Architecture Reconfigurable pour un Equipement Radio Multistandard

Laurent Alaus

N° d'ordre : 4122 **ANNÉE 2010**

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Traitement du Signal et Télécommunications

Ecole doctorale : Matisse

présentée par

Laurent ALAUS

Préparée à l'unité de recherche : SCEE-SUPELEC/IETR UMR 6164
Nom développé de l'unité : Institut d'Electronique et de Télécommunications de Rennes
Composante universitaire : S.P.M.

Architecture Reconfigurable pour un Equipement Radio Multistandard.

Thèse soutenue à SUPELEC-Rennes le 28 Mai 2010

devant le jury composé de :

Olivier SENTIEYS
Président, Professeur, Université de Rennes-I, Rennes, France

Guy GOGNIAT
Rapporteur, Professeur, Université de Bretagne Sud, Lorient, France

Lionel TORRES
Rapporteur, Professeur, Université de Montpellier-II, Montpellier, France

Adel GHAZEL
Examinateur, Professeur, SUPCOM, Tunis, Tunisie.

Christian ROLAND
Invité, Maître de Conférence, Université de Bretagne Sud, Lorient, France

Jacques PALICOT
Directeur de thèse, Professeur, SUPELEC/IETR, Rennes, France

Dominique NOGUET
Co-Directeur de thèse, Docteur, Ingénieur-Chercheur, CEA-LETI, Grenoble, France

Remerciements

Je remercie Olivier SENTIEYS, Professeur à l'Université de Rennes-I qui me fait l'honneur de présider ce jury.

Je remercie Guy GOGNIAT, Professeur de l'Université de Bretagne Sud et Lionel TORRES, Professeur de l'Université de Montpellier-II, d'avoir bien voulu accepter la charge de rapporteur.

Je remercie également Adel GHAZEL, Professeur à SUPCOM, Tunis et Christian ROLAND, Maître de conférence à l'Université de Bretagne Sud, d'avoir bien voulu juger ce travail.

Je remercie enfin Jacques PALICOT, Professeur à SUPELEC, directeur du laboratoire SCEE et Dominique NOGUET, Ingénieur-Chercheur au CEA-LETI d'avoir diriger ma thèse mais aussi de m'avoir accueilli au sein de leur laboratoire respectif et de m'avoir conseillé et épaulé durant ces trois années de thèse.

Table des matières

Introduction

Nous assistons actuellement à la multiplication des normes et des standards de télécommunications. Le nombre croissant des standards normalisés permet d'élargir l'éventail des offres et des services disponibles pour chaque consommateur. Si cette prolifération des normes est un plus non négligeable pour les utilisateurs, elle engendre une complexité accrue des terminaux à considérer. En effet, différents par les services proposés, les standards se distinguent par les bandes fréquentielles utilisées, les modulations, les codages et un ensemble de paramètres qui est spécifique à chacun d'eux. Cette multiplication des services et des standards les mettant en oeuvre à donné naissance à la notion de Handover vertical. Celui-ci vient s'ajouter au concept déjà existant de Handover horizontal et qualifie la capacité d'un terminal à " naviguer " entre les différents réseaux. Afin d'exploiter ce Handover généralisé (combinaison du Handover vertical et horizontal), un Terminal Radio Intelligent a été défini comme un terminal qui se reconfigure en fonction des interactions avec l'environnement dans lequel il opère. La Radio Intelligente se divise en trois grandes parties :

- Le Sensing, la capacité à détecter les signaux et à profiter du spectre radio.
- L'Adaptabilité, la prise de décision vis-à-vis des résultats du Sensing.
- La Reconfigurabilité, la possibilité de reconfigurer notre équipement pour assurer les changements requis.

De cette multiplication des normes, le besoin en terminaux multistandard est apparu.

Les possibilités usuelles pour réaliser un terminal multistandard se sont limitées pendant longtemps à la méthode dite "Velcro", c'est à dire la coexistence de toutes les entités architecturales nécessaires et d'un simple "Switch", permettant de passer d'un mode à l'autre. La méthode "Velcro" permet de traiter plusieurs standards par le biais d'un seul et même terminal. L'intérêt d'une telle méthode est d'accéder à un changement rapide entre les diverses chaînes de traitements. Cependant son évolutivité est limitée par les standards considérés et la complexité de sa mise en oeuvre reste maximale. Ainsi, les designers de terminaux multistandard, avaient comme unique solution de concevoir un circuit spécifique pour réaliser chaque chaîne. Avec l'apparition des systèmes programmés via un microprocesseur, dans les terminaux de communication actuels, les traitements "critiques" et récurrents sont exécutés sur des accélérateurs matériels dédiés (ASIC) et les calculs peu complexes sont exécutés par des processeurs spécialisés (applications de traitement du signal, DSP). Néanmoins, depuis quelques années une alternative de conception existe par la mise en oeuvre de radios dites logicielles.

mitola en 1995 [Mit00], a défini les principaux concepts de la radio logicielle. Celle-ci veut apporter une solution à la co-existence simultanée de protocoles multiples en spécifiant des architectures reconfigurables dynamiquement pour les terminaux dans les environnements radio sans fil. L'atout de la Radio logicielle est de pouvoir ajouter, mettre à jour des fonctionnalités du système radio par logiciel grâce à la reprogrammabilité des processeurs sans ajouter les coûts de changement du matériel.

L'architecture "idéale" se compose :

- une antenne large bande
- un convertisseur analogique numérique à haute fréquence d'échantillonnage et large bande
- un processeur généraliste permettant de réaliser les fonctions radio de divers standards et des interfaces associées.

Cependant, les technologies actuelles ne permettent d'arriver à une telle radio. Parmi ces limitations, nous pouvons citer les limites de performances des convertisseurs, la limitation en termes de bande passante des amplificateurs, le manque de puissance de calcul et la forte consommation des processeurs. Ainsi, une approche réaliste a été développé et conduit à la définition de la radio logicielle "restreinte" (SDR, Software Defined Radio) où les architectures reconfigurables apportent une réponse à la multiplicité des standards de communication.

Parmi les architectures reconfigurables, les plateformes hétérogènes, combinant DSP, μP et FPGA, connaissent un grand succès et deviennent une solution pouvant allier flexibilité logicielle et matérielle aussi bien à la conception que durant l'utilisation. En effet, grâce à un réseau reprogrammable à des niveaux de granularité différents et à l'association avec des processeurs de traitement du signal, ces plateformes représentent une solution intermédiaire qui combine la flexibilité de la programmation et puissance de calcul des architectures spécialisées.

L'intégration de terminaux multistandard mettant en oeuvre de telles plateformes, spécifiquement dans le cadre de la radio intelligente n'est pas pour autant garantie. En effet, celle-ci suppose la capacité du terminal à pouvoir modifier les paramètres d'émission et de réception d'un standard voire à changer l'intégralité du standard considéré. Ces modifications doivent être réalisées en temps réel et obéir à des contraintes imposées par chaque norme. Parmi les techniques actuelles, la paramétrisation est une des techniques clés de la radio logicielle. L'objectif général des études de paramétrisation est d'identifier des traitements dits communs entre les standards afin de définir de nouveaux macro blocs génériques pouvant être implémentés en Software et Hardware afin d'être indépendants des verrous technologiques. Jusqu'à présent, la technique de paramétrisation développée est celle des Fonctions Communes. Celles-ci utilisent la similarité dans l'enchaînement des fonctionnalités requises par les chaines de télécommunication afin de définir les différents éléments communs. Cette technique montre ces limites quant à l'évolutivité de l'architecture développée qui devient spécifique aux standards considérés lors du design initial. L'ajout ou la modification du terminal nécessite par conséquent

la redéfinition intégrale du design.

Problématique de l'étude Proposer un terminal multistandard reconfigurable pour la radio intelligente n'apparait pas comme une chose triviale. La problématique de nos travaux de recherche résidera dans la possibilité de proposer une technique de design d'un tel terminal. Afin de proposer une méthode évolutive, il nous est apparu pertinent d'essayer de trouver une technique de conception de haut niveau indépendante d'une technologie spécifique, pouvant bénéficier des avancées sur les capacités de reconfiguration des différentes cibles telles que la reconfiguration partielle des FPGA. Nous désirons avancer une méthodologie de plus haut niveau qui définirait un terminal multistandard indépendamment de la plateforme hétérogène tout en rendant le terminal obtenu évolutif.

Les Techniques de Paramétrisation répondent aux critères de reconfiguration en temps réel et d'indépendance vis à vis de la cible choisie. Par simple téléchargement de paramètres, elles permettent d'implémenter et de modifier un design sans "toucher" à l'"architecture" prédéfinie. Dans ce contexte, nous recherchons un nouveau type d'entités communes à considérer afin de poser les bases d'une alternative en termes de paramétrisation aux fonctions communes. Outre la définition d'une nouvelle méthode, la problématique liée à nos travaux et non seulement d'arriver à paramétriser la plus grande partie possible du terminal mais aussi à expliciter une méthodologie d'exécution des entités communes.

Contexte de l'Etude Les travaux présentés dans ce document ont été réalisés en collaboration entre le Commissariat à l'Energie Atomique de Grenoble (CEA-LETI Minatec) et l'Institut d'Electronique et de Télécommunications de Rennes (IETR-SCEE, Signal, Communication et Electronique Embarquée - SUPELEC campus de Rennes). Les équipes des deux laboratoires travaillent conjointement sur le projet ANR INFOP et le projet Européen Newcom ++, tous deux s'intéressant à la définition d'architectures multistandard reconfigurables appliquées aux systèmes de traitement du signal et des systèmes de communication numérique.

Plan du Rapport Ce document se décompose en cinq chapitres qui se répartissent comme suit :

Le Premier Chapitre introduit le contexte de notre étude à savoir la Radio Intelligente. La Radio Logicielle et la Radio Logicielle Restreinte seront ensuite comparées et présentées en détails comme un moyen d'appréhender une réalisation pratique des terminaux intelligents avec comme élément constitutif du design la plateforme hétérogène. Celle-ci sera décrite en détails selon les éléments qu'elle met en jeu. Finalement, différentes méthodes existantes de reconfiguration d'une telle plateforme seront introduites afin d'en distinguer forces et faiblesses.

Le Deuxième Chapitre définit la méthode de paramétrisation que nous définissons à savoir "La Technique des Opérateurs Communs". Le chapître 2 se concentrera sur deux points successifs. Tout d'abord, la définition précise et spécifique de ce que nous entendons par "Opérateur Commun" et la technique de paramétrisation développée à partir de ce concept. Ensuite, nous expliciterons la gestion requise par la mise en place de la technique et une solution d'implémentation facilitée par le concept du Banc d'Opérateurs Communs.

Le Troisième Chapitre s'attardera à décrire les différentes fonctions de la chaîne de traitements que nous visons à factoriser. L'opérateur commun s'attachant à factoriser divers opérations de natures différentes tout au long de la chaîne de transmission, une étude minutieuse des opérations requises est indispensable afin de faire ressortir les similarités potentiellement utilisables dans la définition d'un opérateur commun.

Le Quatrième Chapitre spécifiera les deux familles d'opérateurs communs que nous proposons. Nous détaillerons les opérateurs travaillant sur les fonctions de FFT et de décodage Viterbi puis ceux mutualisant les opérations à base de registres à décalages présentes tout au long de la chaîne de traitement. Ce chapitre aboutira à la spécification d'un premier jeu de trois opérateurs communs visant à factoriser le terminal.

Le Cinquième Chapitre constituera la mise en pratique de nos opérateurs par une implémentation sur cible FPGA. Nous implémenterons les OC et mettront en lumière les difficultés de l'intégration d'OC pour un terminal multistandard et le coût d'intégration parfois exorbitant en terme de complexité engendrée. Nous présenterons, ainsi, une solution pour diminuer cette complexité résultante. Deplus, nous intégrerons nos travaux à des recherches connexes à notre thématique qui définissent des algorithmes d'optimisation dans le choix des OC à implémenter. Nous considérerons ces résultats "théoriques" que nous mettrons en balance avec leurs implémentations pratiques.

Chapitre 1

Contexte de l'Etude : La Radio Intelligente

Sommaire

1.1 Introduction : La Radio Intelligente

COGNITIF, IVE Prononciation : kog-ni-tif, ti-v', adj, Terme de philosophie. Qui est relatif à la connaissance. Qui est capable de connaître. Le mot cognitif est un adjectif qualifiant ce qui est relatif :

- A la cognition, c'est-à-dire aux grandes fonctions de l'esprit (perception, langage, mémoire, raisonnement, décision, mouvement...). On parle ainsi des fonctions cognitives supérieures pour désigner les facultés que l'on retrouve chez l'Homme comme le raisonnement logique, le jugement moral ou esthétique...

– Aux sciences cognitives ou qui en adoptent les principaux paradigmes. On parle ainsi de psychologie cognitive ou de thérapie cognitivo-comportementale.

1.1.1 Radio Intelligente ou Radio Cognitive

Nos travaux de recherche ont pour point de départ, la "Cognitive Radio". La raison pour laquelle, nous utilisons l'anglicisme et non les termes francisés de "Radio Cognitive", vient de la différence de signification de vocabulaire entre l'Anglais et le Français. Introduite par J. Mitola en 1999 dans [Mit00], la "Cognitive Radio" est décrite comme une approche qui permet aux objets communiquant de prendre conscience de l'environnement dans lequel ils se trouvent et d'adapter leurs comportements en fonction. Or, les sciences cognitives regroupent un ensemble de domaines beaucoup plus vaste, tels que la Philosophie, la Psychologie, les Neurosciences ou bien encore l'Informatique. Ainsi, au niveau du laboratoire SCEE et par conséquent dans nos recherches, nous ferons référence à la "Cognitive Radio" par le terme "radio intelligente" qui correspond au traitement de l'information, de son acquisition, son interprétation et son utilisation. Ainsi, le terminal Intelligent est défini comme capable non seulement de déterminer le contexte dans lequel il opère mais également à même de modifier son comportement, la méthode d'émission/réception et par conséquent l'architecture nécessaire à celle-ci. Le changement opéré par le terminal voire le va-et-vient entre différentes configurations devra répondre à des contraintes dynamiques et apparaitre comme naturel, transparent pour l'utilisateur. Mitola définit en figure 1.1, le "Cognitif Cycle" en six étapes :

– Observe : Prendre conscience de l'environnement.
– Orient : Orienter le traitement selon divers niveaux de priorité (normal, urgent, immédiat).
– Plan : Planifier les meilleures configurations possibles suivant les priorités précédentes.
– Decide : Allouer les ressources.
– Act : Effectuer la reconfiguration de l'équipement.
– Learn : Apprendre des échecs ou des réussites des précédentes reconfigurations.

1.1.2 La Mise en Pratique de la Radio Intelligente

Illustrons la concept de la radio intelligente par l'exemple du trafic routier. Considérons une voiture qui roule sur une route à double sens. Le conducteur est obligé en permanence non seulement de jauger de la distance entre son véhicule et celui qui le précède (comme celui qui le suit) mais aussi d'anticiper les mouvements de l'ensemble des véhicules et la configuration du terrain. Toute action de ce dit conducteur ne pourra se faire que lorsque la prise de connaissance de son environnement sera effective afin de ne pas créer d'accident. En transposant notre raisonnement au monde des télécommunications. Un terminal, évoluant au seins d'un ensemble d'équipements dans une même cellule, doit satisfaire au mieux les besoins des utilisateurs sans perturber les autres équipements. Ceci représente tout l'enjeu de la radio intelligente : jauger toutes les possibilités d'émission et de réception possible dans un environnement donné en restant

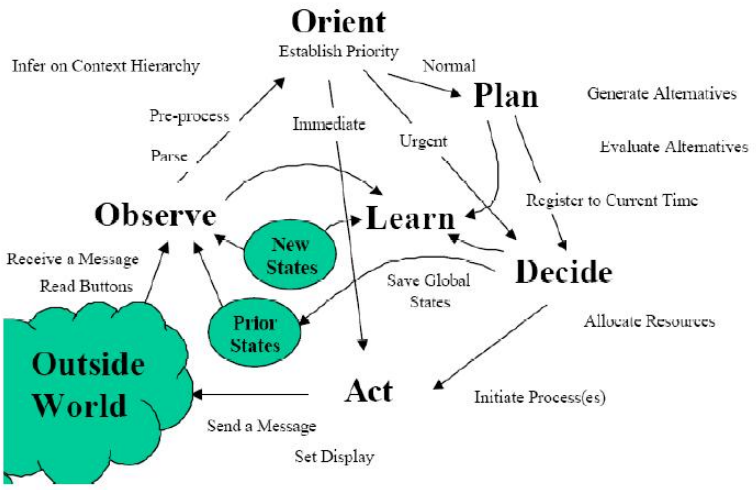

FIGURE 1.1 – Cycle de la Radio Intelligente

transparent pour les terminaux voisins comme pour l'utilisateur. En considérant la multiplication de terminaux intelligents dans un même environnement, la radio intelligente démontrera toute sa pertinence mais aussi toute sa complexité. En effet, si un terminal pris indépendamment étudie son voisinage afin d'agir au mieux selon un critère donné, chaque terminal du voisinage en fera autant et devra analyser et anticiper les actions des autres terminaux. Afin de gérer les interactions entre les utilisateurs, nous pouvons différencier deux approches distinctes, la première où l'intelligence est centralisée dans une seule et même entité et une seconde où chaque terminal est autonome dans la prise de décision.

1.1.2.1 Approche Centralisée

Si nous reprenons l'exemple de la circulation routière précédent. Une approche "réseau" consisterait à centraliser toute l'information concernant l'état actuel du réseau (accidents, bouchons, dégradation de la chaussée, conditions climatiques) et de coordonner les actions des différents véhicules afin de permettre une fluidité du trafic. Dans une certaine mesure, ce procédé existe. Le personnel autoroutier récupère des informations par le biais de personnes physiques, capteurs, caméras et les retransmet aux conducteurs via des panneaux numériques ou des radios dédiées. Une telle approche appliquée au domaine des télécommunications se représentent facilement par la théorie des jeux [OMW+08] qui permet une répartition adéquate des ressources.

Dans le contexte d'un réseau de terminaux intelligents, l'approche orientée réseau permet à chaque équipement radio de partager ses informations avec l'ensemble des autres équipements [NCC05]. Pour ce faire, une entité centrale agrège toute la connaissance du réseau pour ensuite à partir de cette connaissance de l'environnement, répartir

les allocations spectrales aux différents terminaux, chaque terminal communicant un ensemble de paramètres, relevés dans son voisinage, à cette entité de gestion. C'est cette dernière qui régule les paramètres de fonctionnement des différents terminaux suivant les indications acquises. Dans ce cas de figure, le terminal ne possède pas d'intelligence propre. Son rôle consiste à scruter son environnement et à communiquer ces résultats. Ensuite, selon le protocole d'allocations décidé par la centrale de décision, le terminal recevra les paramètres opérationnels qui lui sont affectés et se reconfigurera en fonction. Une approche réseau peut devenir d'autant plus complexe que de nouvelles connexions apparaissent. Dans le cadre du projet Européen E^2R, un canal pilote cognitif (CPC) a été défini. Le CPC est un canal de diffusion qui contient les informations nécessaires pour qu'un mobile puisse s'attacher au réseau sans devoir scanner une large bande de fréquences. Ce canal doit être accessible partout. Il doit donc être diffusé sur une fréquence et une technologie fixes (qui ne subit pas l'allocation dynamique du spectre) et connues à l'avance du mobile. Une solution idéale proposée dans le projet E^2R serait d'avoir une fréquence mondialement harmonisée pour le CPC. La prise en compte de ces nouveaux terminaux n'est pas immédiate et demande donc que l'information soit connue par la centrale de décision et les nouvelles allocations répercutées à l'ensemble des terminaux. Le partage de connaissances et de leurs re-distributions et le contrôle à mettre en oeuvre aboutit à une complexité réelle de cette approche.

1.1.2.2 Approche Décentralisée

L'approche décentralisée délocalise l'intelligence opérationnelle au seins d'une seule et même entité centrale qui pilote chaque terminal. A l'opposé, nous pouvons considérer une approche qui placerait le terminal comme l'entité décisionnelle quant aux opérations qu'il réalise. Ainsi, dans une approche terminal chaque équipement à la possibilité de scruter son environnement et de sélectionner une bande de fréquence sur laquelle il peut transmettre.

Pour ce faire, "l'approche opportuniste", définie dans le cadre des deux approches dé- et centralisées, est la technique couramment utilisée. Celle-ci se base sur la distinction de deux types d'utilisateurs ; (1) Primaire (P) et (2) Secondaire (S). Les utilisateurs primaires caractérisent la répartition "classique" des utilisateurs sur le spectre. Les utilisateurs secondaires regroupent les utilisateurs qui utilisent les même bandes de fréquences que les utilisateurs primaires mais sans interférer, c'est à dire lorsque les bandes sont libres. Une bande libre est définie comme "Une bande de fréquence (qui est) assignée à un utilisateur primaire mais à un instant donné et à un lieu géographique spécifié, la bande n'est pas utilisée par cet utilisateur". Cette technique a été officialisée en [Com03] par la Federal Communications Commission. Nous pouvons reprendre notre exemple concret relatif à la circulation routière. Si un conducteur, circule sur une route à double sens et qu'il désire doubler le véhicule qui le précède, il n'a d'autre choix que d'emprunter la voie venant un sens opposée. Ceci n'est possible que si cette dite voie est libre et si aucun autre véhicule n'arrive en sens inverse.

1.2 La Radio Reconfigurable

1.2.1 Terminaux Reconfigurables

De nous jours, les terminaux sans fils doivent répondre à des besoins croissants en termes d'adaptabilité à différents standards. Ces derniers proviennent non seulement de la prolifération des services mais également de l'incompatibilité entre zones géographiques et évolutions technologiques. En effet, les terminaux doivent pouvoir gérer des standards tel que le GSM, l'UMTS, le GPRS ou EDGE. Outre ces réseaux, un terminal mobile doit pouvoir se connecter à des réseaux sans fils (WiFI, Bluetooth, Wimax), des services de localisation (GPS, Galileo), à la Télévision Numérique Terrestre, DVB (Europe), ATSC (US) ou ISDB (JP). La problématique de cette prolifération est l'incompatibilité de l'ensemble des normes citées précédemment, incompatibilité géographique comme rétro-compatibilité entre nouvelles et anciennes normes dans une même zone. Ce besoin de s'adapter à différents services a été défini sous le nom de "Handover Vertical", par complémentarité avec le "Handover Horizontal" qui qualifie la mobilité géographique d'une cellule à l'autre (figure 1.2).

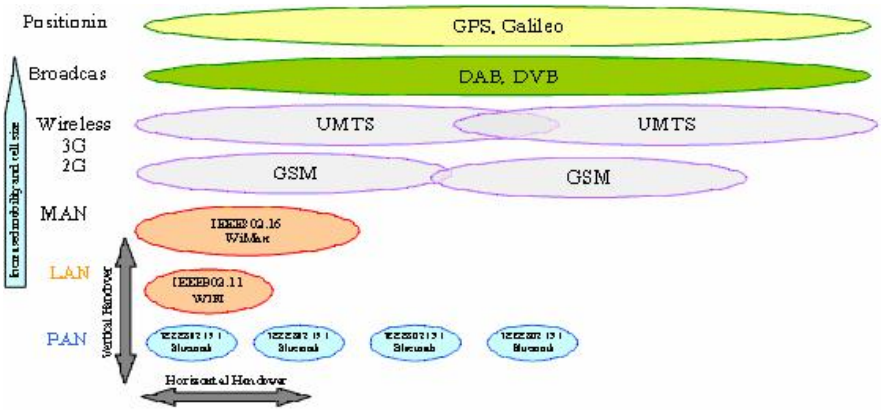

FIGURE 1.2 – Handover Généralisé = Handover Vertical + Handover Horizontal

La conséquence de cette hétérogénéité est le coût croissant en termes de silicium des intégrations successives des anciens et nouveaux standards. La taille des terminaux ne pouvant croitre de manière continue, l'ensemble des standards existants, ajoutés aux incompatibilités de ces équipements dans différents pays, font qu'il n'est plus envisageable de concevoir de manière classique un équipement et qu'il devient nécessaire de pouvoir reconfigurer les terminaux multistandard. En approche classique, un terminal multistandard se défini par la juxtaposition au seins de ce même terminal d'un ensemble de chaînes de traitement, chacun spécifique à un standard donné. Ainsi, un équipement radio ne peut réaliser qu'un petit nombre de standards. L'implémentation est limitée par le coût de chaque circuit dédié (figure 1.3), concaténés dans la même architecture. La méthode Velcro est l'approche dite "classique" [Arm24]. Elle utilise un ensemble de composants spécialisés à un type de traitement, dupliqués pour chaque standard que

doit exécuter le terminal.

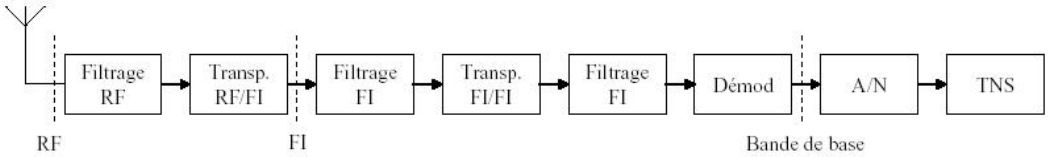

FIGURE 1.3 – Emetteur/Recepteur à bande étroite superhétérodyne

Ces solutions sont dédiées à un terminal précis et établir une liaison à l'aide de l'un ou l'autre des standards se fait rapidement en activant la chaîne correspondante à ce standard. Ainsi, avec une "méthode Velcro", il sera obligatoire de réaliser physiquement une chaîne de transmission différente (figure 1.3) pour chaque standard à intégrer dans le terminal. L'ajout ou la modification d'une norme demandera la reconstruction de l'ensemble du terminal. Cette solution est pertinente pour une gamme spécifiée de standards mais fige le terminal dans le temps et l'espace quant à son évolution technologique.

1.2.2 Radio Logicielle et Radio Logicielle Restreinte

1.2.2.1 Principe de la Radio Logicielle

Les architectures classiques sont figées dès la conception et ne peuvent évoluer dès lors. Pour contourner ce problème, l'idée de radio logicielle est la numérisation du signal juste après l'antenne (figure 1.5), ce qui permet le traitement des opérations (sélection du canal, filtrage, décimation, etc.) en logiciel. Ainsi, il serait possible d'utiliser n'importe quel standard de communication par simple chargement de logiciel. C'est Mitola qui introduit le concept de Radio Logicielle appliquée aux télécommunications en 1991 [Mit00].

Ainsi, la Radio Intelligente ("Cognitive Radio") définie par Mitola exprime le besoin que les équipements radio effectuent en permanence un handover vertical (1.2.2), c'est à dire passer d'un standard de communication à un autre sans interruption dans la communication, afin d'améliorer leur QoS ou la charge du réseau. Ainsi la Radio Logicielle permet de lever la chape de plomb que chaque standard soit implémenté par un circuit dédié. Le terminal radio logicielle dispose d'un circuit générique reconfigurable par du logiciel qui permet de réaliser, par traitement numérique du signal, l'intégralité des traitements de tous les standards. La mise à jour se fait par modification ou ajout de nouveaux logiciels afin de reconfigurer les paramètres opérationnels. Les optimisations entre les différentes couches de protocoles sont facilitées par la possibilité de mutualiser les moyens de traitement de toutes les couches protocolaires, de l'applicatif au physique.

La figure 1.5 donne une schématique de ce que pourrait être une chaine de Radio Logicielle. Un amplificateur large bande LNA amplifie le signal (Low Noise Amplifier). Le

Années 70 : applications militaires (SPEAKeasy) – numérisation de la bande VLF

1992 : J. Mitola lance l'idée pour des applications grand public (Conférence IEEE National Telesystem)

May 1995 : J. Mitola « The software radio architecture », IEEE Communications magazine

May 1997 : 1er Worshop européen à Bruxelles

1999 : - création du forum SDR
- Software radio workshop de Karlsruhe
- nombreux numéros spéciaux IEEE Communications magazine, IEEE Personnal communications magazine, IEEE Selected Areas in Communications

Août 2001 : création du WWRF par les industriels européens avec un groupe de travail SWR

FIGURE 1.4 – Les Principaux Faits Marquants de la Radio Logicielle

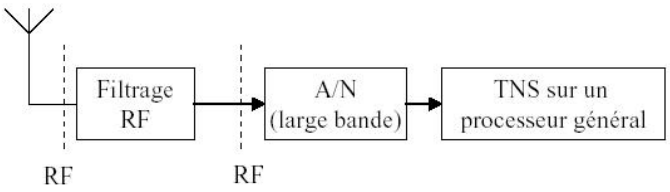

FIGURE 1.5 – Chaine de Traitement dédié à la Radio Logicielle

signal est ensuite filtré afin de rendre disponible sa bande utile qui est échantillonnée par un CAN large bande. En dernier lieu, un Processeur de Traitement Numérique du Signal (PTS) réalise les opérations de traitements du signal (TNS). L'avantage premier d'une telle radio est de traiter tout schéma de modulation sans changer de matériel. Ainsi, sélection de canal, démodulation et filtrage canal peuvent être effectuer en numérique. De plus le TNS permet de faciliter des opérations comme les verrouillages de phases rapide, et d'éliminer le coût et la faible fiabilité des filtres et autres oscillateurs réalisés en analogiques. Néanmoins, les technologies actuelles [LNP+02] ne permettent pas aujourd'hui d'implémenter un tel concept avec une numérisation aussi près de l'antenne que l'on souhaiterait. En effet, les convertisseurs analogique/numérique ne disposent pas d'une dynamique d'entrée suffisante pour matérialiser le concept de radio logicielle. De plus, en [GSS+02], l'auteur démontre que la puissance de traitement numérique nécessaire pour effectuer les traitements en bande de base, ainsi que la consommation sont un réel verrou technologique pour les circuits généralistes actuels. Cette étude estime des performances nécessaires de 10 GIPS pour le GSM jusqu'à 100 GIPS pour l'UMTS. Ces verrous technologiques peuvent être levés par l'adoption d'une architecture dite pragmatique [GSS+02]. Ainsi, avant une évolution future des technologies vers cette radio logicielle idéale, une étape doit être franchie avec l'adoption d'une radio logicielle dite restreinte (SDR, Software Defined Radio), figure 1.6. La chaine de traitement de la radio logicielle restreinte se compose d'une partie analogique qui vise à être totalement éliminée et d'une partie numérique. La numérisation se rapproche donc de plus en plus de l'antenne afin de converger vers la RL idéale. Ainsi, les circuits spécialisés (figés à la conception) sont remplacés par des circuits programmables permettant l'exécution d'applications logicielles.

