial in Reed-Solomon Decoder," *International ITG Conference on Sourcec and Channel Coding No. 5,* vol. 15, no. 4, pp. 337-342, Germany 2004.

[109] S. Gao, "A New Algorithm For Decoding Reed-Solomon Codes," in *Communications, Information and Network Security,* V. Bhargava, H. V. Poor, V. Tarokh, and S. Yoon, Eds. Norwell, vol. 712, pp. 55-68, 2003.

[110] T. K. Truong, P. D. Chen, L. J. Wang and T. C. Cheng, "Fast Transoform for Decoding Both Errors and Erasures of Reed-Solomon Codes over $GF(2^m)$ for $8 \leq m \leq 10$," *IEEE Trans. on Communications,* vol. 54, no. 2, pp. 181-186, February 2006.

[111] P. A. Scott, S. E. Tavares and L. E. Peppard, "A Fast VLSI Multiplier for $GF(2^m)$," in *IEEE Journal on Selcted Areas in Communications,* vol. SAC-4, No. 1, January 1986.

[112] C. S. Wallace, *A Suggestion For A Fast Multiplier*, IEEETransactions on Computers, vol. EC13, pp. 14-17, Feb. 1964.

[113] L. Dadda, *Some Shcemes for Parallel Multipliers*, Alta Frequenza, vol. 34, pp. 349-356, 1965.

[114] E. D. Mastrovito, *VLSI Designs for Multiplications over Finite Fields* $GF(2^m)$, in Proc. Sixth Int. Conf. Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes (AAECC-6), pp. 297-309, 1988.

[115] C. S. Yeh, I. S. Reed, and T. K. Troung, *Systolic Multipliers For Finite Fields GF(2m)*, IEEE Transactions on Computers, vol. c-33, pp. 357-360, Apr. 1984.

[116] L. Gao and K. K. Parhi, *Custom VLSI Design of Efficient Low Latency and Low Power Finite Field Multiplier for Reed-Solomon Codec*, In IEEE International Symposium on Circuits and Systems ISCAS'01, pp. IV 574-577, 2001.

[117] Texas Instruments, *TMS320C64x Technical Overview.*

[118] W. Drescher, G. Fettweis and K. Bachmann, *VLSI Architecture For Non- Sequential Inversion Over* $GF(2^m)$ *Using the Euclidean Algorithm*, in Int. Conf. On Signal Processing Applications and Technology, 1997

[119] J. Garcia and M. J. Schulte, *A Combined 16-bit Binary and Dual Galois Field Multiplier*, In IEEE International Symposium on Circuits and Systems ISCAS'02, pp.63-68, 2002.