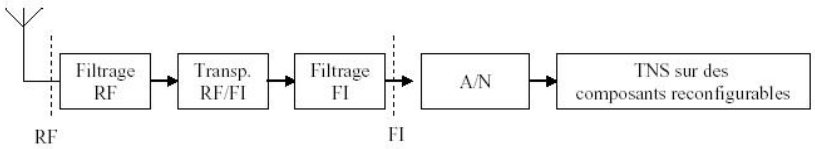

FIGURE 1.6 – Chaine de Traitement dédié à la Radio Restreinte

Par la suite en 1999, avec la définition de la Radio Logicielle Restreinte (RLR, SDR - Software Defined Radio), le SDR Forum sera créé dans le but de valoriser et de permettre l'utilisation de la radio logicielle pour les systèmes sans fils (figure 1.4).

1.2.2.2 Les Projets de La Radio Logicielle

Speakeasy Le projet Speakeasy (Phase I et II) a été le premier projet d'application du concept de radio logicielle. Le projet se déroule de 1992 à 1995 avec l'objectif initial de produire un équipement radio pour l'armée américaine capable de travailler dans une

bande de 2 MHz à 2 GHz. L'idée initiale est que cette radio puisse traiter une dizaine de formes d'ondes (VHF, FM, AM, HF, QAM, ...) afin de palier aux incompatibilités entre les standards des Marines, de l'Air Force et de la Navy.

Le projet aboutit à un démonstrateur lors du 21ième "Transformation of the US Army Exercise" en 1994 à un système multi bandes allant de 4 à 400 MHz. Ce dernier permettait de faire dialoguer entre elles diverses entités opérationnelles, contrôlées via un bus PCI. La Radio produite a été la première à utiliser la technologie FPGA afin d'identifier des interfaces spécifiques à différents modules du terminal ; un contrôleur RF pour contrôler la partie analogique de la radio, un "modem control" pour contrôler le type de modulation (FM,AM,SSB, QAM,...), un module de "waveform processing" pour moduler le signal,... Le résultat n'était pas pour autant portable car codé dans un code assembleur dédié, il était spécifique au matériel utilisé.

Spectrum Ware Le "SpectrumWare Project" est un projet de l'agence pour les projets de recherche avancée de défense de l'armée américaine (DARPA) sous la tutelle du MIT. Contrairement au projet SPEAKEASY qui avait utilisé la meilleure technologie du moment pour définir sa plateforme radio logicielle, l'objectif du MIT est de développer une plateforme qui s'adaptera aux évolutions technologiques et qui tirera bénéfice de la loi de Moore. La figure 1.7 illustre le principe du projet. L'architecture retenue était composée d'un front-end mettant en oeuvre la partie RF et la conversion analogique/numérique et numérique/analogique. Le système d'entrée sortie était dirigé par un bus PCI DMA (GuPPI) et le traitement était implémenté sur un PC/Linux. De manière à assurer la réutilisation de la plateforme et la portabilité logicielle, une approche logicielle orientée objet a été développée. Le premier système reçu des appels téléphonique AMPS, numérisa une des 12,5 MHz de bande et en logiciel, sélectionna un canal de 30 kHz pour réaliser tous les traitements de données sur un PC standard.

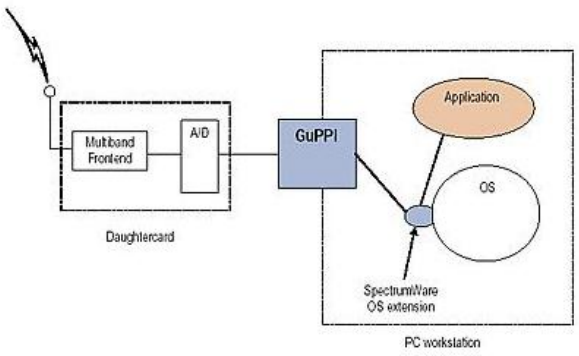

FIGURE 1.7 – Récepteur du SpectrumWare

Joint Tactical Radio System et Software Communication Architecture le Joint Tactical Radio System (JTRS) du département de la défense Américain définit un ensemble d'architectures logicielles baptisé Software Communication Architecture (SCA) basé entre autres sur les spécifications CORBA [Pop98] de l'Object Management Group (OMG) et les standards POSIX. Ce framework ouvert et orienté composants couvre aussi bien les aspects logiciels que matériels ; il s'inspire du modèle OSI de l'ITU pour l'organisation des formes d'ondes. Le SCA défini les principales interfaces logicielles que doivent supporter les applications (" resources ") et les éléments matériels (" devices ") d'une plate-forme de référence dédiée spécifiquement à la Radio Logicielle. A contrario il possède aussi des lacunes : la définition des " ressources " agrège aussi bien la partie fonctionnelle métier que la partie non fonctionnelle (utilisation de CORBA, de POSIX, implantation des ports) et la notion de ports interconnectant l'ensemble des " resources " et des " devices " est insuffisamment formalisée ce qui ne permet pas de garantir la réutilisabilité du logiciel dans des équipements de fournisseurs différents. Enfin la gestion des aspects temps réel et plus globalement de la Qualité de Service est laissée totalement à la charge des fournisseurs de formes d'ondes. Parallèlement au SCA, l'OMG propose la spécification Lightweight CORBA Component Model (Lightweight CCM), une version simplifiée du modèle de Composants Logiciels CORBA (CCM) mieux adaptée aux équipements embarqués. Ses principales qualités sont la simplification des applications et leur portabilité introduites par une séparation explicite et formalisée de la partie fonctionnelle (composants) et de la partie non fonctionnelle (conteneur et services) ainsi qu'une définition précise de types d'interactions entre composants (connecteurs) et de la structure (modèles) des composants. Le déploiement des composants est également spécifié de manière très proche de celui du SCA puisque l'ayant largement inspiré. Cependant malgré ces qualités, Lightweight CCM se positionne plus en tant que socle générique d'élaboration de composants que comme une solution globale à un domaine quel qu'il soit.

1.2.3 La Plateforme Radio Logicielle Restreinte

1.2.3.1 Les Eléments de la Plateforme

La programmation logicielle permet de faire abstraction des spécificités matérielles par l'utilisation d'un langage de programmation de haut niveau (type C/C ++) et d'un compilateur dédié à l'architecture d'exécution. Elle permet aussi la modification d'une application s'exécutant sur une cible technologique par simple changement de code applicatif. Ce type de flexibilité est supporté par des architectures telles que les GPP, les DSP. Pour répondre au besoin d'intégrer des algorithmiques de complexité toujours plus importante, la tendance actuelle est à la conception multiprocesseur. De plus, en adoptant une structure hétérogène, les systèmes multiprocesseur peuvent traiter les tâches plus efficacement aussi bien en termes de surface que d'énergie, deux notions primordiales des systèmes embarqués. De nature hétérogène, ils se composent de multiples éléments que l'on peut classer dans l'une des quatre catégories suivantes : unité de traitement, unité de mémorisation, unité d'interconnexion ou unité d'entrée-sortie.

En outre, chacun de ces éléments peut être doté d'une certaine flexibilité pour répondre aux contraintes de performance.

Dans la suite de ce document, nous désignerons par architecture d'une unité de traitement un ensemble de ressources caractérisé par un agencement matériel et par un ordonnancement de l'exécution des tâches. Ainsi les variations de ressources, d'agencements ou d'ordonnancements autorisent un très grand nombre d'architectures capable de traiter l'ensemble des chemins de données caractérisant les tâches (chaque tâche peut être représentée par un graphe en flots de données spécifiant les dépendances entre les données). Dans cet immense espace de conception, le concepteur doit trouver une architecture satisfaisant aux mieux les différentes contraintes de conception : temps de conception, flexibilité, complexité, performance, consommation... il convient de poser des définitions claires sur le concept de flexibilité. Celle-ci représente la capacité d'adaptation d'une architecture à des tâches différentes. La flexibilité d'une architecture, qui pourrait se mesurer à la variété de tâches supportées, n'est en pratique pas mesurée, car elle est souvent posée comme une contrainte de conception. Par ailleurs, l'imprécision du terme flexibilité oblige à définir des concepts plus en rapport avec les architectures d'unités de traitement. On dira alors qu'une architecture est :

- programmable, si l'architecture sélectionne les chemins de données à chaque cycle d'horloge par l'intermédiaire d'une instruction. Dans ce cas, le concepteur contrôle l'unité de traitement, également nommé processeur, par l'intermédiaire d'un langage de programmation, qui est transcrit ensuite en langage machine, i.e. les instructions du processeur, par l'intermédiaire d'un compilateur.
- reconfigurable, si l'architecture a la capacité de changer la structure des chemins de données (c'est-à-dire les opérations qui y sont réalisées). Le concepteur peut alors contrôler l'architecture grâce à un jeu de configurations. Contrairement à une architecture programmable, le contrôle ne s'exerce pas à chaque cycle d'horloge.
- adaptative, si l'architecture à la capacité de sélectionner des chemins de données parmi un ensemble de chemin de données. Le contrôle est dans ce cas réalisé grâce à un jeu de paramètres dédié à l'application.

Outre un changement de cible technologique, le seul moyen d'augmenter les performances d'une architecture est de la modifier en prenant en compte le parallélisme inhérent aux tâches qu'elle doit exécuter. Le parallélisme désigne le fait de pouvoir traiter à un instant donné lors de l'exécution de la tâche plusieurs chemins de données en parallèle. Nous pouvons distinguer deux types de parallélisme existants : le parallélisme spatial et le parallélisme temporel. Le premier consiste à exécuter en parallèle plusieurs chemins de données indépendants. (Deux chemins de données sont dits dépendants lorsque l'un de ces chemins produit directement ou indirectement une donnée utilisée par l'autre chemin). Le second, ou pipeline, consiste à exécuter en parallèle plusieurs sections (ou étages) d'un même chemin de données. Ce parallélisme, qui permet, certes, d'accélérer le débit sur le chemin, présente quelques limitations. L'ajout de registres sur le chemin impose un léger surcoût au niveau de la latence. Par ailleurs, la

structure pipeline est très sensible aux déséquilibres entre les différents étages. L'horloge du système est partagée par tous les étages, le débit du chemin est dépendant de l'étage ayant le temps de traversée le plus long. Outre la performance obtenue par parallélisme, le choix d'une architecture nécessite de prendre en compte bien d'autres contraintes. Or ces contraintes peuvent interagir entre elles de manière conflictuelle. On sait par exemple que chercher à rendre flexible une architecture se fait toujours au détriment de sa performance et vice-versa. Cela s'explique parce qu'une architecture flexible sous-utilise ses chemins de données et nécessite une structure de contrôle plus importante pour sélectionner et ordonnancer les ressources. Pour les mêmes raisons, une architecture flexible est également plus consommatrice d'énergie.

La figure 1.8 positionne les principales catégories d'architecture que nous détaillons dans la suite de cette section selon les critères de conception.

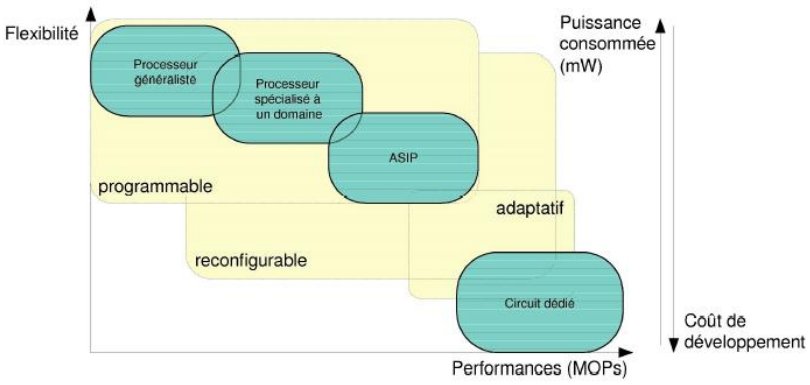

FIGURE 1.8 – Fléxibilité et différents éléments de la Radio Logicielle Restreinte

Processeur Généraliste Les processeurs à usage universel ou processeurs généralistes sont des architectures programmables dont le jeu d'instructions permet de traiter n'importe quel algorithme grâce à un support à la fois de l'algorithmie flottante et de l'algorithmie entière. Parmi les architectures les plus connues, on peut citer les architectures ARM dominant le marché des applications embarquées, les architectures MIPS reconnues pour leur efficacités (en surface et en consommation) pour les calculs scientifiques, et les architectures x86 monopolisant le marché de l'équipement informatique grand public. Pour améliorer leur performance, les processeurs généralistes reposent sur une architecture pipeline.

Processeur Dédié à un Domaine L'objectif des processeurs dédiés à un domaine est de fournir pour un domaine d'application des performances supérieures aux GPP tout en conservant la flexibilité nécessaire pour ce domaine d'application. Dans la pratique, le jeu d'instruction est allégé des instructions superflues ou trop générales et est complété par des instructions dédiées pour accélérer les traitements. Il existe pléthore

de processeurs dédiés à un domaine : des processeurs graphiques (GPU) pour gérer le rendu graphique, des processeurs physiques (PPU) pour prendre en charge les calculs physiques (dynamique des fluides, des corps rigides, ...) principalement utilisé dans les jeux vidéos, des processeurs audios pour prendre en charge le traitement audio, des processeurs multimédias pour applications embarquées comme le Trimédia de NXP Semiconductors [SRD96], ou encore des processeurs de traitement du signal (DSP) dédié au calcul numérique [Man01] [Ins98]. Pour prendre l'exemple des DSP, ces processeurs ne disposent dans leur grande majorité que de l'arithmétique flottante (jeu d'instruction allégé) et sont complétés par des instructions spécialisées comme l'instruction MAC (Multiplication Accumulation).

Processeur Dédié à une Application Pour faire face à la diversité des algorithmes encore présents dans un domaine d'application, un processeur dédié à un domaine ne peut pas exploiter un degré de parallélisme optimal pour chaque application, ce qui se traduit dans les performances. Ainsi pour encore améliorer les performances tout en conservant la programmabilité, les processeurs dédiés à une application disposent de chemins de données exploitant au mieux le parallélisme de l'application tout en conservant un minimum d'opérations élémentaires pour permettre un minimum de flexibilité. Les processeurs dédiés à une application sont le plus souvent désignés par l'appellation ASIP (Application Specific Instruction-set Processor) [VPGLDM94], qui fait référence au jeu d'instruction dédié du processeur. Mais la notion de processeur dédié à une application n'est pas limitée aux ASIPs. Par exemple, de récents travaux [VPGLDM94] présentent un processeur sans jeu d'instruction NISC (No Instruction Set Computer) pouvant exploiter à la fois le parallélisme temporel et spatial d'une application. Il est intéressant de noter que la compilation d'un code sur la structure NISC est assimilable à une synthèse architecturale. Cette structure pousse le concept de processeur dédié à une application à la limite entre la programmabilité et l'adaptativité, puisque les "instructions" correspondent au jeu de paramètres adaptatifs de l'architecture.

Circuit Dédié Les circuits dédiés, couramment appelés ASIC (Application Specific Integrated Circuit) lorsque la cible de conception est un masque de fonderie, désignent un ensemble d'architectures se limitant à l'exécution exclusive d'une application. Comme la conception de ce genre de circuit n'est affectée d'aucune contrainte architecturale pour exploiter le parallélisme, ces architectures permettent d'exprimer les performances optimales d'un algorithme. En revanche, à cause d'un temps de conception conséquent, la flexibilité limitée de ce genre d'architectures est principalement cantonnée à l'expression d'une adaptativité sur un jeu de paramètres réduits ou à l'exploitation de cibles de conception différente.

Architecture Reconfigurable Le domaine des architectures reconfigurables est vaste puisqu'il permet de manière transversale au choix de l'architecture d'ajouter de la flexibilité à une architecture. Grâce aux mécanismes de reconfiguration comme des FPGAs embarqués, on peut par exemple reconfigurer des chemins de données dans un pro-

cesseur [LTC+03] [SLL+00] ou une structure hétérogène [DCPS02] ou reconfigurer un co-processeur traitant une tâche complète [BEM+03] [HW97]. Ainsi on distingue généralement différents niveaux de granularité dans la reconfiguration, ce qui explique pourquoi la reconfigurabilité s'applique aussi bien à des opérateurs dans un processeur (grain fin) qu'à des co-processeurs complets dans une architecture (gros gain). Les architectures reconfigurables peuvent alors améliorer les performances d'une architecture en autorisant par reconfiguration l'exploitation d'une ressource non exploitée dans une architecture non reconfigurable. Ce gain de performance suppose au préalable une perte de performance initiale due à la cible reconfigurable et pose également de nombreux problèmes quant à la gestion des configurations. Premièrement, l'architecture doit assurer un délai de reconfiguration faible devant le délai de calcul pour ne nuire ni aux performances ni à la consommation. Il faut également ensuite tenir compte du stockage des reconfigurations et du surcoût engendré en termes de surface pour l'architecture. Nous détaillons la technologie FPGA en Annexe A.

1.2.3.2 La Plateforme Hybride

Un terminal Radio Logiciel doit dans le domaine des télécommunications satisfaire non seulement aux contraintes des standards de radiocommunications mais aussi à celles inhérentes à un terminal embarqué. Pour arriver à définir un tel terminal, les architectures mises en oeuvres utilisent des types différents d'éléments. Nous pouvons distinguer les processeurs généraux (GPP), les processeurs de traitement numérique (DSP), les circuits spécialisés (ASIC, ASIP, Hardware accelerator), ou les circuits intégrés programmables (FPGA).

Chacun de ces composants permettra de répartir la charge de travail entre application logicielle et matérielle et entre traitements dédiés ou génériques. Ainsi, la chaîne de transmission n'est plus constituée de composants dédiés mais se compose de traitements numériques exécutés sur des "calculateurs" correspondant aux opérations de traitement du signal requises. Selon la complexité des opérations de TNS, un des types d'éléments cités précédemment sera mis à contribution. La Plateforme Radio Logicielle devient par conséquent une plateforme hybride où le logiciel est couplé à des accélérateurs hardwares. La modularité ainsi obtenue permet une réutilisation maximale des parties logicielles et matérielles de configuration en configuration.

La Plateforme RL Vis à vis de la technologie actuelle, la radio logicielle idéale ne peut se faire actuellement et seule des solutions de radio logicielle restreinte peuvent être envisagées. Une approche toute logicielle est elle même à proscrire du fait de la consommation d'énergie prohibitive des processeurs généralistes. La Radio Logicielle s'oriente donc vers la conception d'équipements associant à la fois une flexibilité logicielle et une flexibilité matérielle.

La figure 1.9 illustre une architecture Radio Logicielle Type composée de trois étages de traitement.

1. La première partie de la plateforme consiste en l'acquisition du signal en analogique et sélectionne la bande utile du signal.

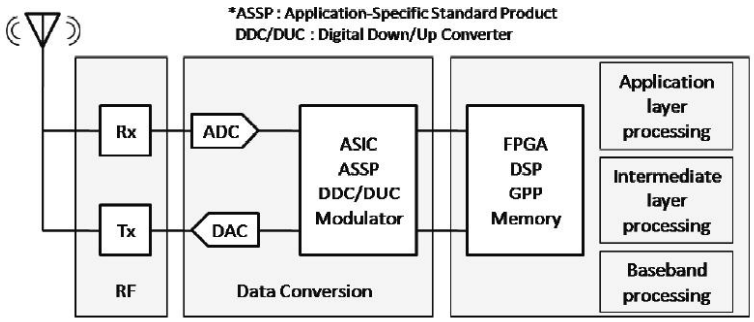

FIGURE 1.9 – Plateforme Radio Logicielle

2. Ensuite, un échantillonnage en fréquence intermédiaire (FI) sera réalisé dans la partie "Data Conversion". L'étage FI a pour but d'améliorer la sélectivité et la sensibilité du récepteur.
3. La troisième partie est le traitement en bande de base. Réalisé grâce à des circuits numériques, il est en grande partie reconfigurable et permet de s'adapter aux différents traitements (synchronisation, estimation de canal, décodage canal,...).

Ainsi, de part la complexité et l'hétérogénéité d'un telle plateforme, les procédés permettant d'effectuer les reconfigurations seront compliqués en mettre en place. Nous devrons cibler les ressources devant être reconfigurées, le chemin des données de reconfiguration, prendre en compte les différences de temps de traitement entre chaque circuit utilisé, ainsi que le remps de reconfiguration en s'assurant de ne pas perturber les traitements en cours.

FPGA, support de La Plateforme Hybride A mesure que le catalogue d'IP disponibles continue de croître, les FPGA ressemblent de plus en plus à des systèmes sur puce. En effet, des microprocesseur 32 bits embarqués constituent des IP possibles pour les FPGA. La création de systèmes embarqués spécifiquement conçus pour une application, avec des FPGA comme technologie de base, gagne actuellement du terrain. Un rapport de la société américaine Gartner Inc. indique que, d'ici à 2010, plus de 40 % des FPGA intégreront un microprocesseur embarqué. Du point de vue de la vitesse de traitement, avec plus de 200 MHz pour les implémentations sur processeurs softcores et plus de 400 MHz pour les implémentations sur processeurs hardcores, près de 80 % des applications embarquées nécessitant 32 bits peuvent bénéficier d'une solution FPGA. De plus, intégrer le microprocesseur au FPGA n'implique pas de devoir faire des compromis sur les performances. Par exemple, Xilinx propose un microprocesseur softcore 32 bits appelé MicroBlaze (figure 1.10), avec taille d'instructions et de cache configurables, qui inclut une unité de gestion de mémoire (MMU, Memory Management Unit) optionnelle pour la protection des accès à la mémoire. Ce softcore peut être intégré à n'importe lequel des FPGA de Xilinx ; ce dernier commercialise par ailleurs

un processeur hardcore PowerPC 32 bits pour sa gamme Virtex de plus haut niveau. Grâce à l'interface bus processeur local (PLB) avec standard ouvert intégré sur ces deux processeurs, ils peuvent bénéficier d'une logique d'accélération ainsi que la connexion à de nombreux périphériques (figure 1.10). Il n'y a aucune implémentation d'architecture fixe et donc aucune obligation que les fonctions soient réalisées par matériel plutôt que par logiciel. Il existe donc un vaste choix de solutions envisageables pour chaque application, allant des plus générales aux plus spécifiques.

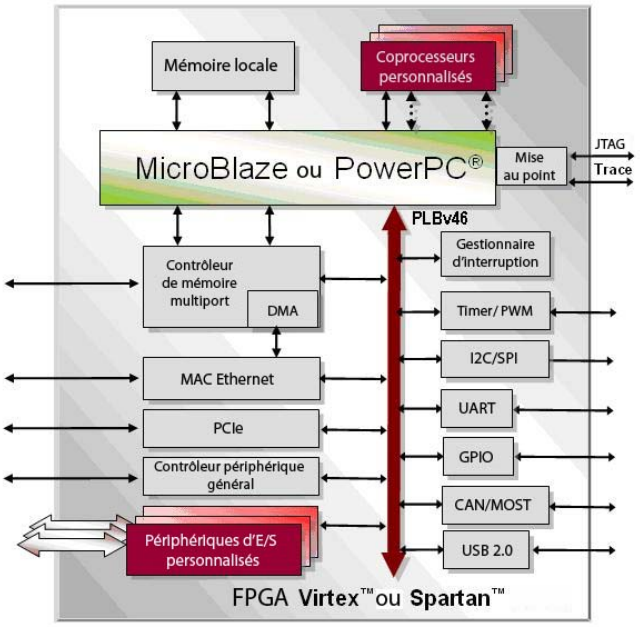

FIGURE 1.10 – Instanciation Soft ou Hard via les connections du microblaze du FPGA

Les FPGAs ont été conçus au départ pour se substituer aux CPLDs comme "logique de colle" pour les PCBs. Toutefois, due à l'évolution des technologies de synthèse de silicium, la taille et la complexité des FPGAs ont augmenté, les menant au rôle de compétiteurs sur le marché des systèmes sur puce. Aujourd'hui le domaine d'applications des architectures à petit grain s'étend aux algorithmes pour l'industrie spatiale, imagerie médicale...

En plus des sytèmes de LUTs (petit grain), les FPGA peuvent arborer toutes les granularités possibles :

- grain moyen : Multiplieurs à X-bit, additioneur intégré, et un accumulateur à K-bit.
- gros grain : DSPs, processeurs PowerPC.

Dans la réalisation d'une plateforme hybride pour la Radio Logicielle, les FPGA sont les architectures qui présentent l'ensemble des niveaux de granularité. Possédant à la fois, des unités matérielles mais aussi logicielles, ils permettent d'avoir accès aux deux

types de flexibilités et de migrer des applications du SOFT ou HARD ou inversement si besoin.

Reconfiguration Partielle de la Plateforme Ainsi, afin de satisfaire le Handover Vertical, il est nécessaire d'utiliser une reconfiguration dite dynamique. La reconfiguration dynamique, à l'inverse d'une reconfiguration statique, implique une reconfiguration sans interruption de service. En [LDP05], Delaye identifie deux types de reconfiguration dynamique, avec ou sans dépendance de données. Cette dépendance des données aura pour conséquence, la sauvegarde de ces dites données entre deux configurations. L'autre aspect critique devient donc le temps de reconfiguration. Ainsi, lorsque l'application le permet, il reste possible de faire appel à une reconfiguration totale du circuit. Par contre si les contraintes de temps sont très strictes, il est judicieux de diminuer le temps de reconfiguration du matériel et donc réduire la taille du fichier de reconfiguration. Ceci peut être fait grâce à l'utilisation de circuits permettant la reconfiguration d'une partie de ce circuit et non de sa totalité. Ainsi en [LDP05], Delaye s'appuit sur l'analyse des traitements des couches physiques dans les chaînes d'émission des standards UMTS, GSM, 802.11g afin de proposer une analyse de factorisation des traitements multistandard et de réduire le nombre de contextes à gérer. Il définit ainsi une approche hiérarchique et distribuée de la gestion de configuration ("HDCM") afin de répondre aux besoins de flexibilité des applications orientées flots de données implantées sur plate-forme hétérogène. Delaye apporte dans ces travaux différentes méthodologies de conception de systèmes sur puce dynamiquement et partiellement reconfigurables sur FPGA.

La technologie utilisée dans certains FPGA permet la reconfiguration d'une partie du composant sans perturber le fonctionnement de l'ensemble ([LDP05]). Ainsi, si plusieurs opérateurs sont instanciés sur un même FPGA, il est possible de venir reconfigurer l'un d'eux sans perturber le fonctionnement des autres. La conception d'un système permettant ce type de reconfiguration se fait par séparation en deux parties du système à instancier sur le FPGA. L'une appelée partie statique qui représente la partie non modifiée par un changement de contexte et l'autre est appelée partie reconfigurable. Le but étant de réduire au maximum la partie reconfigurable comme présenté dans ([Had95]). Un bitstream de reconfiguration est généré, ne contenant que les informations relatives à la zone reconfigurable. Il est possible d'aller encore plus loin que cette technique appelée reconfiguration partielle en générant un bitstream ne contenant que les changements réels entre deux contextes d'exécution. Cette technique est appelée reconfiguration partielle par différence. Ce qui fait une économie non négligeable en termes de poids de fichier ce qui permet à la fois une occupation mémoire réduite et un temps de reconfiguration réduit.

1.2.4 Une méthodologie de conception de la Plateforme Hybride : La Paramétrisation

Les Techniques de Paramétrisation dans la Radio Logicielle Dans un premier temps, nous devons préciser que le terme paramétrisation n'existe pas dans la langue

française et n'est pas reconnu par les différents dictionnaires. En réalité, le terme paramétrisation est dérivé de l'anglais "Parameterization" qui qualifie l'ensemble des paramètres permettant de définir entièrement un objet. De même, le terme "technique de paramétrisation" est avant tout un concept mathématique avec comme objectif d'identifier les degrés de liberté d'un système. De plus dans le domaine des télécommunications et de la Radio Logicielle, diverses approches de la Paramétrisation sont considérées selon les différents auteurs de [Mit00], [Tut02] et [Jon02]. Ainsi, dans nos travaux de recherches, nous considérons la paramétrisation comme une technique qui permet d'identifier parmi un ensemble d'entités (standards, fonctions de télécommunications, opérateurs mathématiques), des entités qualifiées de communes et qui pourront se substituer aux entités initiales par simple "téléchargement" de paramètres. Ainsi, une technique de paramétrisation aura pour objectif de définir de telles entités et les paramètres associés. Appliqué dans le cadre de la Radio Logicielle Restreinte, la paramétrisation prend tout son sens. En effet, un terminal multistandard devra jongler entre différents standards et reconfigurer en temps réel la Plateforme Hybride. La Paramétrisation cherche à définir des "communalités" entre les différents standards indépendamment de la cible sélectionnée. Ainsi, dans le cadre de la Radio Logicielle, la paramétrisation pourra définir des éléments communs, implémentables indépendamment sur DSP, μP ou encore FPGA. De plus la paramétrisation ne préjuge pas de la méthode utilisée pour la reconfiguration utilisée sur ces dites cibles technologiques pour passer d'un élément commun à l'autre. Notre vision de la Paramétrisation est une technique qui définit des blocs qui resteront figés dans leurs designs et qui s'adapteront à une fonctionnalité juste en modifiant un jeu de paramètres. La Paramétrisation permet donc de diminuer qualitativement le nombre d'éléments à implémenter et peut constituer une méthode de reconfiguration en elle-même ou une technique complémentaire aux méthodes existantes telle que la Reconfiguration Partielle d'un FPGA.

Prenons un exemple et supposons qu'une technique de paramétrisation définisse une entité commune (COM1) entre deux standards S_1 et S_2. Supposons toujours que le paramètre α appliqué à COM1 permette d'exécuter une opération de S_1 et que le paramètre β appliqué à COM1 permette d'exécuter S_2. Alors, passer de S_1 à S_2 ne nécessitera plus de reconfigurer le terminal, de charger un code ou d'envoyer un bitstream mais juste de fournir en plus des données le paramètre α ou β selon le cas. Maintenant supposons l'existence d'une deuxième entité commune COM2, suivant le même principe. En combinant Paramétrisation et reconfiguration partielle, nous pouvons sur la même sous partie du FPGA, facilité la reconfiguration partielle en autorisant à reconfigurer seulement deux fois (soit COM1, soit COM2) au lieu de passer par quatre reconfigurations partielles. Ainsi, dans le cadre de la radio logicielle, les techniques de paramétrisation vont permettre de définir des entités Software ou Hardware, communes à chaque standard. En implémentant ces entités, nous définissons donc un niveau de granularité supérieur à celui existant pour proposer de nouveaux IPs qui seront communs aux standards de télécommunication. Or à la différence, d'un MAC ou d'une bascule, qui opère systématiquement la même opération et sur une cible donnée, les entités communes seront modelables dans leurs fonctionnement selon le ou les paramètres associés.

Ainsi, dans notre contexte de Radio Intelligente et de Handover Généralisé, nous envisageons la Paramétrisation comme la technique qui permet de définir ces nouveaux composants paramétrables.

Une Technique de Paramétrisation : La Fonction Commune La première technique de paramétrisation clairement identifiée en [Rhi02] a été la Technique des Fonctions Communes. Cette technique se comprend intuitivement vis à vis d'un standard de télécommunication. Une chaine d'émission/réception d'un standard de télécommunication comprend systématiquement les mêmes étapes de traitements que ce soit pour la définition du signal émis comme pour la restauration des données.

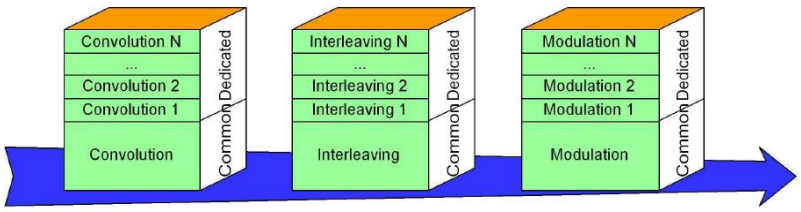

FIGURE 1.11 – Exemple de Fonctions Communes le long d'une chaine de traitements

La particularité de la technique des Fonctions Communes est de se focaliser sur les différents aspects fonctionnels requis par les standards à implémenter dans le terminal (figure 1.11). Les Fonctions Communes ainsi créées, sont des macro blocs qui reprennent des fonctionnalités identiques à chaque standard en les regroupant en une seule et même entité. Les Fonctions communes sont donc des entités paramétrables et permettent d'obtenir un design reconfigurable par macro blocs fonctionnels. Le terminal multi standard devient donc paramétrisable par bloc. Nous pouvons prendre comme exemple de fonctions communes présentes dans la litérature celle de Jondral et de Rhiemeier. La premier exemple concerne le "Generalized parameterizable modulator" présenté par Jondral en [Jon02]. Il s'agit d'une générateur designé pour les modulations GMSK (Gaussian Minimum Shift Keying), QPSK (Quadrature Phase-Shift Keying), $\frac{\pi}{4}$ QPSK and dual QPSK des standards GSM, IS 136 et UTRA FDD15. Une étude détaillée de ces fonctions communes révèlent qu'elles consistent à l'accumulation des différentes modulations regroupées dans une même fonction. La paramétrisation se limite ainsi à un simple choix entre l'une ou l'autre, en sélection le chemin de données associé. Le second exemple est le codeur canal de Rhiemeier dédié à l'UMTS, le GSM et le PMR17. A l'instar du premier exemple, cette fonction commune de codage est plus un ensemblage de différents codeurs et d'un choix vis à vis duquel sélectionné. Ainsi, cette première méthode permet de réaliser de dire que la "Paramétrisation permet de construire des système de télécommunications à partir d'éléments flexible sous la condition que ces éléments appartiennent à un set prédéfini de standard".

1.3 Conclusion

Nous venons de présenter le besoin en terminaux reconfigurables pour la radio intelligente qui se définit initialement comme une application de la Radio Logicielle. Or celle-ci reste encore un objectif idéal à atteindre de part la limitation des équipements actuels à transposer le signal en bande de base. De ce fait, la mise en pratique de la Radio Logicielle se limite à la Radio Logicielle Restreinte. En termes de reconfiguration du terminal multistandard, celui-ci ne peut se réaliser uniquement en Software et demande l'utilisation d'accélérateurs Hardwares. Or ces derniers sont difficilement reconfigurables en temps réel et consommateurs d'énergie. Dans ce contexte, les techniques de Paramétrisation existantes répondent partiellement à la problématique. Elles proposent une vision générale de la plateforme hybride et définissent des éléments qui peuvent être "reconfigurés" en temps réel en ne modifiant qu'une gamme limitée de paramètres. Néanmoins, telles que définies par l'utilisation de Fonctions Communes, elles n'en restent pas moins dépendantes des standards à considérer. Toute la problématique de notre étude est là, arriver à définir une méthode qui permette de proposer un terminal multistandard, supportant l'alternance des standards par une reconfiguration en temps réel et qui soit dans le même temps adaptée à la Radio Intelligente, c'est à dire évolutive vers d'autres normes non prévues à priori.

Chapitre 2

La Technique des Opérateurs Communs

Sommaire

2.1 Introduction

Le chapitre précédent présente les "Besoins de la Radio Intelligente" et le contexte d'étude lié à son développement. La Radio Logicielle reste un concept idéal et ne peut se définir que par le biais d'une version pragmatique que constitue la Radio Logicielle Restreinte. Celle-ci permet permet grâce à une plateforme hybride (DSP, μP et FPGA) d'avoir accès à une flexibilité tant logicielle que matérielle. L'utilisation complémentaire des deux, autorise la définition de terminaux reconfigurables à un niveau Hardware comme Software.

Notre étude porte sur la réalisation d'un terminal multistandard reconfigurable qui permettrait d'alterner l'exécution d'un seul standard à la fois parmi une palette de normes possibles. La reconfigurabilité qui cherche à être la plus transparente possible pour l'utilisateur, est dépendante des contraintes temporelles imposées par ces dites normes et devient donc tributaire des capacités de la plateforme. Dans ce contexte, les techniques de Paramétrisation apportent deux réponses pertinentes. Premièrement, en définissant des éléments reconfigurables par téléchargement de paramètres, le design peut se modifier en temps réel par insertion d'un nouveau jeu de paramètres. Deuxièmement, ces techniques définissent des entités communes indépendamment de la cible, permettant ainsi de pouvoir suivre l'évolution technologique des composants de la plateforme hybride et de combiner des exécutions logicielles et matérielles. Ainsi, nous orientons nos travaux dans la mise en application des techniques de Paramétrisation sur un terminal multistandard. Pour atteindre cet objectif, nous faisons le choix de considérer la plateforme dans son intégralité et de ne plus nous focaliser sur une de ces entités constituantes pour tenter de la reconfigurer plus ou moins partiellement. Nous privilégions la reparamétrisation de la plateforme par simple modification des paramètres en temps réel de manière simultanée avec l'insertion des données. Ces techniques n'altèrent pas le design en lui même, que ce soit du code ou une architecture mais la fonctionnalité de ce design.

Néanmoins, l'idée de trouver des éléments communs entre différents standards ou de mutualiser des traitements n'a rien de nouveau, elle constitue l'idée basique dans le développement d'un équipement multistandard ou de manière général multi-applicatifs. Cependant, là où les techniques de paramétrisation innovent est bien de permettre une reconfiguration en temps réel où seule, une partie des données permet de modifier le fonctionnement du terminal sans devoir modifier la plateforme hybride. Les travaux antérieurs à nos recherches ont définis les éléments communs uniquement par le biais d'une décomposition fonctionnelle de la chaîne de traitement, entrainant une réelle dépendance aux standards considérés. Dans nos travaux, nous définissons des macro blocs génériques, indépendants d'une exécution spécifique et utilisables quelque soit le standard ou la fonction considérés. Ainsi, nous formalisons dans cette thèse le concept de la Technique des Opérateurs Communs.

2.2 Etat de l'Art sur les Opérateurs Communs

2.2.1 Représentation et Optimisation Graphique d'un Terminal Multistandard

L'idée initiale de l'opérateur commun a été définie par Palicot en [PR02]. Pour bien comprendre l'idée de l'opérateur commun, il faut le considérer historiquement comme une possibilité d'optimisation d'un terminal multistandard selon un critère donné (consommation, surface, ...). Ainsi, il n'est plus question dans cette approche de considérer des macro blocs fonctionnels mais de se placer à un niveau de granularité inférieur aux fonctions. Les différentes fonctions de télécommunications sont ainsi "décortiquées" afin d'identifier dans leurs besoins, des entités qui pourraient être réutilisées dans l'exécution d'autres fonctions de télécommunications d'un autre standard ou bien du même standard. Par conséquent, l'opérateur commun ne se focalise pas sur un aspect fonctionnel mais bien sur un aspect opérationnel du besoin global inter et intra standards. En supposant l'existence de ces entités communes appelées opérateurs communs, il devient alors possible de construire un terminal à partir d'un set limité de ces derniers et d'optimiser l'implémentation du terminal. Comme nous l'avons mentionné, l'opérateur commun a été présenté avant tout dans un but d'optimisation du terminal. En [MPRG06], Palicot, Moy et Rodriguez illustrent le concept de l'opérateur commun par une représentation graphique (figure 2.1) sous forme d'arbre et définissent un premier procédé d'optimisation permettant de déterminer le meilleur choix d'éléments communs. L'approche définie en [MPRG06] utilise une hiérarchisation en graphe de la décomposition des différents standards à considérer dans le terminal voulu. Ce graphe 2.1 permet ainsi de faire apparaitre le niveau de granularité entre les entités constitutives. En [MPRG06], Rodriguez spécifie chaque entité de son graphe par deux paramètres :

- Un coût d'implémentation "CC", coût qui doit être " payé " chaque fois que l'opérateur est utilisé.
- Un coût monétaire "MC", coût, qui est " payé " une seule fois lors de l'implémentation physique de l'opérateur.

Ainsi, dans ces définitions Rodriguez fait une hypothèse forte sur la définition de ces entités communes. Dans la mesure où il suppose que l'opérateur considéré ne pourra être implémenté qu'une seule fois quelque soit les fonctions qui l'appellent, qu'elles s'exécutent en parallèle ou à la suite. Dans cette approche, la possibilité d'organiser temporellement ("scheduling") des opérateurs communs est supposée acquise. Ainsi, une seule entité physique de l'opérateur commun est envisagée à partir de laquelle, l'ensemble des opérations peuvent être réalisées. Outre la définition même de chaque entité commune, l'auteur spécifie aussi la relation entre chaque entité de son graphe. Cette relation est le nombre d'appels ("NoC") de l'opérateur appelé vis à vis de l'opérateur appelant qui spécifie combien de fois l'opérateur appelé est nécessaire pour exécuter l'opérateur appelant. Une distinction est faite entre deux types de liaisons de construction différentes, à savoir une liaison OU et une liaison ET. Une fonction de coût et un

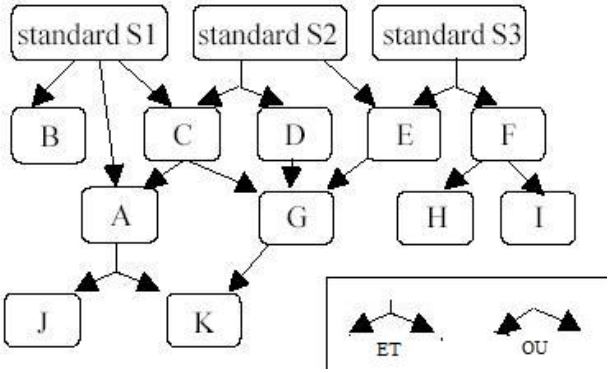

FIGURE 2.1 – Décomposition des Standards en Niveau de Granularité

algorithme de minimisation de celle-ci permet de sélectionner les meilleurs opérateurs à implémenter qui deviendront ainsi des Opérateurs Communs :

$$C_{SDR} = min_{bool((S_{n n\in N}))}(I.\sum_i c_i.S_i + \sum_k I_k.V_k.((S_b)_{n\in N})) \tag{2.1}$$

C'est cette complexité totale que la méthode vise à minimiser pour déterminer quels sont les meilleurs opérateurs pour des standards donnés. Cette équation met en jeu une fonction de coût des opérateurs qui est :

$$I.\sum_i c_i.S_i + \sum_k I_k.V_k.((S_b)_{n\in N}) \tag{2.2}$$

Celle-ci, prend en compte :

- c_i , le coût monétaire du composant, $\sum_i c_i.S_i$ représente le coût total de tous les composants présents dans le système SDR considéré. S_i caractérise si le noeud en question est présent dans le système envisagé.
- $V_k((S_b)_{n\in N})$ représente le coût de " computation " du standard k.
- I et I_k sont respectivement les poids attribués respectivement au coût total monétaire et au cout de computation. I_k , caractérise le taux d'activation d'un standard par rapport à un autre.

Nous pouvons donner un exemple simple de la stratégie suivie par cette procédure et reprendre celui de [MPRG06]. La figure 2.2 illustre une décomposition avec des coûts arbitraires des fonctions d'égalisation et de modulation OFDM. Ici, si nous exécutons l'égalisation par le filtre FIR, nous aurons 1 seul appel, soit un coût d'une implémentation à 500 et d'un appel à 1000 soit un total de 1500. Si nous faisons de même pour la modulation OFDM via la FFT, nous obtenons le même résultat. Les deux fonctions ont par conséquent un coût de 3000. Maintenant, si nous considérons seulement le module

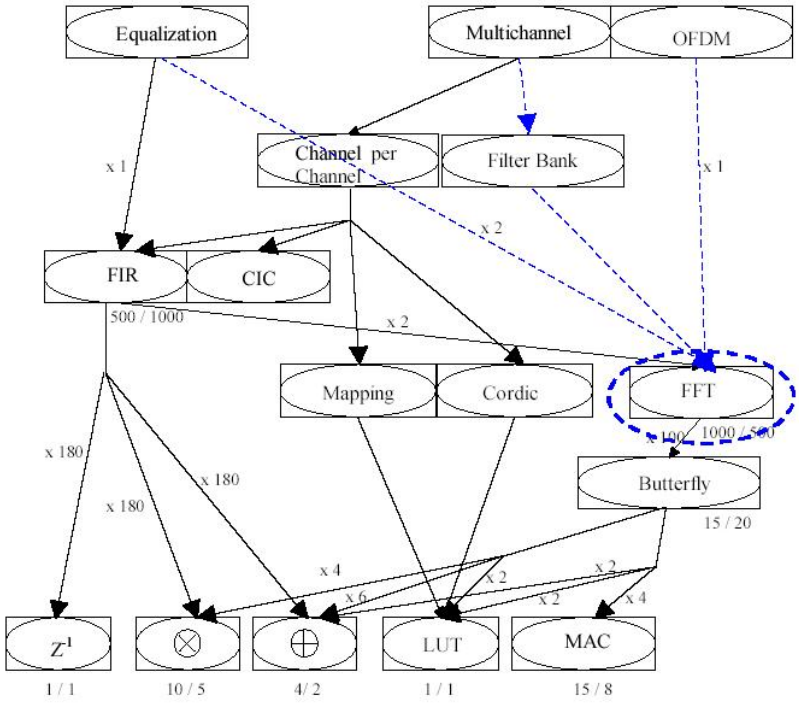

FIGURE 2.2 – Décomposition d'un terminal Multistandard

FFT et non plus la combinaison FFT + FIR, alors il ne sera implémenté qu'une seule fois pour un coût de 1000. Ensuite, viendront s'ajouter les coûts d'appel de 1 x 500 pour l'OFDM et 2 x 500 pour l'égalisation pour un coût total de 2500. Ce procédé a été généralisé en [MPRG06] et par une méthode de recuit simulé, il est possible de déterminer le meilleur set d'opérateurs basé sur les considérations de Rodriguez. Ainsi, on comprend bien par cet exemple toute l'utilité de l'utilisation d'un set limité d'opérateurs communs et pourquoi ces derniers pourraient être définis comme les entités de bases d'une radio logicielle intelligente. Cette première section suppose l'existence de telles entités communes. Nous présentons dans les deux prochaines sections la littérature permettant de définir les premiers opérateurs communs.

2.2.2 La Transformée de Fourier Rapide : Entre Fonction Mathématique et Opérateur

Les Principaux travaux de recherche qui sont à l'origine de la paramétrisation selon des opérateurs communs ont été entrepris par Palicot et Roland en [PR03]. Dans ces travaux les auteurs étudient les différentes possibilités d'utiliser la Transformée de Fourier Rapide (FFT) [CT65] lors de plusieurs étapes d'un chaine de traitement en émission et en réception. Dans ce contexte, la FFT est présentée comme un opérateur mathématique, qui est utilisé, seul ou en complément d'autres opérations mathématiques pour mener à bien des fonctions qui ne sont pas initialement décrites dans la littérature comme des mises en application directe de la FFT.

2.2.2.1 La Transformée de Fourier Rapide : Définition

Nous définissons tout le protocole d'implémentation et de fonctionnement de la FFT en 3.3.2. Nous présentons ici, les fondamentaux relatifs à la FFT afin de comprendre les différentes utilisations qui peuvent en être faites. La FFT, acronyme de Fast Fourier Transform est un algorithme de calcul de la transformée de Fourier Discrète [CT65] qui réduit le nombre d'opérations de $2.N^2$ à 2.N.log(N) pour une longueur de FFT de taille N. Les algorithmes de FFT se divisent en deux grandes classes : " decimation in Time " et " decimation in Frequency ". Une technique de calcul par itération successive est celle de Cooley-Tukey. L'idée est de diviser successivement par deux la taille des données sur lesquelles les calculs sont effectués.

$$\sum_{n=0}^{N-1} a_n.e^{\frac{-2.\pi.i.n.k}{N}} \tag{2.3}$$

$$\sum_{n=0}^{N-1} a_n.e^{\frac{-2.\pi.i.n.k}{N}} = \sum_{n=0}^{N/2-1} a_{2n}.e^{\frac{-2.\pi.i.2.n.k}{N}} + \sum_{n=0}^{N/2-1} a_{2n+1}.e^{\frac{-2.\pi.i.(2.n+1).k}{N}} \tag{2.4}$$

$$\sum_{n=0}^{N-1} a_n.e^{\frac{-2.\pi.i.n.k}{N}} = \sum_{n=0}^{N/2-1} a_{2n}^{pair}.e^{\frac{-2.\pi.i.n.k}{N/2}} + e^{\frac{-2.\pi.i.n}{N}}.\sum_{n=0}^{N/2-1} a_{2n+1}^{impair}.e^{\frac{-2.\pi.i.n.k}{N/2}} \tag{2.5}$$

2.2.2.2 La Transformée de Fourier Rapide : Utilisation Directe

La majorité des standards actuels dont ceux étudiés dans ce document (WIFI [WIF], WIMAX [WIM], 3GPP LTE [LTE]) présentent une option d'émission multi porteuses de type OFDM [BSE04]. Les modulations multi porteuses comme l'OFDM consistent à répartir les symboles sur un grand nombre de porteuses à bas débit, à l'opposé des systèmes conventionnels qui transmettent les symboles en série, chaque symbole occupant alors toute la bande passante disponible. Ainsi dans le cas de l'OFDM, pour un train de symboles initial de période T_{Si}, les symboles seront répartis en N trains plus lents et auront alors une durée T_S = N.T_{Si}. Pour répartir les données à transmettre sur les N porteuses, on groupe les symboles c_k par paquets de N. Les c_k sont des nombres complexes définis à partir des éléments binaires par une constellation souvent de modulation MAQ (QAM en anglais [BSE04]) à 4, 16, 64, 2.q états. La séquence de N symboles (c_0, c_1, ..., c_{N-1}) constitue un symbole OFDM. Le k-ième train de symboles parmi les N trains module un signal de fréquence f_k. Le signal modulé du train k s'écrit sous forme complexe : $c_k.e^{2.j.\pi.f_k.t}$. Le signal total s(t) correspondant à l'ensemble des N symboles ré-assemblés en un symbole OFDM :

$$s(t) = \sum_{k=0}^{N-1} c_k.e^{2.j.\pi.f_k.t}. \tag{2.6}$$

Les Fréquences f_k sont orthogonales si $f_k = f_0 + \frac{k}{T_S}$. En effet chaque porteuse modulant un symbole pendant une fenêtre rectangulaire temporelle de durée T_s, son spectre en fréquence est un sinus cardinal, fonction qui s'annule tous les multiples $\frac{1}{T_S}$. Ainsi, lorsque l'échantillonnage est effectué à une fréquence f_k d'une sous porteuse, il n'y a aucune interférence avec les autres sous porteuses. C'est ce qui permet de recouvrir les spectres des différentes porteuses et d'obtenir ainsi une occupation optimale du spectre. L'analyse algébrique indique que le signal de sortie s(t) est sous la forme :

$$s(t) = e^{2.j.pi.f_0.t}. \sum_{k=0}^{N-1} c_k.e^{2.j.\pi.\frac{k.t}{T_S}}. \tag{2.7}$$

En discrétisant ce signal et en le ramenant en bande de base pour l'étude numérique on obtient une sortie s(n) sous la forme :

$$s(n) = \sum_{k=0}^{N-1} c_k.e^{2.j.\pi.\frac{k.n}{N}}. \tag{2.8}$$

Les s(n) sont donc obtenus par une transformée de Fourier inverse discrète des c_k. En choisissant le nombre de porteuses N tel que N = 2n, le calcul de la transformée de Fourier inverse se simplifie et peut se calculer par une simple IFFT. Ainsi la modulation et la démodulation OFDM correspondent respectivement à une IFFT et une FFT. Ici, nous sommes rentré dans les détails de la modulation OFDM pour bien montrer que cette modulation est réalisée par l'opération mathématique FFT.

2.2.2.3 La Transformée de Fourier Rapide : Utilisation Indirecte

L'utilisation, dite indirecte de la FFT, consiste principalement à transposer les opérations dans le domaine fréquentiel afin de simplifier les traitements. Ainsi, dans les cas que nous explicitons ci-dessous, la FFT ne réalise pas directement les opérations concernées mais effectue une transposition dans le plan fréquentiel afin d'obtenir une nouvelle méthode de réalisation. Ainsi, une étape importante dans une chaine de réception est la détection du signal. Les normes que nous étudions proposent des préambules qui permettent la détection du signal par corrélation avec une séquence d'apprentissage connue. Les formules de la corrélation et de la Transformée de Fourier sont très proches et il est possible d'obtenir une corrélation (" cross-corrélation ", plus exactement) à partir de la FFT. La correlation de deux signaux f et g est donnée par :

$$\int_{-\infty}^{+\infty} \overline{f(\tau)}.g(t+\tau).d\tau = \int_{-\infty}^{+\infty} [\int_{-\infty}^{+\infty} \overline{F(\mu)}.e^{2.\pi.i.\mu.\tau}.d\mu. \int_{-\infty}^{+\infty} \overline{G(\mu')}.e^{-2.\pi.i.\mu'.(t+\tau)}.d\mu'].d\tau \tag{2.9}$$

$$\int_{-\infty}^{+\infty} \overline{f(\tau)}.g(t+\tau).d\tau = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \overline{F(\mu)}.G(\mu').e^{-2.\pi.i.\mu'.(\tau)}[\int_{-\infty}^{+\infty} e^{-2.\pi.i.\tau.(\mu'-\mu)}.d\tau]d\mu d\mu' \tag{2.10}$$

$$\int_{-\infty}^{+\infty} \overline{f(\tau)}.g(t+\tau).d\tau = \int_{-\infty}^{+\infty} \overline{F(\mu)}.G(\mu).e^{-2.\pi.i.\mu.(\tau)}.d\mu = IFFT(FFT(\overline{f}).FFT(g)). \tag{2.11}$$

Ainsi, la corrélation de f et g peut se faire au moyen de la FFT et de son inverse la IFFT. De même, un filtrage numérique suit la même utilisation de la FFT que la corrélation. Un filtrage est obtenue par convolution du signal d'entrée avec la réponse impulsionnelle du filtre dans le domaine temporel. Un produit de convolution est défini par :

$$x(t) \otimes y(t) = \int_{-\infty}^{+\infty} x(t-\tau).y(\tau).d\tau \tag{2.12}$$

En suivant le même raisonnement que pour la corrélation alors la convolution de deux signaux se ramènent dans le plan fréquentiel à la multiplication de ces dits signaux. De la même manière, les fonctions de décorrélation [PR02] nécessaires au niveau du récepteur qui s'assimilent à des fonctions de convolutions peuvent être menées à bien par une transposition dans le domaine fréquentiel. Palicot en [PR02], résume les différentes possibilités d'implémentations de la FFT. Outre la convolution et la corrélation qui peuvent se réaliser dans le domaine fréquentiel, une abondante littérature décrit les possibilités d'implémenter des égaliseurs et d'opérer l'estimation de canal en fréquence. Ces différentes implémentations sont largement décrites en [FCG95] pour le FLMS (Frequency Domain Least-mean square), en [MG82] pour le FLMS sans contrainte (UFLMS) (Figure 2.3), en [BP96] pour l'algorithme de Quasi-Newton et [BP95] pour les égaliseurs

adaptatifs à retour décision. Ainsi, une grande gamme d'égaliseurs peut être implémentée via la FFT par transposition dans le domaine fréquentiel.

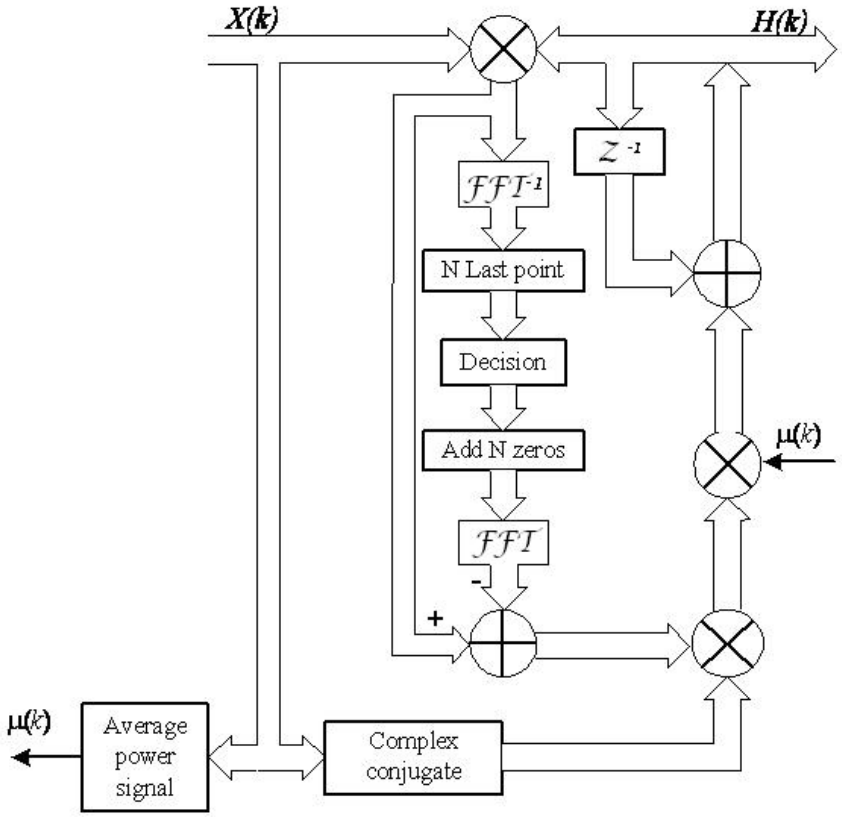

FIGURE 2.3 – Egaliseur UFLMS

En suivant le même procédé de passage en fréquence, en [WH01] et en [Hen02], il a été démontré que respectivement les calculs d'un RAKE et d'une mise en canaux (figure 2.4) pouvait se faire à l'aide d'une FFT. Ainsi, celle-ci peut être opérationnelle pour la corrélation, la convolution, le filtrage, l'estimation de canal, l'égalisation, le désétalement de spectre. La conclusion primordiale à retenir des travaux présentés en [PR02] est qu'une seule et même opération peut être utilisée pour réaliser en totalité ou en partie différentes fonctions de nature distincte, diminuant ainsi le nombre d'algorithmes à considérer dans l'implémentation d'un terminal.

2.2.3 L'opérateur CORDIC

L'algorithme CORDIC, (acronyme de COordinate Rotation DIgital Computer) fut introduit à l'origine [Vol59], pour répondre à des besoins de calculs tels que les rotations de vecteurs ou les changements de coordonnées cartésiennes-polaires et polaires-cartésiennes dans le plan euclidien. L'algorithme recourt uniquement à des primitives

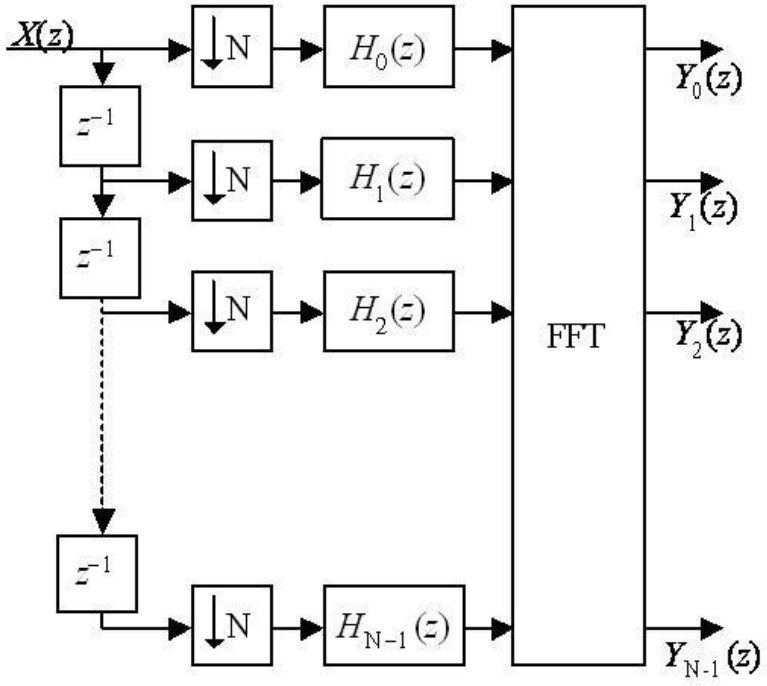

FIGURE 2.4 – FFT en banc de filtres

élémentaires en base 2 : additions, soustractions et décalages et permet accessoirement le calcul de racines carrées et de fonctions trigonométriques sinus, cosinus et arctangente. Une généralisation [Wal71] en a été proposée, par extension à des espaces pseudo-euclidiens dotés d'une métrique hyperbolique ou linéaire, permettant le calcul direct de la multiplication, de la division, des sinus, cosinus et argument-tangente hyperboliques, ainsi que d'exponentielles, et le calcul indirect (par composition des fonctions précédentes), de tangentes circulaires et hyperboliques et de logarithmes. Il apparaît dans cette perspective plus large comme un nouveau paradigme en fonction duquel peut être repensé l'algorithmique d'un grand nombre de fonctions avancées du traitement numérique du signal, aux fins de l'utilisation de l'opérateur CORDIC intégré comme module de base dans une éventuelle réalisations sous forme de circuit spécifique. On peut citer comme exemples d'applications : le filtrage complexe de signaux analytiques, la modulation en bande latérale unique, la transformée de Fourier discrète, directe et rapide [Des79] [Des74], le filtrage fréquentiel sur structures à faible sensibilité en bande passante (treillis de Gray-Markel, filtres orthogonaux et filtres d'onde [RK84][Dep83]), la factorisation spectrale de processus stationnaires par l'algorithme de Schur normalisé [DDN85][Kai85a], la modélisation adaptative de processus non stationnaires (filtrage récursif optimal au sens des moindres carrés sur structure en treillis normalisé [ALMA81], filtrage de Kalman [SH86]) . Plus généralement il intervient dans la résolution d'un grand nombre de problèmes d'algèbre linéaire, dont certaines fonctions précédentes apparaissent comme des cas particuliers : résolution exacte ou approchée au sens des moindres carrés de systèmes linéaires (algorithmes orthogonaux de Givens [9], Fadeeva [17]), équations aux valeurs propres, décomposition en valeurs singulières [SF84][CL87], décomposition QR, de Cholesky [ADM82][MMAD82], le cas des matrices de Toeplitz ou quasi-Toeplitz (à rang de déplacement fini), donnant toujours lieu à des algorithmes spécifiques [JHF86]. L'idée commune à tous ces algorithmes (à l'exception de ceux opérant directement sur des valeurs complexes), est de normaliser les calculs par l'utilisation exclusive à chaque étape de transformations orthogonales. L'utilisation de l'opérateur CORDIC présente également de nombreux avantages pour l'intégration au niveau purement architectural, par son excellente adaptation aux contraintes des VLSI : un même module de base, précaractérisé et éventuellement paramétré avec possibilité de choix des degrés de parallélisme, de pipelinage [Ren87], pour une adaptation optimale aux débits requis par l'application, peut servir comme constituant élémentaire dans une grande variété d'architectures spécialisées, avec une complexité en matériel inférieure à ce qu'on obtiendrait par l'utilisation d'opérateurs classiques [Ahm85]. Concrètement, la résolution de fonctions telle que sinus ou cosinus par l'algorithme de CORDIC s'appuie sur une méthode de rotation de vecteur dans le plan cartésien. Nous développons en Annexe H, le principe mathématique de l'algorithme CORDIC et proposons une nouvelle implémentation de ce dernier.

2.3 Définition de La Technique des Opérateurs Communs

Nous présentons dans la section précédente, l'origine du concept d'opérateur commun ([PR02]) et l'objectif de son utilisation ([MPRG06]). Dans nos travaux de recherches nous formalisons la technique des opérateurs communs en ([ALR+09]) afin de proposer une méthode entièrement définie et pouvant s'appliquer dans des cas réels d'implémentations. Ainsi dans cette section, nous détaillons précisément ce que nous définissons comme opérateur commun et comment nous comptons utiliser de telles entités.

2.3.1 Le Concept de l'Opérateur Commun

2.3.1.1 L'Opérateur Commun (OC)

L'Opérateur Common La section 2.2 précédente, a permit de mettre en lumière, la possibilité d'outrepasser cette dépendance aux standards, en utilisant les même fonctions pour des implémentations spécifiques. Or les fonctions présentées en 2.2 ne sont plus des fonctions au sens de la fonction commune de Rhiemieir [Rhi02] mais des fonctions mathématiques, c'est à dire des "opérateurs" qui systématiquement opèrent la même opération à un paramètre près. Ainsi, à partir de cet état de l'art, nous proposons dans cette thèse de prolonger la définition des opérateurs communs initialement présentée en [MPRG06].

L'opérateur commun s'intègre dans les travaux relatifs à la paramétrisation et se défini à postériori des fonctions communes présentées par Rhiemier avec comme motivation première de construire un terminal à partir d'entités aussi génériques que possible et de ne plus être dépendant d'un standard particulier. Pour ce faire, comme Palicot et Moy en [MPRG06], nous prenons le contrepied de la paramétrisation proposée par Rhiemier en ne se focalisant plus sur des macro blocs fonctionnels mais sur des entités de granularité inférieure aux fonctions. La Figure 2.5 illustre notre propos. En se basant sur les possibilités existantes d'utiliser des opérateurs mathématiques (FFT ou CORDIC) pour réaliser des fonctionnalités différentes d'un même terminal, nous proposons de développer des entités de granularité inférieure aux fonctions et par conséquent aux fonctions communes. Ainsi, le terminal ne sera plus vu comme une succession de fonctions communes mais comme un ensemble d'opérateurs qui seront alloués par des fonctions afin d'exécuter un standard donné.

Ainsi, l'ajout d'un nouveau terminal, d'un nouveau mode ou d'une nouvelle fonction pourra se faire sans changer le design du terminal mais en re-paramétrant les opérateurs leurs étant alloués. L'utilisation d'un opérateur commun pourra se faire par des standards différents et à plusieurs reprises par des fonctions distinctes d'un même standard pour l'exécution d'opérations spécifiques. De manière à créer des opérateurs indépendants des standards les utilisant, nous voulons définir des entités génériques, entièrement définies qui sont implémantables dans un terminal et vu comme des IP de granularité intermédiaire. Ainsi, dans un contexte de Radio Logicielle, nous posons en [ALR+09], la définition de l'opérateur commun comme "une entité hardware ou software paramétrable, prédéfinie dans son design et capable par simple modification de paramètres de

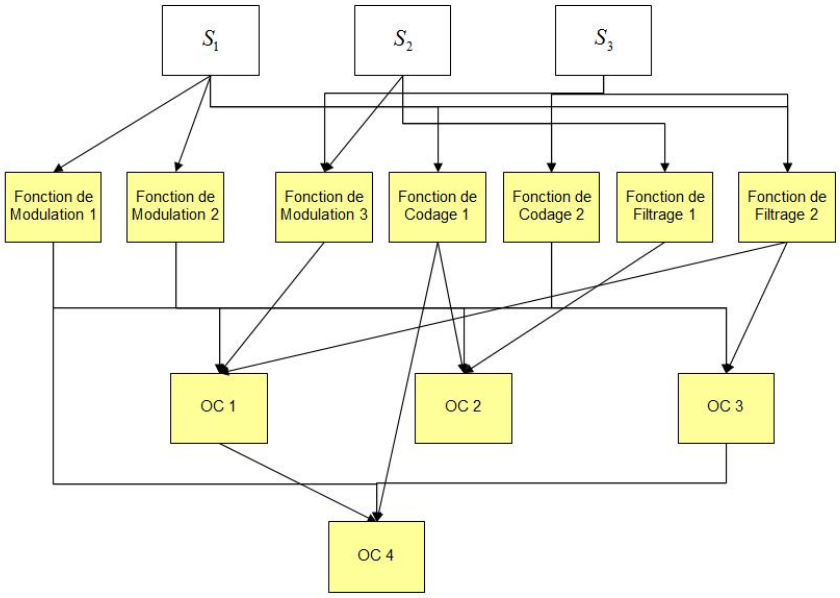

FIGURE 2.5 – Représentation des Opérateurs Communs

s'adapter à l'exécution d'une opération donnée." Un opérateur commun est donc un composant - soft ou hard - paramétrable qui peut réaliser plusieurs opérations sans en connaitre la teneur. Les premiers opérateurs communs que nous pouvons citer sont ceux présents dans la littérature et implémentés dans n'importe quel design que sont, les cellules logiques, les bascules ou bien encore les opérateurs de Multiplication, Addition, Accumulation (MAC). Ces opérateurs ne sont pas dépendants d'un standard et sont utilisés dans la réalisation pratique d'un terminal par différentes fonctions sans que ces derniers ne soient spécifiquement prévues pour une fonctionnalité définie. Le principe de l'opérateur commun a pour objectif de définir un ensemble similaire d'opérateurs mais de granularité supérieure, se situant entre les fonctions et ces dits éléments de base. La figure 2.6, illustre notre propos. Celle-ci se compose de plusieurs niveaux :

- Le premier niveau des différents standards à considérer.
- Un deuxième niveau comprenant Fonction Communes et/ou Fonctionnaliés.
- Un troisième niveau d'opérateurs.
- Le quatrième niveau des "opérateurs de base".

Ce sera par conséquent parmi ce troisième niveau d'opérateurs que nous définirons les opérateurs communs. Le graphe de la figure 2.6 donne une représentation de la décomposition fonctionnelle des opérations de telle sorte qu'on puisse identifier quelle fonction appelle quelle autre afin de déterminer le meilleur set. Ainsi, on n'explicite pas ici comment l'opérateur commun sera concrètement implémenté, c'est à dire en Hard ou en Soft. Notre approche idéale est de pouvoir définir un opérateur commun

comme un composant implémentable d'une manière logicielle ou matérielle et ainsi de pouvoir migrer de l'une à l'autre selon le contexte. Cette idée de migration entre une implémentation Hard ou Soft de la même opération a été développé historiquement par Andrews en [ANJ+04] et Liang et Zhou en [LZWP07]. Dans ces travaux, l'idée est de pouvoir "en ligne" redistribuer en temps réel une opération via son code logiciel ou son implémentation Hard afin d'optimiser l'allocation des ressources et de respecter les contraintes temps réel. Néanmoins, au moment de l'élaboration, cette méthode suppose la duplication dans le Soft et le Hard du même opérateur, ces derniers n'étant pas reconfigurables par essence et encore moins paramétrables. Un surcout d'implémentation est donc requis. Cette vision indépendante de la cible doit être mise en relation avec les travaux de Delaye [WDLP07] qui classifie les fonctions de la chaine de traitement en trois catégories les associant à une cible privilégiée :

- Une classe "Coding Class" qui regroupe les fonctions de codage canal, caractérisée par une grande diversité des schémas de codage et nécessitant une forte flexibilité avec comme cible privilégiée un DSP.
- Une classe "Data Structuring" qui manipule les données et qui nécessite une grande capacité de mémoire avec comme cible privilégiée un μP.
- Une classe liée à la modulation qui nécessite de grandes capacité de calculs avec comme cible privilégiée un FPGA.

La définition de l'opérateur commun se fera donc indépendamment de la cible choisie. Néanmoins, même si l'opérateur peut être défini en logiciel ou en matériel, une fois une cible choisie, cet opérateur présentera des spécificités propres liées à la technologie utilisée. Ces spécificités seront en prendre en compte lors de l'implémentation du terminal multistandard et nous oblige à considérer chaque instance du même opérateur commun sur une cible donnée comme un opérateur commun spécifique. De manière à obtenir, des premiers résultats d'implémentations, nous faisons le choix dans nos travaux de considérer uniquement l'implémentation d'opérateurs communs matériels.

L'Opérateur Common Matériel En se focalisant sur des opérateurs matériels, nous voulons définir l'opérateur commun comme indépendant d'une fonctionnalité afin que celle-ci ne corresponde au final uniquement qu'à une paramétrisation spécifique de ce dit opérateur. Ainsi, à l'instar des éléments de base de toutes architectures (XOR, MAC [Pro07],...), nous définissons l'unicité de l'architecture de l'opérateur commun matériel. C'est-à-dire que :

- " Un Opérateur Commun Matériel présentera un design entièrement défini et qui ne peut être modifié sous la peine de définir un nouvel opérateur commun"

De manière à vérifier l'indépendance vis-à-vis des structures à remplacer dans le terminal mais aussi d'une implémentation Velcro, nous ajoutons à la définition précédente que l'opérateur commun est constitué d'une sous structure qui sera systématiquement

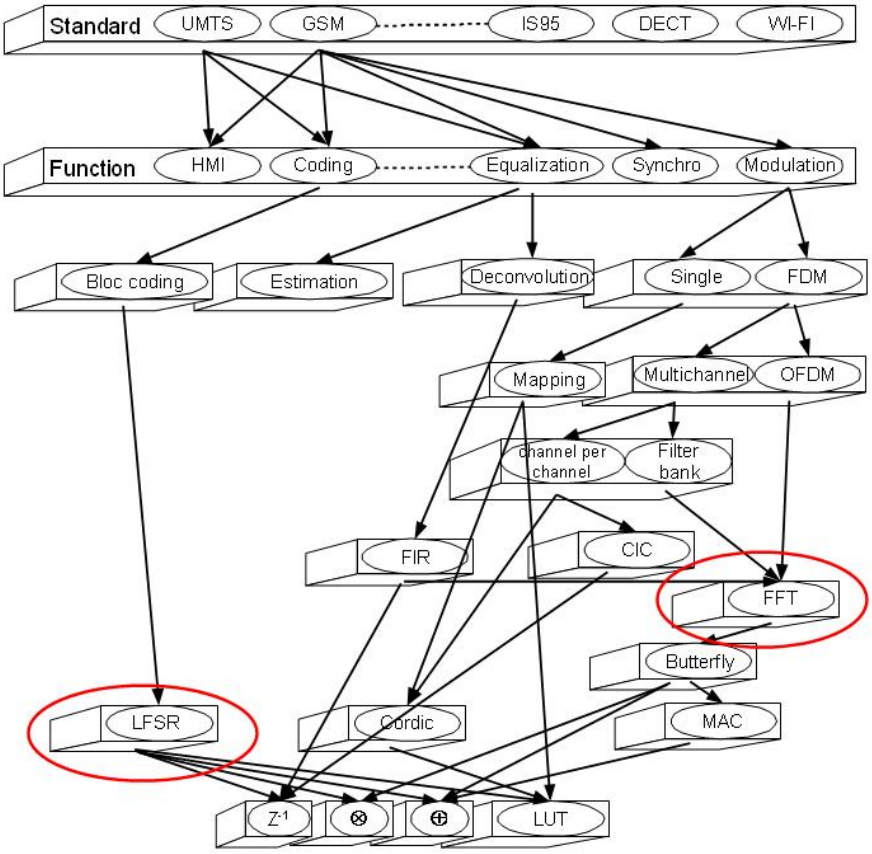

FIGURE 2.6 – Definition de l'Opérateur Commun

utilisée dans l'ensemble des opérations qu'il traite. Ainsi, le paramétrage modifiera les interconnexions propres à la structure et aux éléments adjacents tout en utilisant systématiquement cette colonne vertébrale de l'OC. Par exemple, dans le cas des opérateurs LFSR définis en 4.3, la sous structure est constituée du système de registres à décalages qui, quelque soit la paramétrisation, est utilisé. Cette propriété permet de présenter l'OC comme une réelle entité paramétrable et non comme la concaténation de différentes architectures sélectionnées par un simple "switch". Historiquement, Noguet a développé en [Nog04], une architecture systolique reconfigurable adaptée à la détection en temps réelle UMTS/TDD. Cette architecture est basée sur un enchainement de deux sortes de cellules (MAC et DIAG, cf [Nog04]) qui constituent la structure de base systématiquement utilisée dont les interconnexions sont paramétrables.

De ce fait, un OC pourra être entièrement défini par un ensemble de caractéristiques. Ces dernières pourront être aussi variées et nombreuses que le designer le juge nécessaire. Néanmoins, dans ces travaux et vis à vis des explications nécessaires qui seront entreprises par la suite, nous en définissons deux initiales :

- Le Coût de surface de l'opérateur. Noté N_S, il correspond au coût unitaire de l'implémentation physique d'un Opérateur Commun.
- Le Temps d'exécution de l'opérateur. Notés T_e, il correspond au temps d'exécution de l'opérateur pour une exécution.

Si le coût de surface se définit aisément, il n'en est pas de même pour le temps d'exécution. En effet, selon la structure de l'opérateur, de la paramétrisation choisie et du contexte d'implémentation, le temps pour exécuter "correctement" l'opérateur peut varier. Ainsi, dans ce contexte et vis à vis des explications ultérieures nous posons l'existence d'un temps T_e caractéristique du temps d'exécution d'un opérateur commun. Ce temps d'exécution sera définie comme le maximum du temps d'exécution pour toutes les paramétrisations possible et permet de qualifier la fréquence maximale F_e d'exécution de l'opérateur commun. Outre ces deux premiers coûts, d'autres paramètres peuvent servir à définir un opérateur commun ; consommation dynamique, statique....

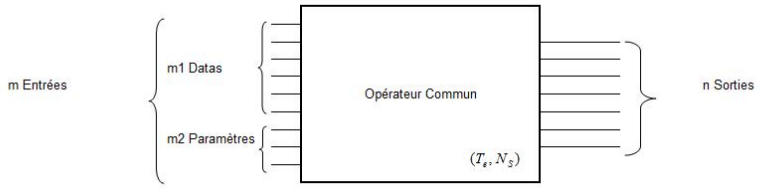

FIGURE 2.7 – L'Opérateur Commun

Ainsi, l'opérateur commun peut être vu comme une boite noire composé de m entrées,

n sorties. Pour chaque paramétrisation, un sous ensemble variable (m2) des m entrées constituera les paramètres permettant de modifier l'architecture reliant entrées et sorties (figure 2.7). Tout l'intérêt d'implémenter un opérateur commun dans la résolution du Handover Vertical est de procéder au paramétrage par le biais d'une sous partie des données entrantes. Il n'y a pas de reconfiguration à proprement parlé du terminal mais une reparamétrisaton. Ainsi, l'opérateur commun traitera les m2 paramètres au même titre que les m1 données, sauf qu'ils serviront à reparamétrer l'architecture qui elle, traitera les données. La paramétrisation est donc une méthode qui "reconfigure" en temps réel.

2.3.1.2 Le Graphe des Opérateur Commun (GOC)

Un opérateur commun ne représente donc plus une fonctionnalité ni une macro fonction mais selon la paramétrisation, une opération nécessaire pour une fonctionnalité ou même une fonction commune. Ainsi, l'exécution d'une seule fonction peut nécessiter l'utilisation d'un ou d'une combinaison de plusieurs opérateurs communs (Figure 2.5). Cependant, outre la spécificité d'une fonction ou d'un standard à utiliser un opérateur commun (Figure 2.6), dans notre vision de la paramétrisation, illustrée en Figure 2.5, un opérateur commun peut être utilisé par un autre opérateur commun qui se situe à niveau de granularité supérieur dans la décomposition graphique d'un standard. En [WLP06], Wang définit une architecture flexible pour l'implémentation sur FPGA de l'algorithme de détection des signaux MIMO de type V-BLAST "Square Root" dans un contexte "Radio Logicielle". L'architecture proposée utilise l'opérateur élémentaire CORDIC comme opérateur de base ; l'optimisation des ressources matérielles s'obtient en adaptant dynamiquement le degré du parallélisme à la puissance de calcul nécessaire pour une configuration MIMO et un débit donnés. Il apparait donc indispensable dans l'implémentation d'opérateurs communs matériels de pouvoir recourir à la parallélisation des traitements et à dupliquer physiquement notre opérateur matériel. Ainsi, si nous considérons l'opérateur commun OC_1 de granularité supérieure à l'opérateur commun OC_2 et qui peut être exécuté à l'aide de OC_2, OC_1 sera qualifié d' "opérateur appelant" et OC_2, "opérateur appelé". Dans ce cas de figure, nous définissons deux paramètres de liaisons entre ces deux opérateurs :

- Coût de Construction. Ce paramètre noté N_C correspond au nombre d'instances simultanées nécessaires de OC_2 pour exécuter la même opération unitaire que OC_1.
- Coût d'itération. Ce paramètre noté N_T, correspond au nombre d'itérations successives nécessaires des N_C OC_2 pour exécuter la même opération unitaire que OC_1.

La définition de ces deux paramètres se basent sur la possibilité de distinguer les instances physiques réellement implémentées et les instances virtuelles disponibles de chaque opérateur. En [BN04], Biard étudie la répartition de la charge de travail dans une implémentation d'un décodeur Viterbi reconfigurable où le nombre d'instances physiques de papillons ([BN04]) peut varier, modifiant ainsi les allocations des instances virtuelles au cours du temps. Pour illustrer ce propos, nous pouvons prendre un exemple

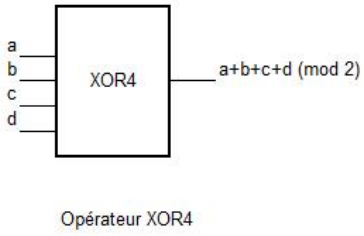

FIGURE 2.8 – L'Opérateur XOR4

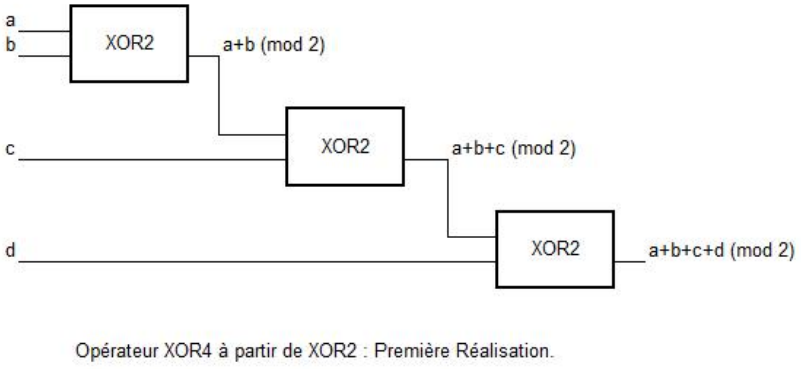

FIGURE 2.9 – L'Opérateur XOR4 : Première réalisation avec XOR2

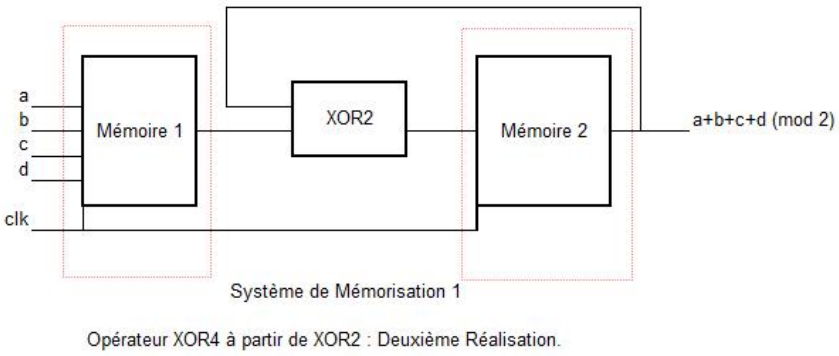

FIGURE 2.10 – L'Opérateur XOR4 : Deuxième réalisation avec XOR2

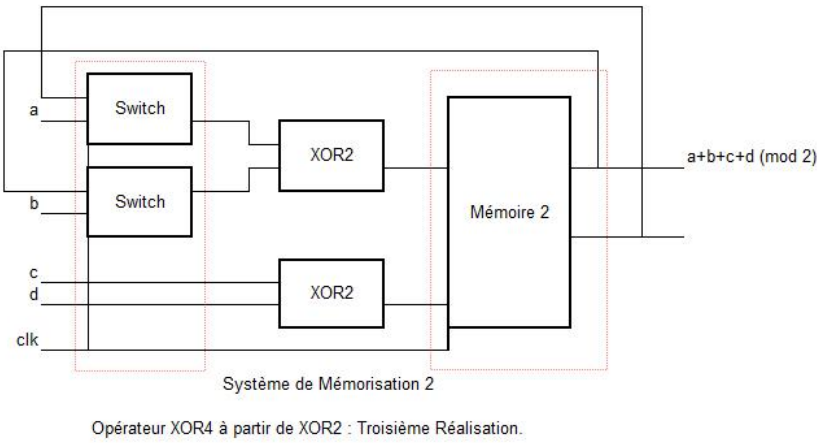

FIGURE 2.11 – L'Opérateur XOR4 : Troisième réalisation avec XOR2

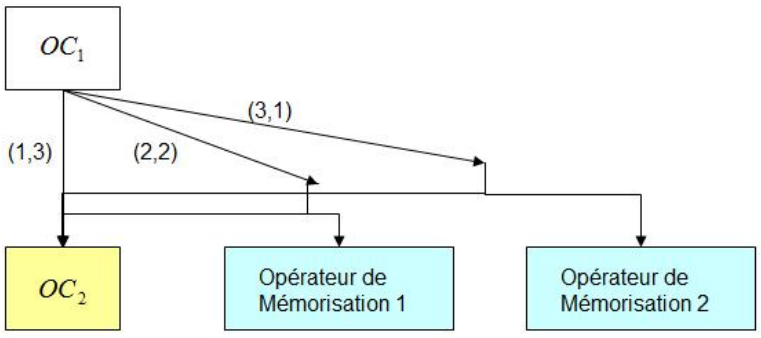

FIGURE 2.12 – L'Opérateur XOR4 : Réalisation avec XOR2

très simple d'un opérateur commun OC_1 qui serait un additionner binaire modulo 2 à quatre entrées (XOR4)(figure 2.8) et un opérateur commun OC_2 qui serait un additionner modulo 2 à deux entrées (XOR2). De manière triviale (figure 2.9), un XOR4 est "équivalent" à trois XOR2. Ainsi, selon la méthode d'implémentation que nous choisissons, nous pouvons définir différents couples (N_T,N_C) qui caractérisent la relation entre ces deux opérateurs.

- Si nous considérons une équivalence où nous implémentons autant de XOR2 que nécessaire, alors nous aurons le couple de paramètres (N_T=1,N_C=3) (figure 2.9).
- Si nous considérons que nous implémentons un seul XOR2, sous condition d'utiliser un système de mémorisation des données (entrées/sorties), la relation sera définie par (N_T=3,N_C=1) (figure 2.10).
- Nous pouvons également définir un cas intermédiaire où deux XOR2 seraient implémentés. Dans ce cas, nous aurions (N_T=2,N_C=2) (figure 2.11), pour 4 XOR2 virtuellement disponibles dont 1, non utilisé.

Ainsi, chaque opérateur commun peut être "relié" à un opérateur de granularité inférieure selon plusieurs couples de paramètres N_T et N_C (figure 2.12). Néanmoins, comme chaque opérateur commun est unique et défini par les deux paramètres initiaux T_e et N_S, alors si nous considérons un ensemble fini d'opérateurs communs, celui-ci pourra être représenté par un graphe acyclique unique où les connections entre opérateurs communs seront fixes (Figure 2.13). Ici, nous définissons le graphe des opérateurs communs (GOC). Celui-ci sera la pièce maitresse de la technique des opérateurs communs car de son identification et de son exploitation dépendra le design de notre terminal multistandard.

Nous devons préciser que le graphe que nous considérons est acyclique de granularité descendante. Le graphe est vu comme un graphe de construction architectural respectant le niveau de granularité des opérateurs les uns par rapport aux autres et non comme un graphe de fonctionnement ou d'exécution. Si nous reprenons, l'exemple précédent, le XOR4 est de granularité supérieure au XOR2 est peut être construit par le XOR2. Or un XOR2 peut être exécuté par un XOR4. Néanmoins, le XOR2 ne sera pas relié au XOR4 dans le graphe dans le sens XOR2 vers XOR4 car la dépendance que nous considérons dans le concept de l'opérateur commun est une dépendance de construction et non d'exécution.

Il est à noter que la définition de l'opérateur commun est restrictive. Cela afin de définir réellement l'OC comme un nouvel IP pour la radio logicielle restreinte. Ainsi, comme un OC doit être définie par un minimum de deux coûts fixes N_S et T_e, des opérateurs mathématiques comme la FFT ne pourront pas figurer directement dans le graphe des opérateurs communs. La FFT est une opération mathématique clairement définie mais dépendant d'un paramètre N caractérisant le nombre d'échantillons considérés. Selon ce paramètre N, le nombre d'itérations de son code ou de manière équivalente le nombre de papillons [AGLP06] de la FFT (butterfly) sera requis en nombre différents. Ainsi, pour un papillon de type Radix 2 [AGLP06], ce nombre s'élève à $(\frac{N}{2}.log_2(N))$. Même si le radix 2 est lui même un opérateur commun entièrement défini avec un coût d'implémentation $N_{S,Radix2}$, alors selon la taille de la FFT voulue, le coût $N_{S,FTT}$ sera de $\frac{N}{2}.log_2(N).N_{S,Radix2}$. Celui-ci ne sera pas fixe et dépendra de N. Si nous désirons inté-

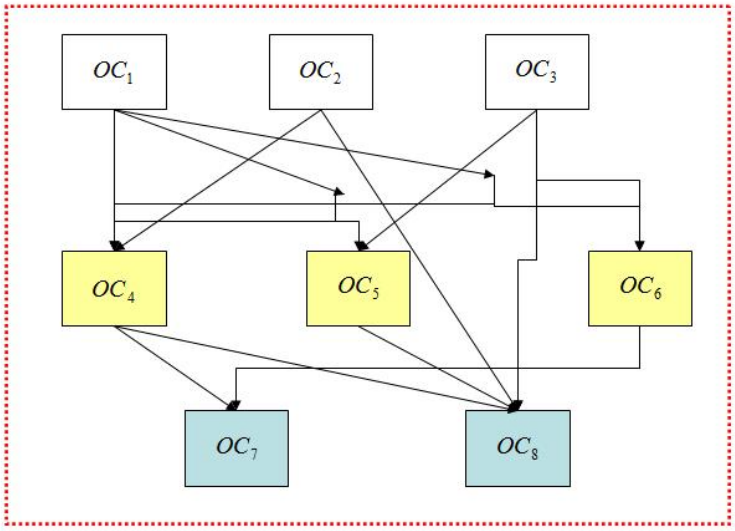

FIGURE 2.13 – Graphe des Opérateurs Communs

grer la FFT dans le graphe des opérateurs communs, nous devrons considérer différentes FFT caractérisées par leurs Tailles (FFT 16, FFT 64, ...). Or si un module FFT64, par exemple, peut réaliser une FFT16, alors N deviendra un élément de paramétrisation du design.

2.3.2 La Technique des Opérateurs Communs

2.3.2.1 Principe

Dans la création d'un terminal multistandard reconfigurable adapté à la Radio Intelligente, la technique des opérateurs communs défini un Terminal construit à partir d'un set limité d'opérateurs communs dont nous venons de donner la définition. Les motivations de cette méthode sont doubles. La première que nous avons déjà explicité dans ce rapport est de ne plus être dépendant des fonctions de la chaîne de télécommunication afin d'obtenir un terminal reconfigurable tout en étant évolutif. L'évolutivité ne pourra jamais être "garantie" mais en utilisant des opérateurs communs, indépendants des standards, le terminal sera plus susceptible à satisfaire aux besoins d'une évolution. La deuxième motivation de cette technique est l'optimisation en termes de complexité. En se basant sur un nombre limité d'opérateurs, la technique vise à obtenir une optimisation du design Hardware.

Dans la section 2.3.1, nous définissons le concept d'opérateurs communs et de graphes d'opérateurs communs. La technique des opérateurs communs consistera à l'identification et l'exploitation d'un tel graphe. En effet, comme le GOC est unique et fixe pour un ensemble d'opérateurs donnés, nous pouvons à partir de ce graphe

construire la graphe intégral d'un terminal multistandard donné. Ce graphe reprend la première représentation illustrative de la figure 2.6 en y appliquant le formalisme lié aux connections entre opérateurs et fonctions. Chaque opérateur du GOC constitue un point d'entrée à partir duquel nous pouvons mapper les besoins des différents standards. Une fois que nous aurons " mappé " lesdits standards sur l'ensemble des opérateurs communs, nous obtenons plusieurs possibilités de réalisation de notre terminal par différentes combinaisons d'OCs (Figure 2.14). Pour un terminal, donné nous pourrons évaluer selon des critères spécifiques (coût de surface, consommation, etc), la meilleure combinaison d'OC à implémenter. Ainsi, nous donnons la définition de la méthode :

"La technique des opérateurs communs consiste à identifier l'ensemble des opérateurs communs possibles et à déterminer le meilleur set capable de réaliser le terminal multistandards voulu au sens d'un critère d'optimisation à minimiser".

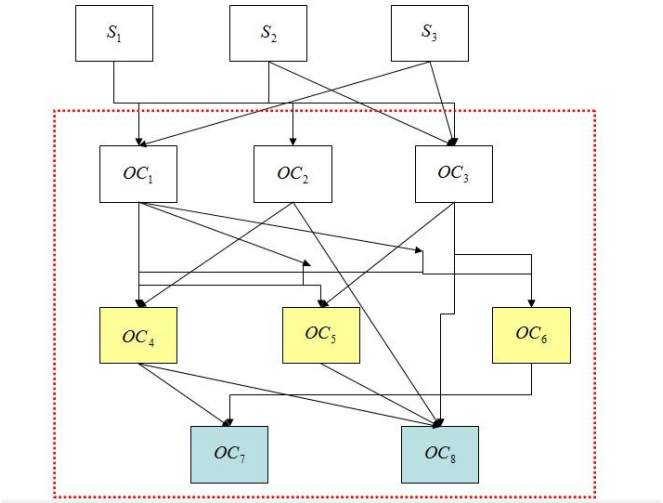

FIGURE 2.14 – Décomposition d'un standard en Opérateurs Communs

La technique des opérateurs communs consiste donc en deux étapes : l'identification des dits opérateurs communs et l'exploitation de ces opérateurs communs afin de définir le terminal optimal selon un critère spécifique. Toute la question est de savoir comment obtenir les opérateurs communs. Afin d'atteindre cet objectif d'identification et de choix des Opérateurs Communs, nous définissons deux approches dans le cadre de la techniques des opérateurs communs.

2.3.2.2 Approche Théorique

Le principe de l'approche théorique reprend la vision initiale décrite par Rodriguez en [MPRG06]. Il s'agit de la vision idéale de l'opérateur commun qui peut être implémenté une seule fois et dont les appels successifs par des fonctions différentes permettent

de réaliser l'ensemble de la chaîne de traitement. Les éléments à considérer comme Opérateurs Communs sont sélectionnés par un procédé d'optimisation décrit par Rodriguez et reprit dans les travaux de GUL. Ainsi, la méthode consiste à décomposer le terminal multistandard en autant d'opérateurs de granularités différentes possibles, d'un grain haut proche de la fonction jusqu'au grain le plus bas des opérations élémentaires. A partir de cette décomposition, l'objectif est d'étudier pas à pas, toutes les solutions et de ne retenir que les plus optimales. Ainsi, en parcourant le graphe, nous déterminons quels sont les opérateurs les plus propices à être implémentés. L'idée fondatrice de cette approche est de permettre la définition de nouveaux opérateurs à partir d'une combinaison des opérateurs déjà présents dans le graphe à sa conception, si celle-ci permet une optimisation du design. Nous distinguons alors deux types d'opérateurs, ceux qui sont déjà présents dans le graphe lors de la décomposition initiale (opérateurs existants) et ceux qui sont "construits" par l'approche (opérateurs construits).

Un exemple a été donné de cette approche dans des publications connexes à nos travaux en [GMP07]. Ici, Gul travaille sur un prolongement des travaux de Rodriguez [MPRG06] sur l'algorithme de sélection en illustrant le principe de l'approche théorique par un terminal multistandard basé sur le WIFI et l'UMTS. Nous explicitons, ici, cet exemple afin d'illustrer le principe de l'approche théorique. La figure 2.15 représente la décomposition schématique de cette application. Ainsi, Scrambler, Convolutional Coder et Interleaver nécessitent un opérateur commun qui est une Bascule (Flip Flop). Or pour traiter les données de chaque symbole OFDM de taille N_{CBPS}, l'interleaver nécessite concrètement N_{CBPS} x $2.log_2(N_{CBPS})$ bascules. Ainsi en [GMP07], Gul propose par cette approche théorique de définir un nouvel opérateur commun définis par N_{CBPS} x $2.log_2(N_{CBPS})$ bascules et 2 XOR. Cet opérateur n'existe pas dans la decomposition initiale du standard, il est crée de toute pièces en optimisant pas à pas le graphe. Nous mutualisons nos travaux avec ceux de GUL en [GAP+10], en intégrant les opérateurs à base de registres à décalages que nous avons développés.

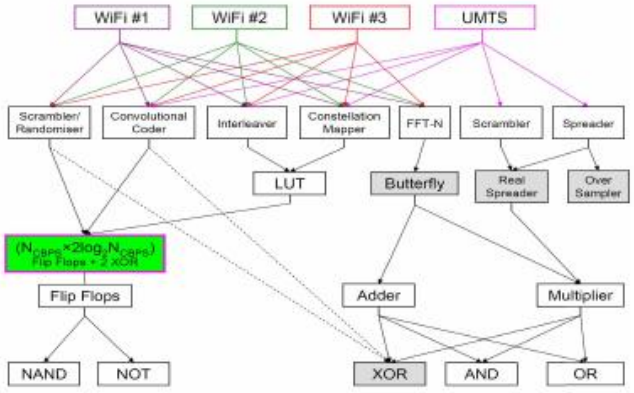

FIGURE 2.15 – Approche Théorique

2.3.2.3 Approche Pragmatique

La deuxième approche que nous développons dans ces travaux et que nous utilisons dans le reste du document est une approche qualifiée de pragmatique. Celle-ci arbore un coté plus pratique dans l'application de la technique des opérateurs communs. En effet, l'intégration de l'opérateur commun dépendra de la cible du terminal. Deplus, comme nous le mentionnons dans la section 2.3.1.2, l'opérateur commun matériel peut être dupliqué pour répondre au besoin des différents standards. En outre, il nous ai apparu pertinent de laisser au designer la possibilité de définir des opérateurs communs à partir d'une étude spécifique des différentes spécifications des normes de manière à pouvoir accroître les liens dans le graphe.

Ainsi, l'approche pragmatique distingue les étapes d'identification des opérateurs communs et le choix de ces derniers. Dans un premier temps, nous construisons le graphe des opérateurs communs à partir d'opérateurs communs identifiés ou construits et dans un second temps nous appliquons un algorithme de choix. Cet algorithme de choix sera plus complexe que l'algorithme initial de Rodriguez car il devra considérer les instances virtuelles et physiques, ordonnées dans le temps. Nous expliquons ce point en 2.4.

2.3.2.4 Compromis de la Technique.

Si l'opérateur commun a pour objectif d'être paramétrable et utilisé par le maximum de fonctions possibles, les objectifs à proprement parlé de la technique des opérateurs communs appliquée à un terminal multistandard doivent être définis. Le premier objectif d'une telle méthode est la reconfigurabilité du système, c'est-à-dire la capacité du terminal à se reconfigurer pour des modes définis à l'avance. Outre ce premier objectif, la technique des Opérateurs Communs présente deux autres visées qui peuvent paraître opposées. En effet, l'opérateur commun factorise un design en se substituant a différents opérateurs des multiples standards. En l'exécutant à plusieurs reprises tout au long de la chaine d'émission/réception nous pouvons supposer que l'implémentation de la technique des opérateurs communs engendrera un gain en complexité vis-à-vis d'une approche Velcro traditionnelle notamment. Or développer un design " évolutif " et obtenir " un gain de place " constituent tout l'enjeu de la méthode, mais sont deux objectifs parfois différents. En effet, la volonté de créer un opérateur évolutif peut résulter en une architecture complexe qui une fois factorisée pourrait s'avérer plus " coûteuse " que l'ensemble des structures qu'elle vise à remplacer. Réciproquement, la décision de préférer l'optimisation mettant en oeuvre des opérateurs moins génériques vis-à-vis d'opérateurs plus complexes mais offrant plus de possibilité d'implémentations, va à l'encontre d'une chaîne reconfigurable et évolutive. Pour prétendre avoir une architecture évolutive, il est pertinent de supposer que nous devrions être capables au préalable de réaliser toutes les opérations " existantes " par un ou plusieurs OCs. Ceci afin de pouvoir envisager que tout autre nouveau standard se greffera intégralement sur nos opérateurs. L'aspect " Evolutif " peut nuire au gain de place et la recherche de gain peut restreindre les possibilités d'évolution du terminal. Un compromis sera des lors nécessaire de la part du designer.

2.4 Gestion des Opérateurs Communs

Développer l'intégralité d'un terminal multistandard basé sur les opérateurs communs et fournir une méthode générique de gestion est une tache bien au delà des aspirations de nos travaux de recherches. Nous venons de décrire ce qu'était pour nous l'opérateur commun et la technique des opérateurs communs. La question de l'ordonnancement des ressources matérielles est une question pertinente tant le développement des plateformes hybrides et reconfigurables se développent. La technique des opérateurs communs peut se voir comme une méthode de virtualisation des ressources à partir d'un nombre limité d'éléments réellement implantés. En effet, selon la paramétrisation, l'opérateur commun sera apte à réaliser telle ou telle opération d'un fonctionnalité donnée. Ainsi, nous obtenons un nombre de ressources plus grand que le nombre présent physiquement sur la plateforme. La gestion d'une telle virtualisation fait déjà l'objet de recherches poussées. Ainsi, un projet ANR appelé FOSFOR dont les acteurs principaux sont le CNRS (LEAT), l'IRISA, Thales et Xilinx travaille sur cette thématique. FOSFOR [BMB07] pour Flexible Operating System FOr Reconfigurable platform est un projet qui vise à reconsidérer la structure du RTOS qui est généralement logiciel, centralisé et statique en un RTOS flexible, distribué en proposant d'exploiter la reconfiguration dynamique et partielle des SoC reconfigurables. Ce projet propose une approche qui rendra possible cette flexibilité de l'OS grâce à des mécanismes de virtualisation des services de l'OS nécessaire pour que les tâches de l'application s'exécutent et communiquent sans connaissance a priori de leur affectation à une unité de traitement logicielle ou matérielle. Ce projet ne prend pas en compte une reconfiguration par paramétrisation d'opérateurs communs mais par reconfiguration partielle du FPGA, qui elle aussi procure une virtualsation des ressources.

Vis à vis de toutes ces considérations, nous devons clairement poser quels axes nous avons choisis de suivre et quels points concrets sont abordés dans le suite de ce rapport. Tout d'abord, nous nous focalisons sur des opérateurs matériels comme défini en 2.3.1.1. Ensuite, nous suivons une approche pragmatique quant à l'identification de ces opérateurs. Les opérateurs basés sur la FFT, les LFSR ou les CORDIC que nous présentons ont été définis à partir de l'étude des différents besoins du terminal afin de proposer des structures communes paramétrables. Finalement, nous implémentons dans divers cas de figures les opérateurs communs définis afin d'étudier l'impact de la technique des opérateurs communs sur le terminal et de mettre en lumière la problématique d'une telle méthode. Jusqu'à présent, les recherches actuelles sur les opérateurs communs n'ont pu que définir les opérateurs en eux-mêmes, laissant de coté la question de leurs intégration effective sur le terminal. Dans nos travaux, nous ne pouvons évaluer l'impact de la technique des opérateurs communs que si nous considérons les différentes fonctions par lesquelles ils sont appelés. Comme l'opérateur commun est commun à différentes fonctions d'un même standard, une problématique majeure est à prendre en compte : l'organisation temporelle des différents opérateurs. De surcroit cette organisation doit être prise en compte dès le choix du meilleur set. En effet, une fonction pourra être exécutée par diverses combinaisons d'opérateurs communs. Ces opérateurs ayant des

temps d'exécution différents, la possibilité d'allouer plus ou moins d'opérateurs pour une fonction se répercutera sur le nombre d'opérateurs à implémenter et donc sur le choix du meilleur set. Ainsi, l'exploitation d'un graphe des opérateurs en vue de proposer une méthode générique de sélection ou de gestion ne sera pas abordé dans cette thèse. Néanmoins, nous explicitons dans la section suivante, les considérations relatives au cadencement des opérateurs et une solution d'implémentation que nous proposons.

2.4.1 Considération Simplifiée

Prendre en compte le scheduling dans la technique des opérateurs communs n'est pas une chose triviale car nous devons tenir compte du scheduling de tous les opérateurs les uns par rapport aux autres.

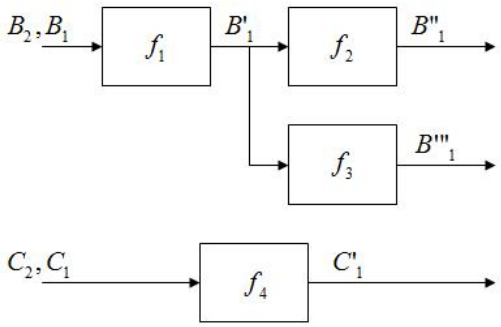

FIGURE 2.16 – Enchainement des traitements

Cadencement de Opérations et Dépendance des Données La Technique des opérateurs communs vise à se focaliser sur une granularité inférieure aux fonctions de télécommunications exécutées par le standard donné. Or, dans l'implémentation de nos opérateurs communs, nous ne pouvons pas pour autant occulter le cadencement des fonctions les unes par rapport aux autres et la dépendance des données vis à vis des fonctions qui se succèdent. La figure 2.16 illustre un possible enchainement de quatre fonctions nécessaires pour obtenir les données $B"_i$, $B'"_i$ et C'_i à partir des données B_i et C_i. Ici, $B"_i$ et $B'"_i$ ne peuvent s'obtenir qu'une fois que $f_1(B_i)$ c'est à dire B'_i est calculé. Nous parlons donc de dépendance des données entre l'entrée d'une fonction et la sortie d'une autre. Cette dépendance peut être d'autant plus compliquée à gérer que le type de données traitées n'est pas le même d'une fonction à l'autre (bit, bloc, flot, paquet, ...).
La littérature est riche d'exemples de dépendance de cette sorte. On trouve notamment le cas des dépendances fonctionnelles qui se représentent par graphe dans le cas du traitements des bases de données [ZWZ95] ou bien encore le graphe de fluence dans les domaines de l'électronique ou de l'automatique [RBR63] qui permettent de manière

générale par la règle de Mason de résoudre des systèmes différentiels [RBR63]. Dans notre cas d'étude la problématique est plus complexe que les exemples précédents étant donné que le cadencement des fonctions ne doit servir qu'à permettre le cadencement des opérateurs communs. Ce sont ces possibilités d'allocation que nous devons prendre en compte dans l'agencement des opérateurs mais aussi dans le choix des opérateurs. Nous représentons en figure 2.17 un possible graphe des opérateurs vis à vis des quatre fonctions précédentes.

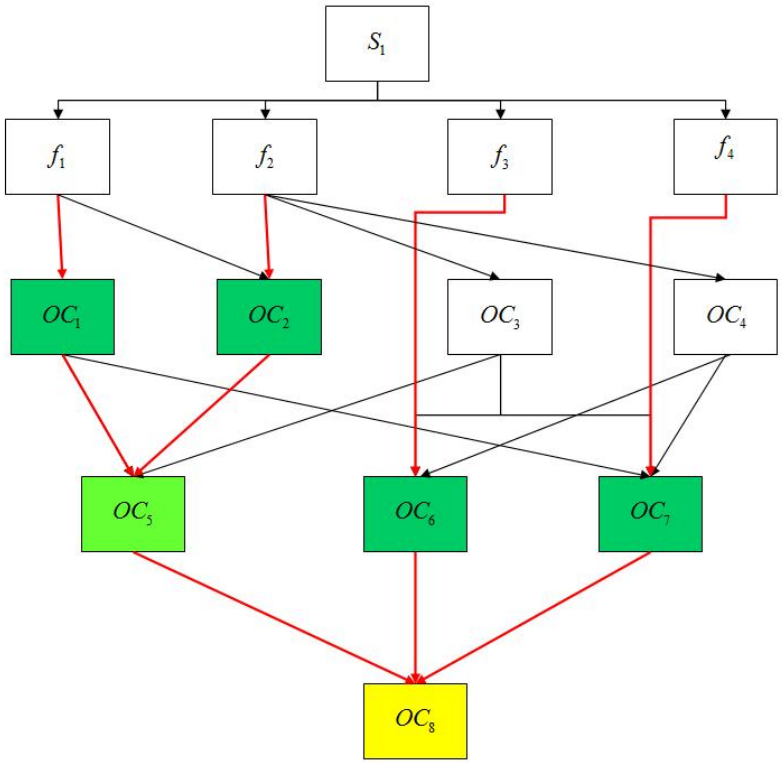

FIGURE 2.17 – Introduction de la donnée temporelle dans la représentation graphique

Ainsi, étant donné qu'un standard sera exécuté à partir d'un set limité d'opérateurs communs, ces derniers pourront être utilisés pour différentes opérations qui s'enchainent ou qui s'exécutent en même temps. Par conséquent, les opérateurs communs seront interdépendants dans la mesure où le résultat de l'un pourra constituer l'entrée d'un autre et ainsi de suite tout au long de la chaine.

De plus, chaque standard dans une spécification bien précise d'un mode définira un débit de sortie de chaîne d'émission. Ce débit sera lié à une fréquence de sortie F_{Fs} du système et à une période d'horloge T_{Fs}. Or dans la conception d'une chaine d'émission mais aussi de réception, l'ensemble des éléments de la chaîne ne sont pas tous cadencés via une même horloge. En effet, le débit du système n'affecte uniquement que la sortie

finale des données. Tout au long de la chaine, des fréquences intermédiaires sont utilisées par le biais de diviseurs ou multiplicateurs de fréquences afin de permettre à chaque opération d'être effectués par rapport aux types de données qu'elle traite (bit, bloc, flot ...) et à la plus ou moins grande complexité de ces traitements.

De plus, un opérateur commun vise à être utilisé par une fonction, or ils ne réalisent pas l'intégralité de cette fonction, il peut être utilisé en combinaisons avec d'autres opérateurs communs et/ou des modules d'interfacages ou de gestion des données. Par conséquent, un opérateur commun sera dépendant du sous système que représente la fonctionnalité qu'il contribue à exécuter. Celle-ci sera cadencée, à une fréquence spécifique F_i et l'opérateur commun à une fréquence équivalente ou multiple de F_i. L'intérêt de l'opérateur commun est de pouvoir être utilisé pour l'exécution d'un maximum de fonction au sens de la définition des ces fonctionnalités mais aussi au sens de l'exécution. En effet, un opérateur commun se définit comme possédant un temps d'exécution T_e, c'est à dire une fréquence d'exécution F_e maximale à laquelle il est capable d'être exécuté correctement. Ainsi, l'avantage pour une instance d'un opérateur commun est d'être utilisée via cette fréquence maximale afin de réaliser autant que possible le maximum des opérations de la chaine de traitement. Evidemment, ce type d'exécution nécessitera des registres de stockages de données pour synchroniser les résultats de chaque exécution de l'OC avec le reste de la fonction appelante et du terminal.

Cadencement et Graphe des Opérateurs Communs En nous référant aux figures 2.16 et 2.17, un opérateur commun va être donc affecté à une ou plusieurs fonctions de la chaine traitement. L'attribution de ces allocations qui résultent du processus est hors de propos ici, nous nous intéressons à l'exécution d'un opérateur commun qui est déjà affecté.

Pour tenir compte de l'utilisation du même opérateur par différentes opérations f_i, nous définissons des temps d'exécution $T_{f,i}$ pour chaque opération à exécuter. Si maintenant, nous nous rapportons à une chaine de traitement où un même opérateur peut être utilisé par plusieurs f_i alors nous définissons selon une échelle de temps, deux autres données liées à f_i :

- Un temps $T_{d,i}$. Instant où doit commencer l'opération sur l'axe des temps.
- Un temps $T_{fin,i}$. Instant où doit se terminer l'opération ($T_{fin,i} = T_{d,i} + T_{f,i}$).

Maintenant, pour tenir compte du besoin réel d'un opérateur commun durant ces temps $t_{f,i}$, nous devons intégrer, cette donnée temporelle dans le graphe des opérateurs communs. En effet, pour chaque liaison dans le graphe, les deux paramètres N_c et N_T sont à considérer. Ainsi, un standard S_j répercutera les différents temps $T_{f,i}$ sur les opérateurs communs constituants les points d'entrée du graphe des opérateurs. Ces données se propageront d'un opérateur appelant vers l'opérateur appelé, constituant à chaque fois un ensemble de paramètres de définitions P_d, à transmettre : ($T_{f,i}$, N_T, N_c). Ainsi, un opérateur appelé pourra recevoir plusieurs P_d et en devenant opérateur appelant ces mêmes P_d, seront transmis à un autre opérateur appelé de granularité inférieure. Nous pouvons illustrer les définitions précédentes avec la figure 2.18.

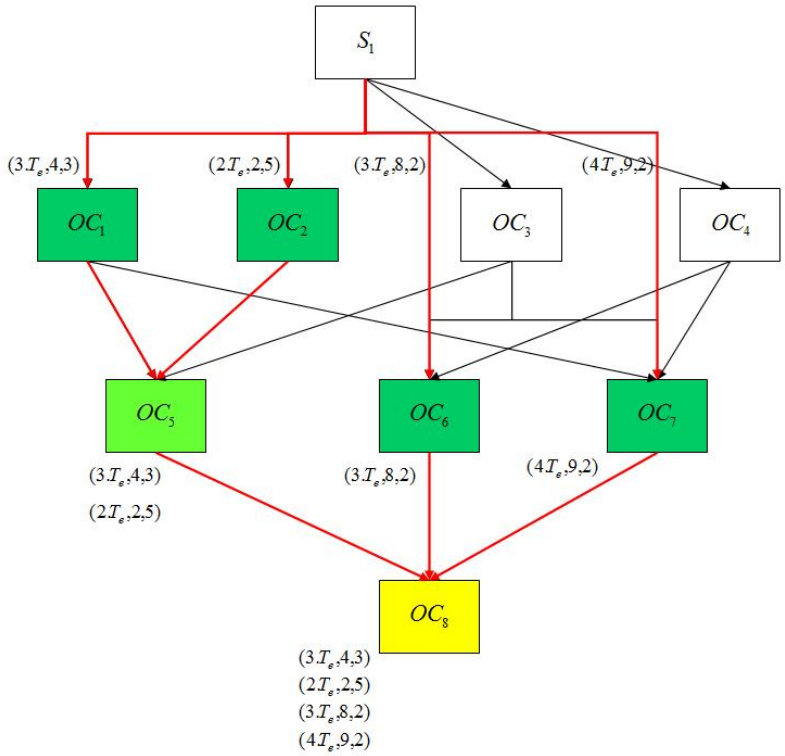

FIGURE 2.18 – Introduction de la donnée temporelle dans la représentation graphique

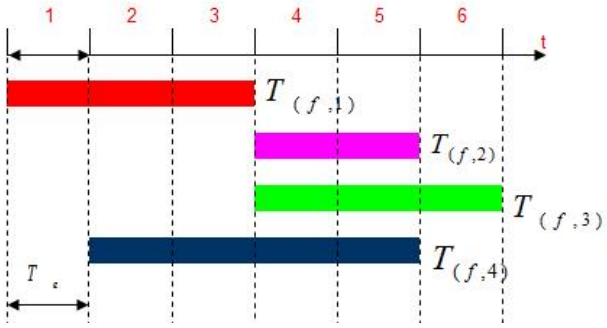

FIGURE 2.19 – Introduction de la donnée temporelle

Instances Physiques et Virtuelles Les temps $T_{f,i}$ peuvent être vu comme le temps d'horloge de l'opérateur qui est remplacé par l'opérateur commun vis à vis de l'ensemble de l'architecture qui le met en oeuvre. Ainsi, l'opérateur commun pourra dans ce temps d'horloge être exécuté $N_{execution}$=E($\frac{T_{f,i}}{T_e}$) fois. Nous reprenons l'exemple précédent, où l'opérateur commun OC_8 sera utilisé par quatre f_i (2.19). Nous construisons pour chaque opérateur commun une échelle de temps basé sur le temps d'exécution T_e de l'opérateur. De plus, nous faisons deux hypothèses dans la construction de cette schématique.

- D'abord, comme l'échelle des temps est celle de T_e, nous supposons que dans l'ordre chronologique, l'origine de l'axe des temps commence avec le premier $T_{f,i}$.
- Ensuite, comme les $T_{f,i}$ peuvent ne pas s'étendre exactement sur un multiple de T_e, nous considérons seulement sur le schéma le temps $T_{f,i}$ qui recouvre un nombre entier de T_e. Ainsi, l'exemple précédent, aboutit à la représentation de la figure 2.19.

Pour chaque instance de l'opérateur physiquement implémentée alors "chaque T_e" représente une instance virtuelle. Ainsi, dans l'exemple de la figure 2.19, l'intérêt est de trouver une allocation spécifique qui minimise l'utilisation des mêmes instances physiques en répartissant la charge de travaille sur les instances virtuelles.

A partir de la figure 2.19, nous pouvons distinctement illustré l'intérêt de l'opérateur commun dans l'exécution d'une seule et même chaine de traitement pour un standard donné. Comme les opérations, f_2 et f_3 s'exécute après f_1 alors les instances de OC_8 affiliées à f_1 dans un premier temps peuvent être ensuite allouées à f_2 et f_3. De plus, comme f_4 s'exécute en parallèle de f_1, f_2 et f_3, alors il est envisageable d'allouer en alternance les opérateurs. En effet, pour chaque instance de l'opérateur physiquement implémentée alors "chaque T_e" représente une instance virtuelle. En effet, si nous négligeons le temps de reconfiguration de l'opérateur commun alors le nombre d'instances de OC_k à implémenter pour réaliser une f_i est donnée par $N_{b,i}$. Afin de tenir compte que chaque opération demandera un nombre d'instances physiques minimum (N_c) et un nombre d'appels (N_T) de celles-ci, tel que décrit dans la définition de l'opérateur commun en 2.3.2, chaque temps $T_{f,i}$ sera associé à un couple ($N_{c,i}$,$N_{T,i}$).

$$N_{b,i} = E(\frac{N_{T,i}.T_e}{T_{f,i}}).N_{c,i} \tag{2.13}$$

Pour illustrer ce principe, nous complétons en figure 2.20, le cas de la figure 2.19 en associant (arbitrairement) à chaque $T_{f,i}$ les deux paramètres N_T et N_c et calculons les $N_{b,i}$ associé :

- ($T_{f,1} = 3.T_e$, $N_{T,1} = 4$, $N_{c,1} = 3$) donne $N_{b,1} = 6$.
- ($T_{f,2} = 2.T_e$, $N_{T,2} = 2$, $N_{c,2} = 5$) donne $N_{b,2} = 6$.
- ($T_{f,3} = 3.T_e$, $N_{T,3} = 8$, $N_{c,3} = 2$) donne $N_{b,3} = 5$.
- ($T_{f,4} = 4.T_e$, $N_{T,4} = 9$, $N_{c,4} = 2$) donne $N_{b,4} = 6$.

C'est à dire que nous devrions implémenter au total un nombre de $N_{imp} = \sum N_{b,i}$ soit ici, 23 instances du même opérateur commun. Nous proposons une réprésentation en figure 2.20. Chaque instance d'un opérateur commun est représentée par un rectangle noir en traits fins. Pour chaque f_i, nous représentons en traits gras les groupes de $N_{c,i}$

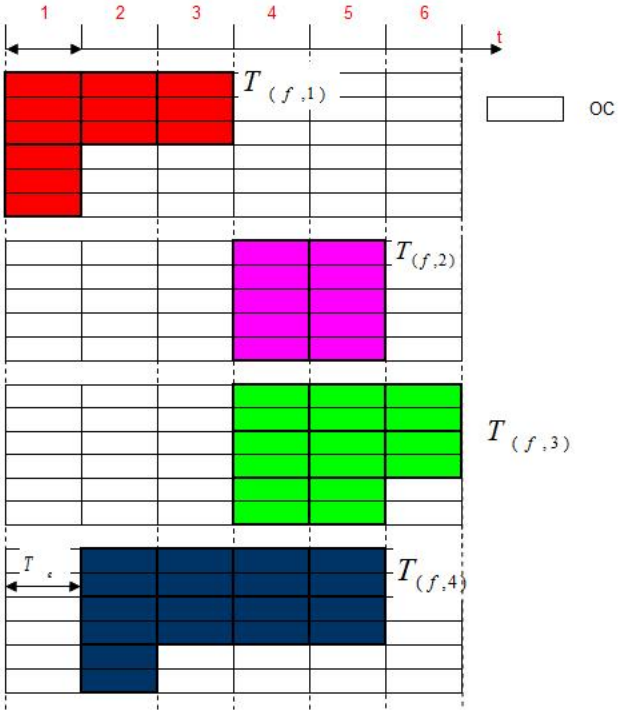

FIGURE 2.20 – Introduction de la donnée temporelle - Instances de l'opérateur commun

opérateurs qui représentent les opérateurs qui doivent être implémentés en simultanée par construction. Ainsi, pour chaque f_i, identifiée par une couleur spécifique, il existe des opérateurs non utilisés (non colorés). L'interêt de la technique des opérateurs communs et de diminuer ces instances virtuelles non allouées.

2.4.2 Considération d'un Flux

La Technique des opérateurs communs définie l'identification d'un graphe d'opérateurs communs et le choix du meilleur set via un algorithme d'optimisation de graphe. Dans ce contexte, nous ne voulons pas imposer de contraintes au designer dans l'agencement des opérateurs. Ainsi, celui-ci dépendra exclusivement de la manière dont le designer défini son terminal, agence et cadence ces opérations. A partir de ce point, l'exemple d'agencement proposé en 2.19 semble restreint quant aux cas de figures qu'ils recouvrent. En effet, la figure 2.19 représente un agencement temporel possible vis à vis de la chaine de traitement présenté en figure 2.16. Or en figure 2.19, nous considérons l'exécution de l'opérateur commun pour la succession des opérations f_1 puis f_2 et f_3 avec en parallèle f_4. L'exécution du cas considéré en 2.4.1 suppose que l'opérateur OC_8 traite les données entrantes pour f_1, puis des données entrantes pour f_2 et f_3 en considérant que les sorties du premier traitement réalisé par f_1 soient les entrée de f_2 et f_3. Cela signifie que dans ce cas d'étude, une deuxième exécution de OC_8 pour un autre traitement f_1 ne pourra se faire que lorsque l'utilisation de OC_8 par f_2 et f_3 sera terminée. Ce typê d'exécution parait restrictif.

En effet, en se référant à la figure 2.16, si nous considérons le bloc de données B_1 en entrée de f_1 et que f_2 et f_3 ont comme entrée le bloc de données B'_1 résultant du traitement de B_1 par f_1 alors le bloc de données B_2 ne pourra être traité qu'une fois B'_1 traité par f_3. De plus, comme f_4 est indépendant de l'enchainement f_1, f_2 et f_3, le bloc de données C_2 à l'entrée de f_4, n'a pas besoin d'attendre la fin de f_3 pour être traité et peut directement commencer son traitement par f_4, une fois le traitement de C_1 réalisé. Nous ne pouvons donc ne pas uniquement nous focaliser sur la schématique de la figure 2.19, mais introduire l'enchainement des différents traitement dans le temps. Nous illustrons différentes configurations en figure 2.21 et en figure 2.22.

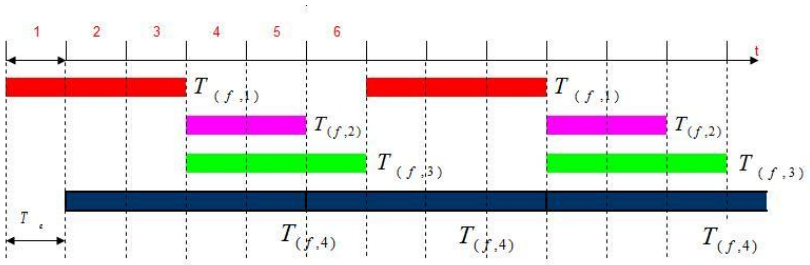

FIGURE 2.21 – Enchainement des traitements - Cas A

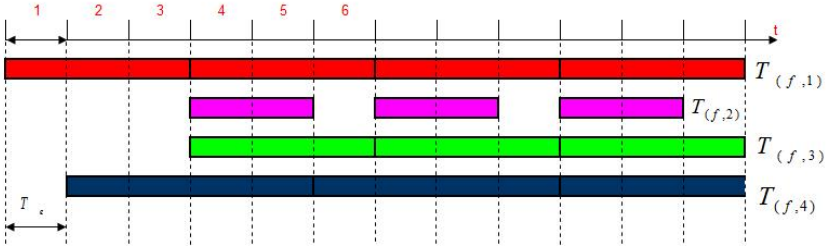

FIGURE 2.22 – Enchainement des traitements - Cas B

En figure 2.21 (cas (A)), nous considérons que chaque bloc B_i doit effecteur l'enchainement des traitements f_1, f_2 et f_3 pour que le le bloc B_{i+1} soit a son tour traité. En parallèle, les bloc de données C_i sont traités à la suite par f_4.

En figure 2.22 (cas (B)), nous considérons que chaque bloc B_i est traité par f_1 sans attendre l'exécution de f_2 et f_3. Et que le bloc résultat B'_i peut être stocké en mémoire si besoin pour être traité par f_2 et f_3. En parallèle, les blocs de données C_i sont traités à la suite par f_4. Ce dernier cas est non seulement adapté pour un traitement des données par blocs mais aussi sur un flot de données de taille non définie.

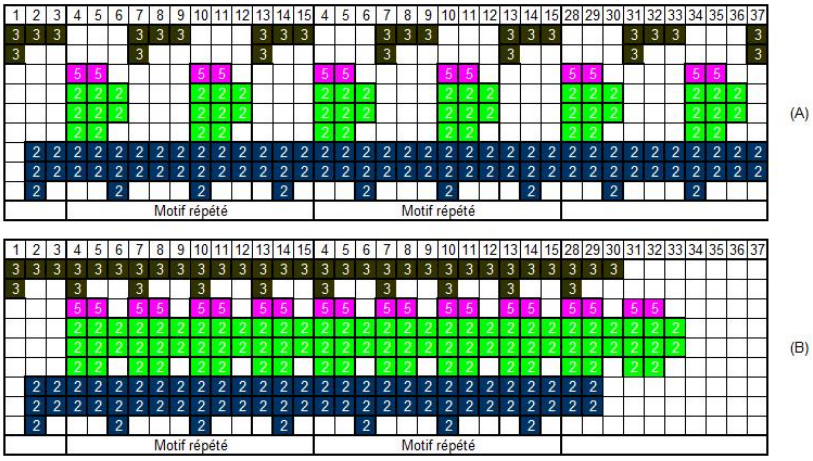

FIGURE 2.23 – Enchainement des traitements - Cas A et B - Motif Repeté

Ainsi, selon le cas, le besoin en instances du même opérateur communs va varier et les possibilités de réutilisation vont diminuer. Comme cette organisation est laissée au choix du designer, nous devons nous placer dans un cadre plus général que celui proposé en 2.4.1. Pour représenter le besoin en opérateur commun et les allocations de chaque instance pour une opération nous proposons la figure 2.23. L'allocation de

différentes opérations sur différentes instances virtuelles du même opérateur commun reste possible. En effet, en figure 2.23, nous représentons en abscisse l'échelle des T_e. Pour chaque f_i, nous représentons un groupe de $N_{c,i}$ opérateurs par un carré du quadrillage. Chaque "carré" est identifié par une couleur (celle de la f_i correspondante) et un nombre (celui du $N_{c,i}$ correspondant). Ainsi, selon le cas, il est donc toujours possible d'allouer les mêmes instances d'un opérateur pour des opérations distinctes lors de T_e différents. Cette gestion des opérateurs sera d'autant plus complexe selon l'exploitation des canaux de communications, c'est à dire dans la mise en oeuvre de liaisons Simplex et Duplex. La mise en place de l'une ou l'autre de ces liaisons nécessitera la prise en compte de l'opérateur commun pour des opérations simultanément à la réception et à l'émission. Nous pouvons différencier (figure 2.24) :

- Les Canaux simplex qui transporte l'information dans un seul sens
- Les Canaux half-duplex permet le transport d'information dans les deux directions mais pas simultanément.
- Les Canaux full-duplex pour lesquels l'information est transportée simultanément dans chaque sens.

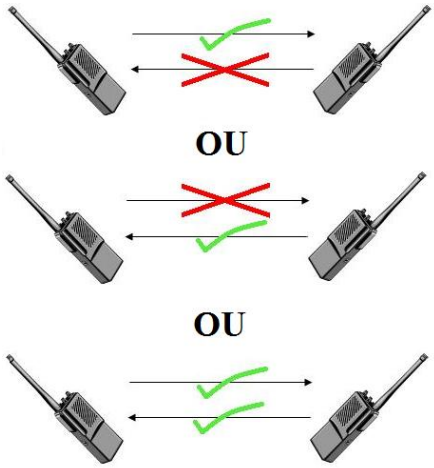

FIGURE 2.24 – Principe des Communications Duplex

2.4.3 Banc d'opérateurs Communs (BOC)

A partir des cas de figures précédents, nous pouvons appliquer un algorithme d'allocation des instances virtuelles sur les instances physiques que nous exposons dans l'Annexe C et obtenir les nouvelles allocations des figures 2.26 et 2.25. Nous illustrons, par ces exemples, la gestion nécessaire pour un même opérateur commun, utilisé par diverses opérations d'un même standard. De part la répétition des mêmes traitements

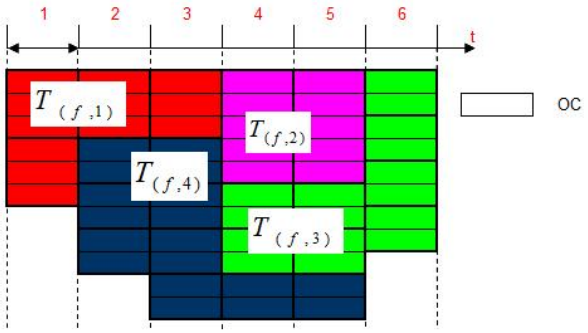

FIGURE 2.25 – Amélioration de la réallocation

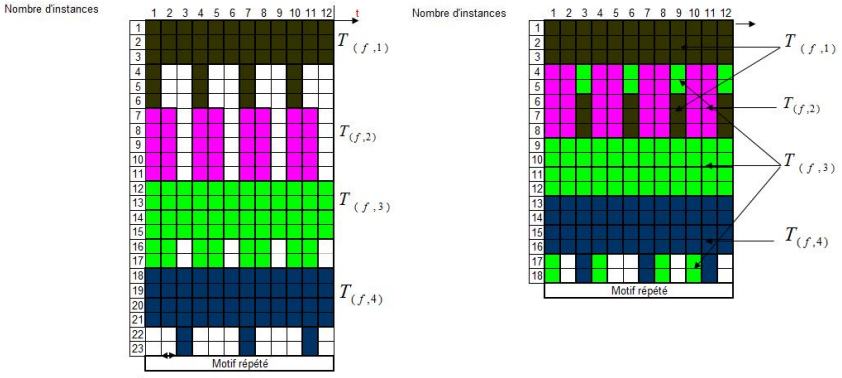

FIGURE 2.26 – Enchainement des traitements - B - Motif Repeté et Réallocation

sur des données différentes, nous avons identifié une fenêtre de temps de taille T_F (Annexe C) qui se répète dans laquelle le besoin en opérateurs est constant. Ces exemples mettent en lumière tout l'intérêt de l'opérateur commun en permettant une allocation de différentes opérations et en économisant les instances à implémenter. Cependant, cette approche ne peut se faire si et seulement si un ensemble de techniques de gestion est implémenté afin d'assurer le bon fonctionnement et les allocations adéquates entre opérations et opérateurs. En se référant aux travaux de Wang [WLP06] relatif à un décodage MIMO utilisant un CORDIC, les organes de contrôle pour ce seul décodage dans le cas où le nombre d'opérateurs CORDIC était minimisé s'avéraient engendrer une complexité matérielle supérieure aux opérateurs communs eux même. De plus, l'allocation est certes simplifiée par la répétition d'un même motif pour un opérateur donné, il n'en reste pas moins que pour N opérateurs communs, nous aurons N gestions différentes par standard à gérer ce qui fait N.P procédures à prendre en compte pour le terminal multistandard (Pour rappel, P est le nombre de standard). En se référant à l'exemple de la figure 2.26, nous pouvons noter que pour seulement 4 opérations demandant le même opérateur, la gestion de celui-ci n'est pas triviale. L'implémentation de cette gestion aura un coût à prendre en compte par rapport au coût total du terminal qui est jusqu'ici ignoré dans les algorithmes de sélection. Deplus, en définissant un terminal utilisant les OC à leurs fréquences maximales, nous limitons la possibilité d'évolution d'un terminal donné quant à la réutilisation de l'OC par un nouveau standard en diminuant les instances virtuelles. L'allocation des instances virtuelles étant déjà optimisée, elles peuvent ne pas être en nombre suffisant pour une nouvelle norme.

Ainsi, pour palier à cette gestion qui "complique" l'implémentation de l'opérateur commun, nous décidons de proposer une méthode d'implémentation appelée, le Banc d'Opérateurs Communs. En effet, l'idée première dans le concept de l'opérateur commun est de développer un opérateur qui peut être localement plus complexe que les opérateurs qu'ils visent à remplacer mais qui de part son utilisation à plusieurs reprises au seins de plusieurs standards permet une optimisation globale du système. Nous conservons cette idée mais au lieu de l'appliquer aux opérateurs nous décidons de l'appliquer directement à chaque standard. C'est à dire que nous proposons de développer un terminal qui par comparaison vis à vis de chaque standard pris individuellement pourra être plus complexe selon un critère donné mais qui dans le cas d'un terminal multistandard pourra s'avérer optimisé selon le même critère.

De manière à bien discriminer les tenants et les aboutissants de notre méthode, nous distinguons deux organisations des OC ; un organisation dite " locale " et une seconde qualifiée de " Globale ". L'organisation locale correspond à l'agencement d'un opérateur par une opération donnée. Une fois paramétré un opérateur, devient le "réplica" de la structure à remplacer, et à l'instar de la structure originale à laquelle il se substitue, l'OC peut faire l'objet d'un agencement temporel spécifique à une fonction donnée. Un exemple de cette organisation " locale " peut être illustrée par l'opérateur " papillon " (Butterfly) de la FFT [AGLP06]. L'organisation " globale " est une organisation

introduite par la technique des opérateurs communs. Celle-ci qualifie l'utilisation en parallèle d'un même OC par des fonctions réalisées en simultané, c'est cette organisation que nous traitons dans les sections précédentes 2.4. Cette dernière organisation globale peut devenir ardue dans l'implémentation de notre Technique. Pour y remédier, nous proposons d'implémenter les OC en Bancs. Pour chaque type d'OC, nous implémentons une quantité fixe du même OC_k pour former un groupe d'opérateurs communs OG_k. Nous implémentons le nombre suffisant d'OC_k pour que chaque opération f_i qui requiert leurs utilisations, possède ses OC dédiés. Un BOC sera la concaténation de plusieurs OG_k. Ainsi, pour un standard S_j, une opération f_i n'aura pas à recourir à l'utilisation d'opérateurs réservés pour une autre fonction f_j (i $\neq$ j). Par rapport aux exemples précédents, et plus précisément à la figure 2.26, nous ne cherchons plus à optimiser l'allocation des ressources pour un standard donné mais par rapport au terminal multistandard dans son ensemble avec l'avantage d'une implémentation plus simple afin d'éviter la gestion complexe des opérateurs.

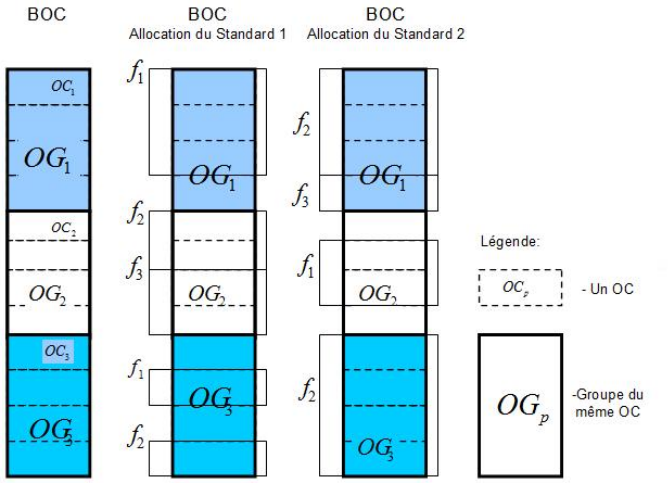

FIGURE 2.27 – Banc d'Opérateurs Communs (BOC)

Le principe du BOC répond à la problématique du "Scheduling" en dupliquant les OC. Comme une fonction peut être exécutée par des combinaisons distinctes d'OC, différents BOC sont possibles dont certains pourront minimiser des critères spécifiques. Ainsi, en Annexe B, nous définissons l'équation du terminal SDR :

$$C_{SDR} = min_{\chi}(\sum_{k=1}^{N} max_{j,\chi}(N_{imp,j,c,k}.C(OC_k))) \tag{2.14}$$

Nous renvoyons le lecteur à l'Annexe B pour les explications relatives à ce résultat. Ici χ représente l'ensemble des sets hybrides possibles, c'est à dire un set obtenu à partir

d'un ensemble de standards S_j, chacun utilisant une combinaison spécifique c des OC. $N_{imp,j,c,k}$ représente le nombre d'OC_k d'un $S(j,c)$. Dans le cas d'un BOC, ce nombre devient égal à la somme du nombre d'OC_k, $N_{b,i}$ requis par chaque fonction f_i.

$$N_{imp,j,c,,k} = \sum_i N_{b,i,j,c,k} \tag{2.15}$$

Il est à noter que pour chaque standard l'allocation des entrées/sorties de chaque fonction sur le banc d'opérateurs sera spécifique. Cette organisation en banc permet ainsi, d'éviter la dite organisation " globale " et d'obtenir une gestion facilité des opérateurs, limitée à l'organisation locale classique. Nous pourrons ainsi allouer les différentes opérations pour chaque standard sur les opérateurs communs. Pour chaque standard, le paramétrage consistera à

- Reconfigurer les liaisons entre les entrées/sorties des dites fonctions et les entrées/sorties de chaque type d'opérateurs regroupés en banc.
- Reparamétrer chaque opérateur une seule fois pour la fonction qui va l'appeler.

2.5 Conclusion

Nous venons de définir la technique des opérateurs communs. Nous spécifions plus précisément, les considérations des opérateurs communs matériels que nous cherchons à construire et à tester dans nos recherches. La gestion des opérateurs communs ne sera pas au centre de cette étude mais ne peut pourtant pas être occulté dans l'application de la technique, ce qui nous a amené à définir une alternative architecturale par la présentation du Banc d'Opérateurs Communs. Dans la suite de ce rapport, nous présentons une classification des éléments de la chaine de traitement en quatre parties selon les architectures mises en oeuvre. Nous exposons, deux grands types de classes ; la première factorise les système de treillis/papillons et la seconde met en oeuvre les registres à décalages.

Chapitre 3

Factorisation des Structures de la Chaine de Traitement

Sommaire

3.1 Introduction

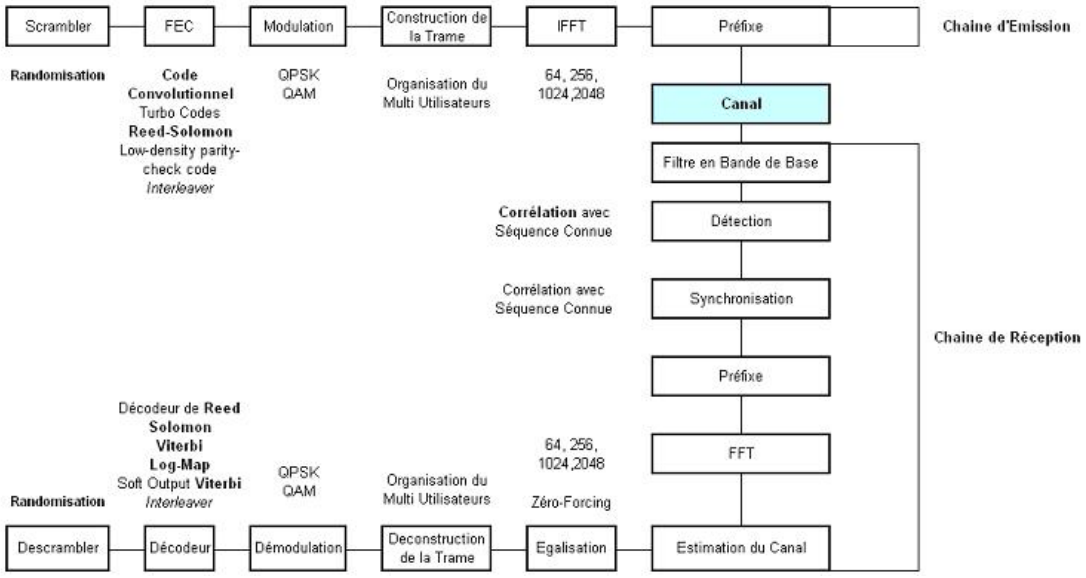

FIGURE 3.1 – Identification des Fonctions de la Chaine de Traitement

Concevoir un émetteur/récepteur multistandard nécessite de comprendre les différentes opérations requises par les chaines de traitements des standards. Nous représentons une chaîne de transmission en figure 3.1. De manière générale, nous pouvons distinguer des fonctionnalités qui sont systématiquement les même que ce soit à l'émission ou à la réception. Dans l'optique de définir des opérateurs communs, nous devons nous attacher à étudier les architectures et les algorithmes mis en jeux par ces fonctions afin d'identifier ou de construire les possibles OC. Pour ce faire, nous proposons une classification des différentes fonctions de la chaine de traitement par rapport aux types de structure qu'elles mettent en oeuvre et aux données qu'elles traitent. Cette classification (figure 3.2) repose non pas sur le rôle des fonctions dans la chaine de traitement mais sur les éléments qui les constituent :

1. Les fonctions utilisant les opérations de multiplications et d'additions à l'instar de la FFT, des FIRs, de la corrélation...
2. Les fonctions réalisant des opérations de décalages et des opérations logiques telles que le codage convolutionel, la randomisation, le turbo codage...
3. Les fonctions réalisant des opérations sur la localisation des bits comme " l'interleaving " ou le " puncturing ".
4. Les fonctions réalisant des associations de données comme les différentes modulations.

En considérant, la classification précédente, nous présentons dans ce chapitre, les principales structures présentes dans les classes 1 et 2.

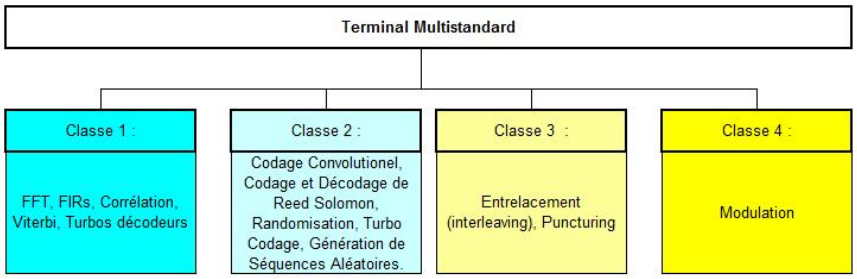

FIGURE 3.2 – Classification Architecturalle des Opérations

3.2 Factorisation des Fonctions Utilisant des Registres à Décalages

Nous présentons en Annexe D, le principe des registres à décalages et les deux familles de LFSR, Galois et Fibonacci. Nous étudions, ici, les structures usuelles utilisées dans les standards de télécommunications. A chaque fois, nous proposons un récapitulatif des besoins spécifiques des normes que nous considérons en chapitre 5 lors de l'implémentation à savoir le IEEE 802.16, le IEEE 802.11 et le 3GPP LTE.

3.2.1 Embrouillage

3.2.1.1 Préambule

L'embrouillage est une opération destinée à transformer un signal numérique en un signal numérique aléatoire ou pseudo aléatoire, de même signification et de même débit binaire, en vue d'augmenter la protection des données lors de leur transmission. Dans les réseaux de télécommunications, et les normes liées, le module d'embrouillage (scrambler) se constitue d'un générateur de séquences aléatoires binaires et d'un mélangeur. En travaillant au niveau des données binaires le mélangeur est traditionnellement un additionner binaire modulo 2 (XOR) qui permet de "mélanger" bit par bit, le signal de données à la séquence aléatoire. A la réception, le signal reçu est "re-mélanger" avec la même séquence aléatoire afin de retrouver les données de départ. En effet, comme la fonction XOR est sa propre inverse, alors en considérant le signal de données E, et PN la séquence pseudo aléatoire alors à la réception :

$$(E\ XOR\ PN)\ XOR\ PN = E \tag{3.1}$$

Le scrambler dépend donc du générateur de séquences aléatoires. Vis à vis des trois standards considérés dans cette étude ; IEEE 802.11, IEEE 802.16 et 3GPP LTE, nous résumons toutes les opérations de génération de séquences aléatoires dans ces standards

(figure 3.3) en spécifiant le type de séquence produite comme le type de structure les réalisant. Nous présentons ces différentes architectures dans la suite de cette section.

Générateurs de Séquences Aléatoires et Scramblers (IEEE 802.11, IEEE 802.16, 3GPP LTE)					
Standard	Polynôme	Type de Séquence émise	Type de Structure Requise	Utilisation	Remarque
IEEE 802.11	$x^7 + x^4 + 1$	PN-Séquence	LFG	Embrouillage et Désembrouillage des données pour les type de modulation FHSS et OFDM	Emission et Réception
IEEE 802.11	$x^7 + x^4 + 1$	PN-Séquence	LFSR	Embrouillage des données pour les types de modulations DSSS et High Rate DSSS	Uniquement à l'émission. Pas de séquence Initiale, insertion des données directement
IEEE 802.11	$x^7 + x^4 + 1$	-	*Registre à Décalage*	*Désembrouillage des données pour les types de modulations DSSS et High Rate DSSS*	*Uniquement à la Réception.*
3GPP LTE	$x^{25} + x^3 + x^2 + x + 1$	Gold-Séquence	Deux FILTRES IIR interconnectés	Embrouillage et Désembrouillage des données pour la voix montante	Uniquement pour l'Uplink
3GPP LTE	$x^{18} + x^{10} + x^7 + x^5 + 1$	Gold-Séquence	Deux FILTRES IIR interconnectés	Embrouillage et Désembrouillage des données pour la voix descendante	Uniquement pour le Downlink
IEEE 802.16	$x^{15} + x^{14} + 1$	PN-Séquence	LFG	Embrouillage et Désembrouillage des données pour les différentes modes Single Carrier, Single Carrier Access, OFDM et OFDMA.	Emission et Réception
IEEE 802.16	$x^{22} + x^{21} + 1$	PN-Séquence	LFG	Génération des séquences pour la construction de la modulation « Spread BPSK » dans le cas Single Carrier et Single Carrier Access	Emission et Réception
IEEE 802.16	$x^{11} + x^9 + 1$	PN-Séquence	LFG	Génération des séquences pour la construction des pilotes des sous porteuses dans le cas de l'OFDM et OFDMA.	Emission Séquences Initiales différentes pour Uplink et Downlink

FIGURE 3.3 – Inventaire des Générateurs de Séquences Alétoires Requis

Un générateur de nombres pseudo-aléatoires est un algorithme qui génère une séquence de nombres présentant des propriétés du hasard telles que "l'impossibilité de prévoir le prochain nombre tiré" et le respect des fréquences. Il existe plusieurs types de générateurs de séquence aléatoires ; (1) générateurs congruentiels, (2) générateurs de Fibonacci Décalé (Lagged Fibonacci Generator - LFG) et (3) les registres à décalages à rétroactions linéaires (LFSR).

3.2.1.2 Congruence linéaire

Le but de cette méthode est de créer une suite de nombres de manière aléatoire ou plutôt avec aussi peu de régularité que possible. Pour cela on choisit une valeur m qui sera la valeur maximale des entiers considérés, une "racine" qui correspond au premier terme de la suite et deux entiers a et b tels que si on nomme u_n le nième terme de la suite et u_{n+1} le suivant est donné par la relation :

$$u_{n+1} = a.u_n + b \ (mod\ m). \tag{3.2}$$

Les entiers obtenus sont alors compris entre 0 et m - 1, cependant dès qu'un nombre apparait pour la deuxième fois la suite entre dans une boucle, chaque terme n'étant déterminé que par le précédent. Ainsi la taille maximale de la boucle ou période maximale est de m. Donald E Knuth [Knu74] définit le protocole de sélection de a, b et m. Dans un premier temps, pour un m choisit, on peut alors prendre b = 0 pour faciliter l'étude et ensuite choisir a tel que les restes des divisions euclidiennes par m des puissances successives de a ne soient égales à a que le plus tard possible. Avec $u_{n+1} = a.u_n$, le

n-ième terme de la suite noté u_n s'écrit : u_n = r.a^n (mod m), r étant la racine de la suite. On dit que a est primitif modulo m si la plus petite valeur de t pour laquelle on ait a^t = a (mod m) soit m. Ces critères ont été définis par Donald E Knuth [Knu74]. Dans un second temps, pour b non nul, il faut tout d'abord que m et b soient premiers entre eux. Ensuite a-1 doit être un multiple de p pour tout p diviseur premier de b et un multiple de q pour tout q diviseur premier de m. Par exemple, a-1 doit être un multiple de 4 si b et/ou m est multiple de 4. De cette manière, on obtient un générateur à une racine de période m, on peut alors choisir la racine au hasard entre 0 et m - 1 sans que cela influence la période.

3.2.1.3 Racines nombreuses et générateurs de Fibonacci Décalé (LFG)

Au lieu de définir le terme de rang n de la suite uniquement par celui qui précède on utilise les deux termes précédents ou même des termes encore plus éloignés. Par exemple, soit a, b, c, p et q des entiers et 0<j<k :

$$\begin{aligned} u_n &= a.u_{n-1} + b.u_{n-2} + c \\ u_n &= u_{n-j} + u_{n-k} \end{aligned} \tag{3.3}$$

Les LFG sont des variantes des générateurs congruentiels qui se basent initialement sur la récurrence de Fibonacci avec comme objectif d'augmenter la période de la suite produite. La suite de Fibonacci relie la valeur n d'une suite à ces deux prédécesseurs dans la suite tel que : $u_n = u_{n-1} + u_{n-2}$. Le LFG est la généralisation d'un telle structure pour tout j et k tel que : $u_n = u_{n-j} \diamond u_{n-k}$, où $\diamond$ peut représenter une addition, une multiplication ou une addition modulo 2 (XOR). Ces trois possibilités distinguent trois types de LFG, Additive LFG (ALFG), Multiplicative LFG (MLFG) ou Two-Tap Generalized LFSR (GFSR).

Avec cette méthode, la série ne bouclera que si les deux valeurs au rang n-p et n-q se répètent simultanément. Pour le même m (m=2^M), on obtient des périodes plus grandes de (2^k - 1).2^{M-1} pour le ALFG, (2^k-1) pour le GFSR et (2^k - 1).2^{M-3} pour le MLGF. La période maximale que l'on peut viser est de m^2 - 1. Ainsi, il n'est plus nécessaire de choisir un m grand puisqu'une suite pour laquelle u_n est définit par u_{n-1}, u_{n-2},etc jusque u_{n-k} aura une période maximale de $m^k - 1$.

3.2.1.4 Linear Feedback Shift Register (LFSR)

En prenant, m=2, les seules valeurs possibles sont 0 et 1. On retrouve une suite binaire. En appliquant les récurrences précédentes entre u_n, u_{n-p} et u_{n-q}, pour p=1 et q=2, on retrouve la suite classique de Fibonacci et le GFSR précédemment cité. Pour une suite nécessitant la sommation de k (u_{n-p}), on retrouve une suite de Fibonacci généralisée qui donne son nom au LFSR de Fibonacci présenté précédemment. La figure 3.4 illustre le scrambleur du 802.16, basé sur un LFSR de Fibonacci construit avec le polynôme générateur $g_1(x) = x^7 + x^4 + 1$.

L'intérêt d'utiliser une forme généralisée d'un LFSR est sa capacité à créer des séquences aléatoires spécifiques. En effet, un LFSR de taille r possède un polynôme

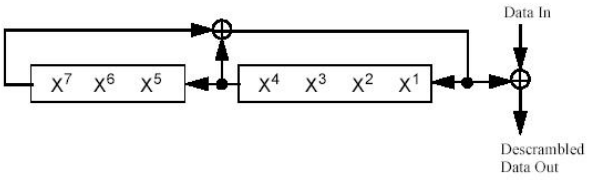

FIGURE 3.4 – Architecture du Scrambler du 802.11 mode OFDM

primitif à partir duquel il est possible de créer une m-séquence, c'est à dire une séquence aléatoire de période $2^r - 1$. Golomb [Gol82] a démontré les trois propriétés d'une m-séquence. Ainsi, une m-séquence a comme particularité principale de présenter une fonction d'autocorrélation proche de la fonction de kronecker, permettant ainsi de mesurer les réponses impulsionnelles.

En utilisant, deux m-séquences de même longueurs maximales et en les combinant par le biais d'un additionner modulo 2 (figure 3.5), la séquence obtenue est appelée "Gold Sequence". Les séquences de Gold présentent des propriétés de cross-corrélation meilleure que les m-séquences en présentant seulement trois valeurs possibles [AQD09].

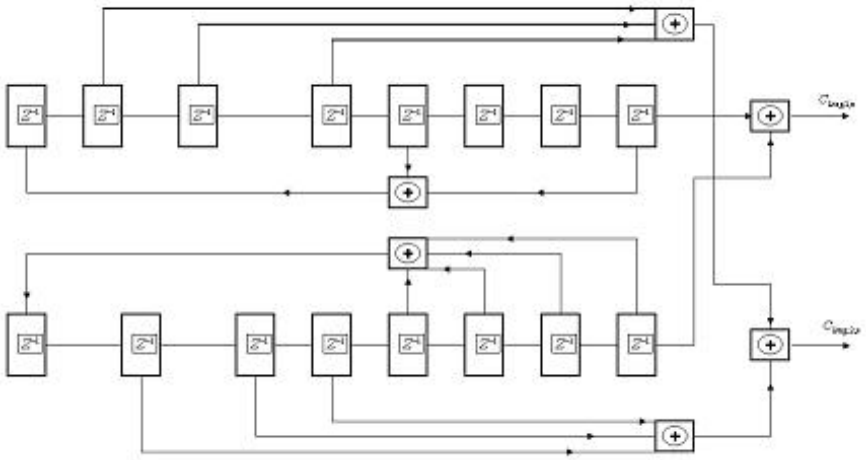

FIGURE 3.5 – Scrambler 3 GPP LTE - Gold Sequence.

3.2.1.5 LFSR et synchronisme

Les méthodes d'embrouillage présentées précédemment et illustrées par la figure 3.4, distinguent le générateur de séquences aléatoires et le mélangeur, dans notre cas d'étude un simple XOR à deux entrées. Néanmoins, cette séparation nécessite un processus de synchronisation au niveau de la réception pour permettre au récepteur de bien faire correspondre le début du message préalablement scramblé au début de la

séquence aléatoire, afin de "dé-mélanger". Cette synchronisation est couteuse en temps et pour y remédier des normes visant des débits élevés comme le mode High Rate DSSS du 802.11, n'utilise pas à proprement parlé de séquences initiales pour la générateur de séquence mais directement les données du message entrant permettant une auto synchronisation au niveau de la réception. Ainsi, par rapport au scrambler du mode OFDM du 802.11(Figure 3.4), avec le même polynôme générateur, le scrambler avec auto synchronisation devient celui de la figure Figure 3.6.

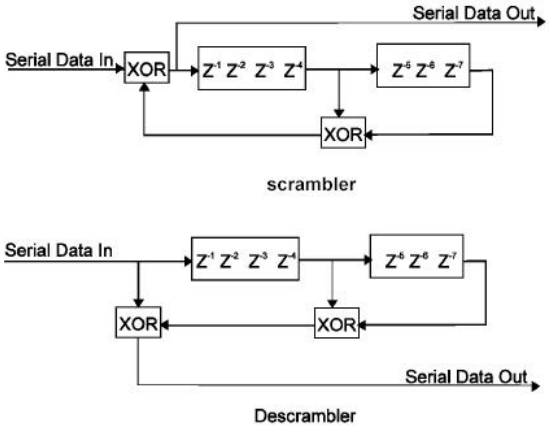

FIGURE 3.6 – Architecture du Scrambler et du Descrambler du 802.11 mode High Rate DSSS

Ainsi, pour une data e_i a embrouillée et la sortie associée s_i, la relation liant s_i et e_i dans le cas de la figure 3.6 est $s_i = e_i + s_{i-4} + s_{i-7}$. Nous proposons ici de généraliser l'équation d'une telle structure :

$$s_i = \sum_{k=1}^{N} s_{i-k} \cdot ak + e_i. \tag{3.4}$$

Le décodeur de ce type de scrambler ne peut être réalisé par la même structure d'embrouillage étant que les données sont directement intégrées dans la génération de séquence pseudo aléatoire. Nous devons utiliser la forme Duale de cette structure qui permettra de dé-embrouiller le signal recu. Celle-ci est présentée en 3.6 et correspond à un filtre FIR (convolueur), avec pour équation de sortie des s'_i :

$$s'_i = \sum_{k=1}^{N} s_{i-k} \cdot bk + s_i. \tag{3.5}$$

Ainsi pour l'équation finale de l'embrouillage/désembrouillage donne :

$$s'_i = \sum_{k=1}^{N} s_{i-k} \cdot bk + s_i = \sum_{k=1}^{N} s_{i-k} \cdot bk + \sum_{k=1}^{N} s_{i-k} \cdot ak + e_i. \tag{3.6}$$

Comme nous travaillons dans des opérations de GF(2), en prenant $b_k = a_k$ alors on retrouvera bien $s'_i = e_i$.

3.2.2 Codes Cycliques Binaires.

Le Cyclic Redundancy Check ou CRC est une technique couramment utilisée en télécommunications pour détecter les erreurs mais non pour les corriger. Dans la définition des CRC, un nombre fini de bits est ajouté à la suite du message et permet ainsi lors de la réception de vérifier l'intégrité de ces données. Si le contrôle d'erreur revient négatif, alors le récepteur envois un acknowledge négatif (NAK) à l'émetteur et demande la retransmission du message. Ce système est non seulement utilisé dans le domaine des télécommunications mais aussi dans la sauvegarde et l'enregistrement des données comme les disques durs.

3.2.2.1 Principe du Codage Cyclique Binaire.

Les codes cycliques constituent un sous-groupe des codes linéaires qui a pris une très grande importance dans les applications du fait de la simplicité d'implantation des algorithmes de codage et de décodage. En informatique, un contrôle de redondance cyclique ou CRC (Cyclic Redundancy Check) est un outil permettant de détecter les erreurs de transmission par ajout de redondance. La redondance ajoutée communément mais faussement appelée somme de contrôle (checksum) est obtenue par un type de hachage sur l'ensemble des données. Les CRC sont calculés avant et après la transmission (figure 3.7), puis comparés pour s'assurer que de leurs similitudes. Les calculs de CRC les plus utilisés sont construits de manière à ce que les erreurs de certains types, comme celles dues aux interférences dans les transmissions, soient toujours détectées.

Le contrôle de redondance cyclique est un moyen de contrôle d'intégrité des données puissant et facile à mettre en oeuvre. Il représente une des principales méthodes de détection d'erreurs utilisées dans les télécommunications. Le contrôle de redondance cyclique consiste à protéger des blocs de données, appelés trames. A chaque trame est associé un bloc de données. Le code CRC contient des éléments redondants vis-à-vis de la trame, permettant de détecter les erreurs, mais aussi de les réparer. Le principe du CRC consiste à traiter les séquences binaires comme des polynômes binaires, c'est-à-dire des polynômes dont les coefficients correspondent à la séquence binaire. Ainsi la séquence binaire 0110101001 peut être représentée sous la forme polynomiale suivante : $0 * x^9 + 1 * x^8 + 1 * x^7 + 0 * x^6 + 1 * x^5 + 0 * x^4 + 1 * x^3 + 0 * x^2 + 0 * x^1 + 1 * x^0$ soit $x^8 + x^7 + x^5 + x^3 + 1$. De cette façon, le bit de poids faible de la séquence (le bit le plus à droite) représente le degrés 0 du polynôme. Une séquence de n bits constitue donc un polynôme de degré maximal n-1. Toutes les expressions polynomiales sont manipulées par la suite avec une arithmétique modulo 2.

Dans ce mécanisme de détection d'erreur, un polynôme prédéfini (appelé polynôme générateur et noté g(x)) est connu de l'émetteur et du récepteur. La détection d'erreur consiste pour l'émetteur à effectuer un algorithme sur les bits de la trame afin de générer un CRC, et de transmettre ces deux éléments au récepteur. Il suffit alors au récepteur

d'effectuer le même calcul afin de vérifier que le CRC est valide.

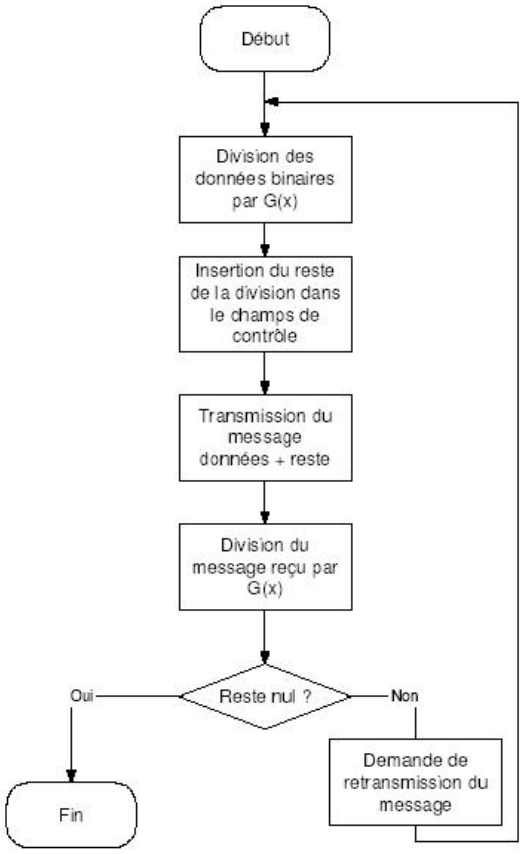

FIGURE 3.7 – Principe du CRC

3.2.2.2 Codes Cycliques Binaires.

Le calcul du CRC est basé sur l'arithmétique de polynôme et plus particulièrement sur le calcul du reste de la division polynomiale sur GF(2) (Corps de Galois à deux éléments). Comme expliqué précédemment, le reste de la division est calculé en divisant le message reçu. Or cette division opère dans GF(2). L'addition et la soustraction sont identiques et s'opèrent via une addition modulo 2 c'est à dire un XOR. Une division peut se ramener à une suite de décalages et soustractions ou a chaque itération, on vérifie la présence dans le dividende d'une puissance du diviseur et ainsi de suite jusqu'à ce que le reste soit de degrés inférieur au diviseur. Dans GF(2), la soustraction se réalise par le biais d'une opération logique de type ou-exclusif. Nous représentons en figure 3.8, la division de $x^7 + x^6 + x^4 + x^3 + x + 1$, par $x^3 + x + 1$. Nous pouvons des lors bien distinguer la succession de décalage et d'addition modulo 2 qui s'enchaine. Il est

à noter que la soustraction n'est pas systématique à tous les étages, elle ne se produit que lorsque le degrés du sous polynôme représenté par le dividende est équivalent au degrés du diviseur. Concrètement, elle ne se produit que lorsque le bit le plus à gauche du dividende considéré vaut 1.

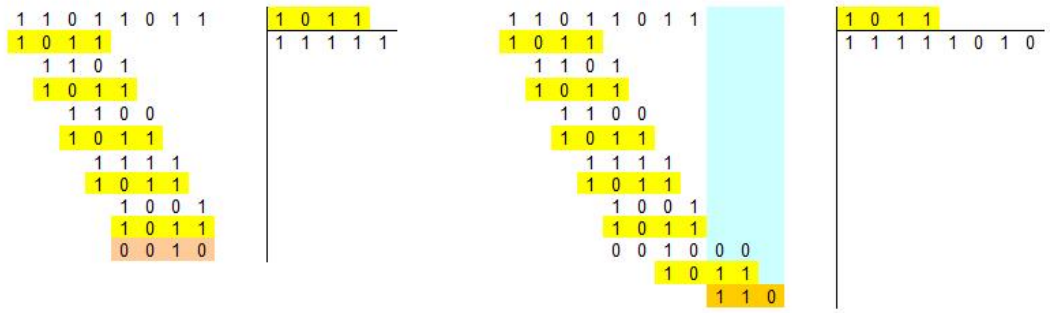

FIGURE 3.8 – Division Polynomiale

La représentation de cette opération de décalage/soustraction peut se réaliser par un LFSR (Figure 3.9) dans la mesure où les opérations se déroulent sur GF(2). Ici, contrairement à l'arbre de la figure 3.8, ce n'est pas le diviseur qui est décalé mais la partie à diviser. Le diviseur sera représenté par les coefficients du LFSR et sera additionné (XOR) au message entrant décalé à chaque itération. La soustraction dépend du bit de poids forts du dividende considéré, pour permettre :

- soit uniquement le décalage
- soit le décalage et la soustraction

Dans le cas du codage du CRC, pour un message M(x) reçu, et le polynôme diviseur g(x) de degrés r, nous devrons diviser $x^r.M(x)$ par g(x). Comme la multiplication polynomiale est associative, cette division pourra être réalisée en ajoutant r itération à la division avec comme entrée "0", ce qui correspond à une multiplication à postériori par x^r. Les deux divisions (figure 3.8) et le LFSR (3.9) illustrent ce principe avec M(x)= 11011011 et $g_(x)$ =$x^3 + x + 1$.
Vis à vis des trois standards considérés dans cette étude ; IEEE 802.11, IEEE 802.16 et 3GPP LTE, nous résumons toutes les opérations de CRC dans ces standards 3.10.

3.2.3 Codages Convolutifs

Lors d'une transmission à distance, l'information peut subir des transformations, pouvant être dues à des réflexions multiples sur des obstacles ou à des atténuations dues au canal de transmission. Pour protéger les données numériques lors d'un transfert, il existe des technologies de codage correcteur d'erreur (FEC), ou codage de canal, qui ajoutent une information de redondance selon des règles qui sont connues du récepteur. Il permet donc d'extraire au mieux l'information d'origine, même si le signal

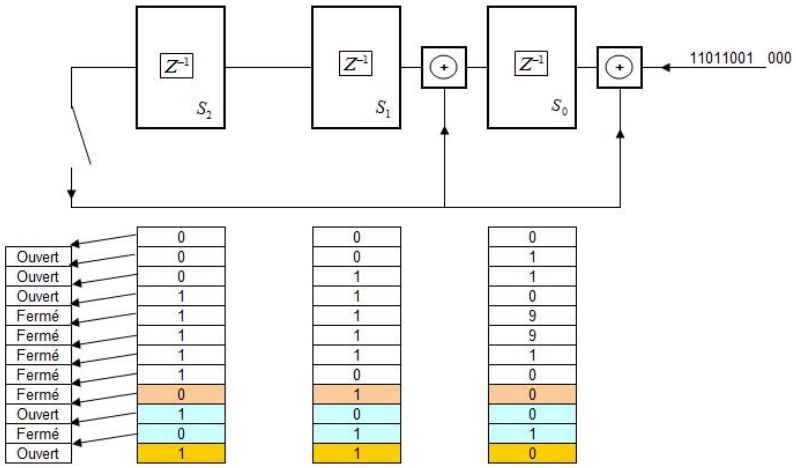

FIGURE 3.9 – LFSR et Dividion Polynomiale

CRC (IEEE 802.11, IEEE 802.16, 3GPP LTE)				
Standard	Polynôme	Type de Structure Requise	Utilisation	Remarque
IEEE 802.11	$x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^9+x^8+x^6+1$	LFSR - Galois	Couche PHY – Codage Data.... (Optionnel)	CRC Castagnoli Emission et Réception
IEEE 802.11	$x^{16}+x^{12}+x^5+1$	LFSR - Galois	Couche PHY – Codage PLCP, HEC, FEC....	CRC 16 – CCITT Emission et Réception
3GPP LTE	$x^{24}+x^{23}+x^6+x^5+1$	LFSR - Galois	Couche PHY – Codage Data....	Emission et Réception
3GPP LTE	$x^{16}+x^{12}+x^5+1$	LFSR - Galois	Couche PHY – Codage Data....	Emission et Réception
3GPP LTE	$x^{12}+x^{11}+x^3+x^2+x+1$	LFSR - Galois	Couche PHY – Codage Data....	Emission et Réception
3GPP LTE	$x^8+x^7+x^4+x^3+x+1$	LFSR - Galois	Couche PHY – Codage Data....	Emission et Réception
IEEE 802.16	$x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^9+x^8+x^6+1$	LFSR - Galois	Couche PHY – Codage Data.... (Optionnel)	CRC Castagnoli Emission et Réception
IEEE 802.16	$x^{16}+x^{12}+x^5+1$	LFSR - Galois	Couche PHY – Codage Data....	CRC 16 – CCITT Emission et Réception

FIGURE 3.10 – Inventaire des CRC Requis

est fortement altéré. Les codes convolutifs forment une classe extrêmement souple et efficace de codes correcteurs d'erreurs. Ce sont les codes les plus utilisés dans les systèmes de télécommunications fixes et mobiles, s'appliquant sur des séquences infinies de symboles d'information et générant des séquences infinies de symboles codés.

3.2.3.1 Code convolutif de rendement 1/n

Considérons le code en bloc linéaire binaire de taille B, spécifié par la matrice génératrice en Figure 3.11. Chaque éléments g_l $(0 \leq l \leq L-1)$ de la matrice est un vecteur ligne binaire à n composantes de la forme : $g_l = [g_{l1}, g_{l2}, ...g_{ln}]$.

$$G = \begin{bmatrix} \mathbf{g_0} & \mathbf{g_1} & \mathbf{g_2} & \cdots & \mathbf{g_{L-1}} & & & & \\ & \mathbf{g_0} & \mathbf{g_1} & \mathbf{g_2} & \cdots & \mathbf{g_{L-1}} & & & \\ & & \mathbf{g_0} & \mathbf{g_1} & \mathbf{g_2} & \cdots & \mathbf{g_{L-1}} & & \\ & & & \ddots & & & & \ddots & \\ & & & & \ddots & & & & \ddots \\ & & & & & \mathbf{g_0} & \mathbf{g_1} & \mathbf{g_2} & \cdots & \mathbf{g_{L-1}} \end{bmatrix}$$

FIGURE 3.11 – Matrice du Codage en Bloc

Les zones laissées vides en bas à gauche et en haut à droite correspondent à des éléments tous nuls. Chaque ligne s'obtient en décalant les L éléments de la ligne précédente de n colonnes vers la droite. Cette matrice, de taille B x (n.B+L-1) décrit donc un code linéaire en bloc dont les mots de code pourraient être engendrés par le dispositif de la Figure 3.12. Généralement, les mots de codes sont très longs, c'est à dire B > L, permettant de considérer cette matrice comme infinie. Il existe un rapport 1/n entre le nombre de bits d'information et le nombre de bits du mot de code. Le rendement de ce code est donc : r=1/n (Figure 3.12).

La quantité L est appelée longueur de contrainte du code convolutif. Le terme convolutif s'applique a cette classe de codes parce que la suite de bits codes s peut s'exprimer comme le résultat de la convolution de la suite de bits d'information e par les coefficients g. En effet, puisque le code est linéaire, nous avons : s = e . G. En observant la forme particulière de G, les n bits en sortie du codeur à l'instant t, correspondant à une entrée, s'écrivent :

$$s_t = \sum_{u=max(l,t-L+1)}^{t} e_k \cdot g_{u-k} \tag{3.7}$$

Nous pouvons illustrer cette définition par un codeur de rendement 1/2, de longueur de contrainte L=3 de polynomes générateurs $g_1(x) = 1+x+x^2$ et $g_2(x) = 1+x^2$ (Figure 3.13).

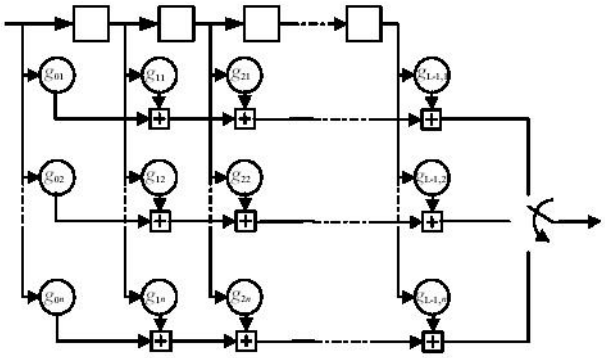

FIGURE 3.12 – Schématique d'un codeur convolutif

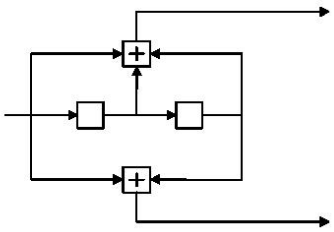

FIGURE 3.13 – Schématique d'un codeur convolutif (1/2)

3.2.3.2 Code convolutif de rendement k/n

Pour obtenir un rendement égal à un nombre rationnel quelconque inférieur à 1, il faut généraliser la forme de la matrice génératrice (3.14) et le schéma à la figure 3.12 du codeur. Nous supposons maintenant que k bits d'information sont introduits en parallèle à un instant donné dans le codeur ou qu'un décalage temporel du vecteur e des bits d'information se fait par bloc de k bits. Cela revient à remplacer la ligne $[g_0, g_1, , g_{L-1}]$ de taille 1 x L.n, par une sous matrice binaire de taille k x L.n. dont les éléments constituant g_l sont de taille k x n. La construction générale par décalage de la matrice reste la même (figure 3.14).

$$G = \begin{bmatrix} \overset{n}{\longleftrightarrow} & & & & & \\ k \updownarrow \mathbf{g}_0 & \mathbf{g}_1 & \cdots & \mathbf{g}_{L-1} & & \\ & \mathbf{g}_0 & \mathbf{g}_1 & \cdots & \mathbf{g}_{L-1} & \\ & & \ddots & & & \ddots \\ & & & \mathbf{g}_0 & \mathbf{g}_1 & \cdots & \mathbf{g}_{L-1} \end{bmatrix}$$

FIGURE 3.14 – Matrice de rendement k/n

Prenons l'exemple d'un codage convolutif de rendement 2/3 et de longueur de contrainte L=2 défini par les coefficients g_0 et g_1. La structure de ce codeur est donnée en figure 3.15 :

$$g_0 = [111; 010] \; ET \; g_1 = [101; 110] \tag{3.8}$$

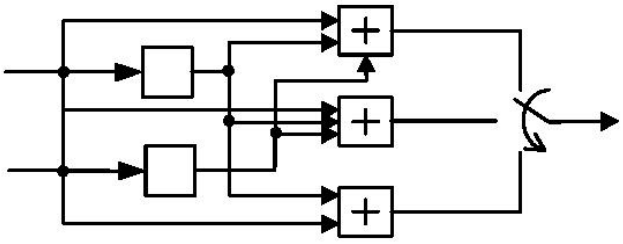

FIGURE 3.15 – Codeur 2/3

La sortie d'un codeur convolutif à l'instant t dépend de ses entrées a l'instant t et de l'état des registres du codeur à cet instant. En posant $\nu = L-1$, tout code convolutif de rendement k/n et de longueur de contrainte L possède $k.\nu$ registres, qui peuvent donc prendre $2^{k.\nu}$ états différents. Comme l'entrée est constituée de k bits différents, il existe 2^k transitions possible à partir d'un état du codeur à un instant donne.

Soit un code convolutif de rendement k/n et de longueur de contrainte L. Son entrée peut être vue comme la donnée de k séquences infinies de bits, que nous représentons par des séries en x :

$$e_i(x) = \sum_{l=0}^{+\infty} e_{il} \cdot x^l, i = 1, ..., k \tag{3.9}$$

où l'inconnue x symbolise un retard d'une période. Le coefficient e_{il} est le bit présent à la i-eme entrée du codeur à l'instant l. De la même manière, les n sorties du codeur peuvent être indexées par j et s'écrivent :

$$s_j(x) = \sum_{l=0}^{+\infty} e_{jl} \cdot x^l, j = 1, ..., k \tag{3.10}$$

La relation introduite par le code convolutif entre les entrées et les sorties peut s'écrire de façon élégante avec ses notations, en introduisant les polynômes générateurs. Le polynôme générateur qui lie la i-eme entrée à la jieme sortie est note $g_{ij}(x)$, et est de degré inférieur ou égal à L-1. Le coefficient de degré r de ce polynôme vaut 1 si une connexion existe entre la sortie j et le bit présent dans le r-ieme registre correspondant a l'entrée i.

En se référant à l'écriture polynomiale précédente nous obtenons l'écriture de chaque sortie du codeur :

$$s_j(x) = \sum_{i=1}^{k} e_i(x) \cdot g_{ij}(x), j = 1, ..., n \tag{3.11}$$

3.2.3.3 Code convolutif récursifs systématiques

Un code convolutif est dit récursif lorsque ses polynômes générateurs sont remplacés par les quotients de deux polynômes. Une partie de la sortie est alors réintroduite dans le registre à décalage selon les connexions définies. Un code convolutif est quant à lui, dit systématique lorsque une partie de ses sorties est exactement égale à ses entrées. Cela revient à dire que $(g_{11}(x) = 1, g_{21}(x) = 0, , g_{k1}(x) = 0), (g_{12}(x) = 0, g_{22}(x) = 1, , g_{k2}(x) = 0)$ et $(g_{1k} = (x) = 0, g_{2k}(x) = 0, ..., g_{kk}(x) = 1)$.

Nous pouvons rajouter qu'il est possible de passer d'un codeur NSC de rendement k/(k+1) en un codeur récursif RSC. Si nous prenons l'exemple d'un codeur NSC de rendement 1/2. Il est défini par $g_{11}(x) = g_1(x)$ et $g_{12}(x) = g_2(x)$ tel que $G = [g_1(x), g_2(x)]$.

La transformation en RSC donne $G = [1, \frac{g_1(x)}{g_2(x)}]$. Reprenons l'exemple de la Figure 3.13, les sorties $s_1(x)$ et $s_2(x)$ s'exprime en fonction de x telles que : $s_1(x) = (1 + x + x^2).e(x)$ et $s_2(x) = \frac{1+x^2}{e(x)}$. Sous forme d'un code RSC, la sortie $s_1(x)$ suit l'entrée $(s_1(x) = e(x))$ et la sortie $s_2(x)$ subira la division polynomiale $s_2(x) = \frac{1+x^2}{1+x+x^2}.e(x))$. structure du codeur RSC devient donc celle d'un Filtre IIR comme l'illustre la figure 3.16.

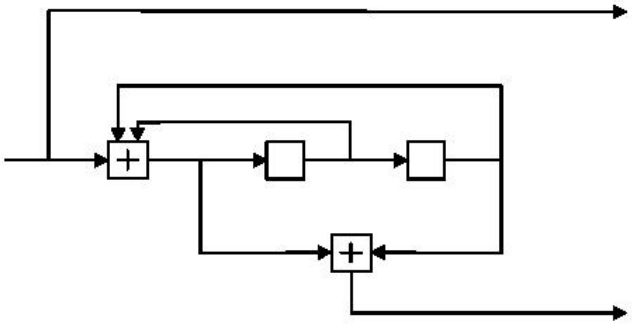

FIGURE 3.16 – Codeur RSC (1, 5/7)

3.2.3.4 Code convolutif et Turbo code.

Les Turbo Codes [BC99] consistent à améliorer le décodage par un procédé itératif qui permet d'affiner progressivement le résultat. Pour celà, l'information est codée de deux manières différentes, en parallèle, au niveau de l'émetteur. Le décodage est quant à lui effectué en série, suivant l'un puis l'autre des codes élémentaires. Ce deuxième décodage constitue la grande originalité des Turbo Codes. Celui-ci permet en effet d'apporter des informations susceptibles d'améliorer le premier traitement des données. Celles-ci peuvent être ensuite exploitées pour relancer le processus. En itérant ce décodage, de plus en plus d'erreurs peuvent être corrigées. Cette particularité a d'ailleurs donné son nom à cette technologie : les informations échangées entre les décodeurs au cours des itérations améliorent les performances, comme une partie des gaz d'échappement favorise la combustion dans les moteurs turbo.

Pour illustrer au mieux le fonctionnement des Turbo Codes, leurs inventeurs, Claude Berrou [BC99] et Alain Glavieux, ont fait une analogie avec les mots-croisés. Il faut pour cela imaginer que les mots de la grille correspondent aux données d'origine et que les définitions horizontales et verticales correspondent aux informations de redondance issues du codage. En effet, dans une grille de mots-croisés, les définitions constituent deux codages parallèles mais différents pour un seul et même résultat : l'ensemble des mots dans la grille. Si les définitions ainsi que la grille sont transmises et que des erreurs apparaissent dans cette dernière, le décodage va alors permettre de retrouver la grille d'origine en appliquant successivement des décodages horizontaux et verticaux. Le procédé est ensuite reproduit, en utilisant les informations produites par le décodage précédent, autant de fois que nécessaire jusqu'au décodage complet de la grille.

Le principe des turbo-codes, comme tout code correcteur d'erreur, est donc d'introduire une redondance dans le message afin de le rendre moins sensible aux bruits et perturbations subies lors de la transmission. Les Turbos Codes ont pour particularité d'associer plusieurs codeurs convolutionnels (en général au nombre de 2) mais en appliquant sur les entrées d'un des deux codeurs un Interleaver qui modifie de manière connue la séquence d'entrée. Les codeurs convolutionnels utilisés dans ce type de codage

sont majoritairement des codeurs RSC, présentés précédemment.

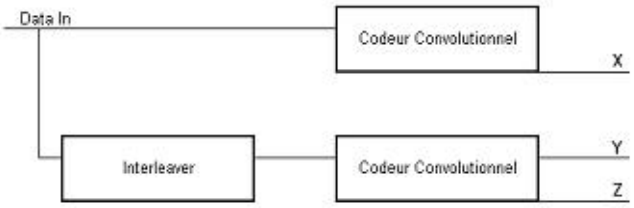

FIGURE 3.17 – Principe du Turbo Codage

3.2.3.5 Code Circulaire

Les codes circulaires ne sont pas un type de codes particuliers mais une méthode de codage qui peut s'appliquer aux codes NSC comme RSC. Même s'ils ne nécessitent pas une architecture particulière, nous présentons ici ce types de codes car ils sont très utilisés et permettent de définir les turbo codeurs duo-binaires [BC99] utilisés dans le 802.16.

Le codage d'un bloc d'information à l'aide d'un code convolutif fait apparaître, au niveau du décodage, des discontinuités aux extrémités de ce bloc. En effet, les algorithmes de décodage des codes convolutifs font appel, pour le décodage du symbole à l'instant i, à l'ensemble des informations antérieures et postérieures à i. Dans le cas d'une transmission par blocs, si l'état initial du codeur est inconnu, les informations en début de bloc ne peuvent pas bénéficier du " passé ". De même, si l'on ne dispose d'aucune information sur l'état du codeur en fin de codage du bloc, les dernières données ne peuvent pas bénéficier du " futur ". En conséquence, les performances de correction sont dégradées par une moindre protection des symboles situés aux extrémités du bloc. Ce phénomène est d'autant plus marqué que le bloc est de petite taille [BC99]. Ce problème peut être résolu si l'on sait "fermer le treillis", c'est-à-dire si le décodeur connaît l'état initial et l'état final du codeur. En pratique la connaissance de l'état initial du codeur ne pose pas de problème particulier, car il suffit de convenir à l'avance de sa valeur : le plus souvent, tous les éléments mémoire des registres de codage sont forcés à la valeur 0 (état "tout zéro", noté 0) au début du codage de chaque bloc. La connaissance de l'état final pose un problème beaucoup plus délicat, celui-ci dépendant du contenu du bloc à coder. Parmi les différentes techniques pouvant être mises en oeuvre, la plus simple d'entre elles consiste à fermer le treillis d'un ou des deux codes élémentaires à l'aide de bits de terminaison ou tail bits. Dans le cas des turbocodes du standard UMTS (base du 3GPP LTE), les bits assurant la terminaison de l'un des deux treillis ne sont pas utilisés dans l'autre codeur [BC99]. Ces bits ne sont donc pas turbocodés ce qui conduit, mais dans une moindre mesure, aux mêmes inconvénients que ceux présentés par une absence de fermeture. De plus, la transmission des bits de fermeture entraîne une diminution du rendement de codage et donc de l'efficacité spectrale. Par conséquent, cette méthode de

terminaison, bien que couramment utilisée pour les codes convolutifs, n'est pas satisfaisante dans le cas des turbocodes comme le décrit Berrou en [BC99]. Avec la technique des codes convolutifs circulaires (ou tailbiting codes) [BC99], le codeur retrouve en fin de codage de chaque bloc son état initial, cet état dépendant du contenu du bloc. En résumé, il existe pour chaque bloc de taille k de la séquence de données un état S_c, appelé état de circulation, qui garantit que, si le codage du bloc démarre à partir de l'état S_c, il se termine également dans l'état S_c.

Cette méthode présente donc l'inconvénient de nécessiter une étape de pre-codage de chaque message pour connaitre la valeur S_c. Le treillis du code prend alors la forme d'un cercle [BC99] et peut être considéré comme un treillis de longueur infinie du point de vue du décodeur. L'existence de cet état S_c dépend de l'inversibilité de la matrice caractéristique du registre à décalage à rétroaction linéaire considéré. En effet, en [BC99] et illustré par la figure 3.18 de la partie 3.2.3.6, le codeur convolutif du turbo codeur circulaire peut se définir par l'équation de récurrence qui lie l'état des registres S_i à l'itération i :

$$S_i = G.S_{i-1} + X_i \tag{3.12}$$

Cette relation de récurrence peut s'écrire :

$$S_i = G^i.S_0 + \sum_{p=1}^{i} G^{i-p}.X_p \tag{3.13}$$

Ainsi, il devient possible de trouver un état S_c tel que : $S_c = S_k = S_0$:

$$S_c = \left\langle I + G^k \right\rangle^{-1} . \sum_{p=1}^{k} G^{k-p}.X_p \tag{3.14}$$

Cet état ne peur être atteint que si $\left\langle I + G^k \right\rangle$ est inversible. De manière à pouvoir comparer nos architectures en LFSR proposées en 4.3, nous utiliserons une expression matricielle identique dans la définition de nos OC.

3.2.3.6 Turbo Code m Binaires Circulaires

Berrou et Douillard ont utilisé le principe du codage circulaire, pour améliorer la performance des turbo codes et définir les Turbo Code m Binaires Circulaires [BC99]. Ces derniers reprennent le principe de codage circulaire présenté précédemment et utilise des codeurs convolutifs RSC à m entrées binaires. Ces codeurs bien que pouvant présenter m entrées font en pratique appel à des codes élémentaires à huit ou seize états (3.18) pour des raisons de complexité de décodage. Ainsi, un turbo code duo-binaire circulaire à huit états a été normalisé pour la voie de retour du réseau satellitaire de télévision numérique DVB-RCS [ETS00]. Les applications DVBRCS visent des transmissions de données utilisant sept tailles de blocs possibles, de 12 à 216 octets, et cinq rendements de codage différents (1/2, 2/3 , 3/4, 4/5 et 6/7). Ce code a ensuite été étendu à des codes élémentaires à seize états afin d'en augmenter les distances minimales et, par

conséquent, les performances asymptotiques. Ce premier codeur duo-binaire est celui la même utilisé et défini dans la norme IEEE 802.16 dans le cas des modulation OFDM et OFDMA, où il vise à remplacer le codage NSC 1/2 initialement spécifié.

La conception des codeurs m Binaires Circulaires redéfinis le concept du codeur RSC. Celui-ci est présenté comme une générateur de séquences aléatoires où chaque registre est alimenté par des combinaisons linéaires des m entrées, permettant ainsi de mélanger les données avec la séquence aléatoire formée par ces mêmes entrées aux itérations précédentes. Les n sorties d'un tel codeur sont générées par une matrice polynomiale de taille n représentant les polynômes générateurs de chaque branche de sortie [BC99]. Ainsi, vis à vis des explications fondatrices présentées en [BC99], nous dénombrons des avantage certains à ce type de codeurs :

- Meilleure convergence du processus itératif
- Sensibilité réduite vis a vis du poinçonnage pour l'obtention de rendements élevés
- Latence plus faible
- Meilleure robustesse du décodeur envers la sous optimalité
- Plus grandes distances minimales

Ce type de codeur transforme le traitement convolutif présenté précédemment avec les codeurs RSC binaires et codages en bloc de taille m. En ce qui concerne, notre étude d'opérateurs communs, nous devons identifier la forme générale que prend ce type de codage. Nous présentons, en figure 3.18, l'architecture du codeur à 16 états et la forme générale, du codeur RSC m Binaires en occultant les branches de sorties afin de permettre une meilleure visibilité de notre représentation.

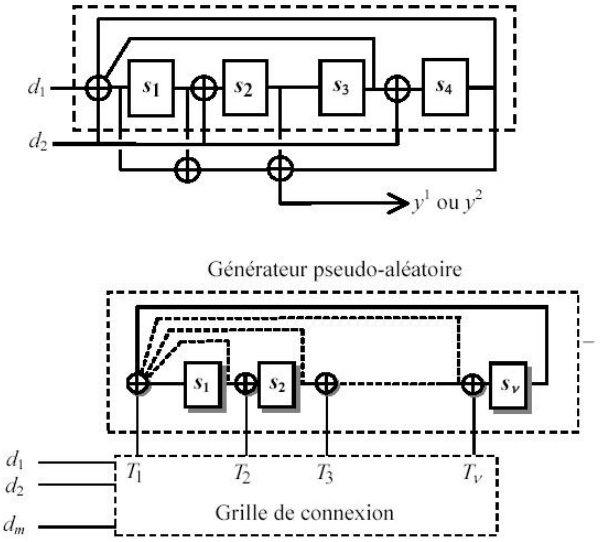

FIGURE 3.18 – Architecture du codeur RSC d'un Turbo Code m Binaires Circulaires

Vis à vis des trois standards considérés dans cette étude, IEEE 802.11, IEEE 802.16

et 3GPP LTE, nous résumons toutes les opérations de codages convolutionnels dans ces standards (figure 3.19).

3.3 Factorisation Des Opérations utilisant des systèmes de Treillis et de Papillons

Dans les sections qui suivent, nous étudions les différentes étapes d'exécution des algorithmes de Viterbi et de la FFT. La FFT (présentée en 2.2) et l'algorithme de Viterbi appartiennent à la Classe 2 qui met en oeuvre des modules d'additions et de multiplications. Ici, nous allons plus loin que les fonctions d'addition/multiplication en faisant apparaitre une similarité de fonctionnement en treillis/papillons. Il est à noter que les deux algorithmes FFT et Viterbi sont aussi présentés comme des algorithmes communs à différents traitements.

3.3.1 Décodage des Codes Convolutifs et Algorithme de Viterbi

3.3.1.1 Le Décodage

Nous expliquons longuement le principe des codeurs convolutifs en 3.2.3. Il existe peu de procédés de réalisation pour l'algorithme de Viterbi [Ras02], [STGB02] et [BSL+04]. Cet algorithme a pour but de trouver la séquence d'états la plus probable ayant produit la séquence mesurée $[y_0,...,y_r]$. Concrètement, dans un codeur convolutif, le bit entrant dans le registre à décalage fait se décaler le registre et produit un résultat en sortie. Le message que nous recevons est en théorie la sortie de ce codeur. Or il a été altéré par le canal. Pour pouvoir reconstituer le message d'origine, c'est-à-dire les bits entrants précédents, nous allons procéder par récurrence.

Chaque bit que nous recevons est du à un bit entrant qui a fait se décaler le registre et qui produit une sortie. En théorie, cette sortie doit être égale au bit reçu. En réalité, c'est rarement le cas. Par contre, en comparant la valeur reçu avec toutes les sorties possibles du registre, nous pouvons connaitre quel a été le déplacement de ce registre le plus probable qui a engendré la sortie et donc la valeur reçue. En connaissant, ce déplacement, nous pouvons connaitre la bit qui l'a provoqué et par conséquent, la valeur correspondant du bit entrant au message reçu. Nous obtenons pour chaque valeur du message reçu une structure en treillis dont un exemple est donné en figure 3.20. Pour chaque donné reçu, nous obtenons le passage le plus probable d'un état du registre à l'autre. En répétant, ce calcul pour chaque, valeur du message reçu, on obtiendra un chemin reliant les états les plus probables des registres et par conséquent les valeurs des entrées.

Nous avons besoin d'une étape de calcul pour chaque valeur reçue, l'algorithme de Viterbi définit trois entités ; (1) le module BMC qui calcule la différence entre la valeur reçue et les sorties possibles du registre, (2) le module ACS qui va additionner cette valeur calculée aux valeurs calculées pour les valeurs reçues précédentes et choisir la plus faible, (3) le module de détermination du message entrant (SMM) qui va déterminer la

Codage Convolutif (IEEE 802.11, IEEE 802.16, 3GPP LTE)

Standard	Polynôme	Type de Structure Requise	Utilisation
IEEE 802.11	$x^6+x^4+x^3+x+1$ $x^6+x^5+x^4+x^3+1$	Codeur Convolutionnel NSC 1/2	Codage pour les données dans le cas des modulation FHSS, OFDM, DSSS et High Rate DSSS.
IEEE 802.11	x^4+1 et x^3 x^4+x^2+1 et x x^3+x et x^3+x	Codeur Convolutionnel RSC 2/3	Codage pour les données dans le cas du mode ERP.
3GPP LTE	$x^8+x^4+x^3+x^2+1$ $x^8+x^7+x^5+x^3+x^2+x+1$	Codeur Convolutionnel NSC 1/2	Codage pour le cas d'un canal de type : BCH, PCH, RACH.
3GPP LTE	$x^8+x^7+x^6+x^5+x^3+x^2+1$ $x^8+x^7+x^4+x^3+x+1$ $x^8+x^5+x^2+x+1$	Codeur Convolutionnel NSC 1/3	Codage pour le cas d'un canal de type : CPCH, DCH, DSCH,FACH.
3GPP LTE	x^3+x^2+1 x^3+x+1	Turbo Codes 1/3 Deux Codeurs RSC	Codage pour le cas d'un canal de type : CPCH, DCH, DSCH,FACH.
IEEE 802.16	$x^6+x^3+x^2+x+1$ $x^6+x^3+x^2+1$	Codeur Convolutionnel NSC 1/2	- Optionnelle en complément d'un code RS (GF(8)) pour le mode Single Carrier. - Obligatoire dans les modes OFDM et Single Carrier Acces en complément du RS. -Obligatoire dans l'OFDMA (Pas de RS)
IEEE 802.16	x^3+x+1 x^3+x^2+1	Turbo Codes 2/2 Duo- Binaire CRSC	-Alternative dans le cas de l'OFDM.
IEEE 802.16	x^3+x+1 x^3+x^2+1 x^3+1	Turbo Codes 2/3 Duo- Binaire CRSC	-Alternative dans le cas de l'OFDMA.

FIGURE 3.19 – Inventaire des Codes Convolutionnels Requis

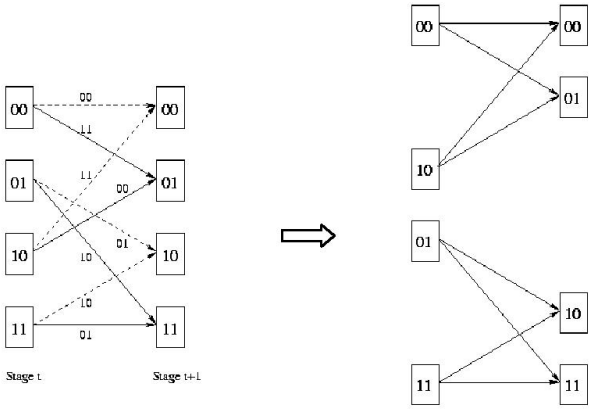

FIGURE 3.20 – Exemple de diagramme en treillis (K=3, R=1/2)

valeur du message entrant à partir des états les plus probables du registres pour chaque valeur du message reçu :

- BMC Calcul des Métriques de Branches, ceux sont les valeurs Bm.
- ACS Calcul des Métriques de Chemin et des survivants, ceux sont les valeurs Pi.
- SMM Stockage et remontés des survivants

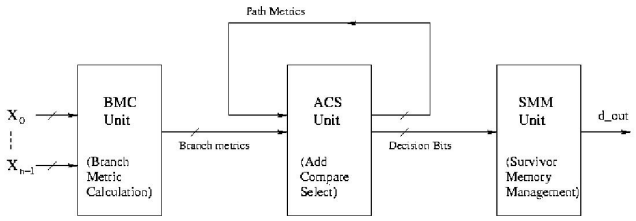

FIGURE 3.21 – Principe de Fonctionnement du Décodeur de Viterbi

Ainsi, la structure en treillis, met en oeuvre entre chaque état du tréillis une double interconnection appelée "Papillon". Dans nos travaux, nous étudierons spécifiquement ce papillon qui peut selon le cas intégrer le module ACS avec ou sans le module BMC.

3.3.1.2 Extention du Décodage

D'autres types de codeurs utilisant des codeurs convolutifs existent, ils sont appelés "Turbo Codeur" (3.2.3). Ainsi, les premiers turbo-codes on été construits à partir d'une concaténation parallèle de deux codes NSC (figure 3.22). On parle alors de Turbo codes convolutifs. Le codeur NSC est généralement de rendement 1/2 et les deux codeurs sont séparés par un entrelaceur. Seulement, une des deux sorties systématiques des deux codeurs est utilisée. Deux algorithmes différents peuvent être utilisés pour le décodage du Turbo codage, l'algorithme MAP (Maximum a Posterori Probability) et

l'algorithme SOVA (Soft Output Viterbi Algorithm). L'algorithme MAP présente un taux d'erreur binaire meilleur que l'algorithme SOVA. Or comme son nom l'indique le SOVA est un algorithme de Viterbi à sorties "soft" et peut être implémenté, pour un décodage de codes convolutifs comme de turbo codes. Dans ce contexte, une architecture reconfigurable dynamiquement (VITURBO) et implémentée sur FPGA a été proposée dans [Vay02] pour le décodage Viterbi et Turbo décodage. Cette solution, basée sur l'algorithme SOVA, tire profit des similitudes qui existent entre ce dernier algorithme et l'algorithme Viterbi pour optimiser l'architecture proposée (figure 3.23).

Un Turbo décodeur (figure 3.22) est composé de cinq blocs de traitement principales : deux blocs SISO (Soft Input Soft Output) implémentant l'algorithme SOVA ou l'algorithme MAP et trois blocs d'entrelacement/désentrelacement.

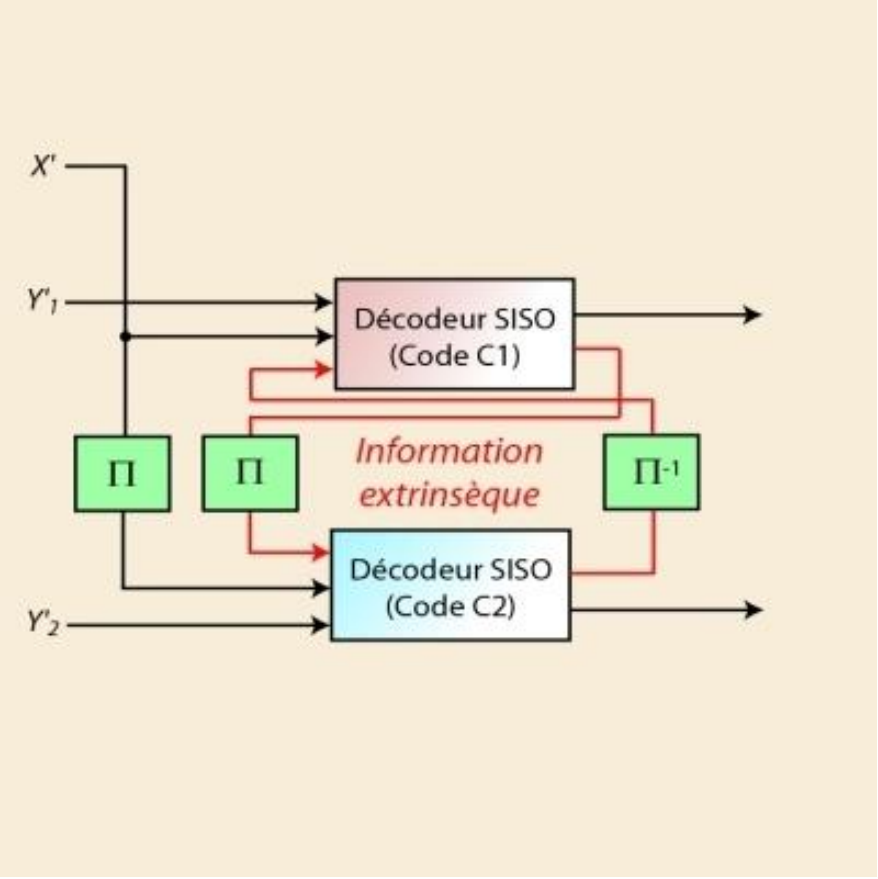

FIGURE 3.22 – Principe du Viturbo

Dans le cas de l'algorithme SOVA utilisé dans l'architecture VITURBO, les trois parties formant le bloc SISO sont similaires aux parties constituantes du décodeur Viterbi mais avec une complexité plus importante. Surtout la partie SMM, elle est extrêmement complexe pour le Turbo décodage que pour le décodage Viterbi vu qu'elle génère des

sorties "Soft". Tenant compte de la complexité que présente SOVA et en profitant des similitudes qu'il présente avec l'algorithme de Viterbi, les auteurs de VITURBO ont trouvé des aspects communs entre ces deux algorithmes 3.23. La structure du treillis est différente. Les similarités dans les opérations de traceback entre les deux types de décodeur ont été exploitées à un troisième niveau de granularité. Par contre, l'opération ACS est par nature similaire et c'est à ce niveau précisément que nous proposons un premier opérateur commun entre la FFT et le Viterbi.

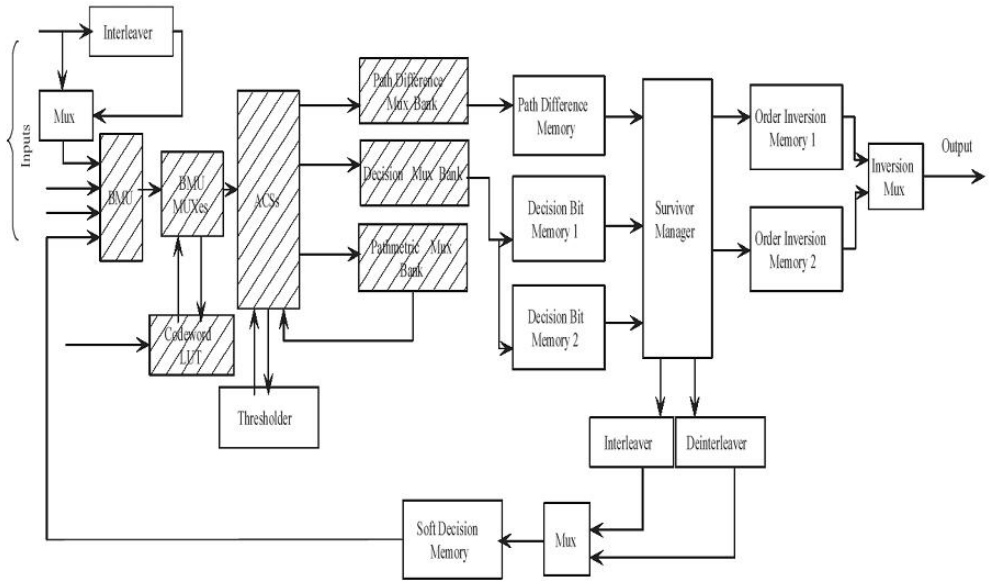

FIGURE 3.23 – Décodeur Viturbo

Les auteurs proposent en figure 3.23 un schéma d'implémentation commun entre les deux techniques :

- Unité BMcompute, selon le type de codeur, elle procède au calcul des métriques de branches. Autres que les entrées données/sorties données, cette fonction reçoit en paramètre une information lui indiquant le type du décodeur actif .
- Unités BMmuxes, suivant le paramètre Codeword fourni par le block Unité Codeword look-Up Table, chacun de ces BMmuxes fournit à son ACS associé, la paire de métriques de branche adéquate.
- Unité Codeword look - Up Table, en fonction de ses paramètres en entrée : longueur de contrainte K, taux de codage (1/n) et type du codeur, fournit le paramètre Codeword
- Unités ACS, chacune parmi eux, reçoit comme entrées les métriques de chemins concernées et calcule les métriques de chemins des chemins survivants, les bits de décision pour les deux types de décodage et la différence entre les métriques de

chemin rentrant dans chaque état du treillis dans le cas du Turbo décodage.
- Unité Multiplexer Banks ; la nouvelle métrique de chemin produit par un ACS est rebouclée à un ensemble approprié d'ACS en fonction de la longueur de contrainte K, c'est le rôle attribué à cette unité.
- Servivor Management Unit (SMU), utilise comme paramètres le type du décodeur, la longueur de contrainte, l'état courant et le bit de décision mémorisé à un certain état pour évaluer l'état précédent et le bit décodé. Le SMU inclut du hardware additionnel pour le calcul de la décision douce nécessaire au Turbo décodage.

3.3.2 La Transformée de Fourier Rapide

La transformée de Fourier rapide (acronyme anglais : FFT ou Fast Fourier Transform) est l'algorithme de calcul de la transformée de Fourier discrète (TFD) que nous avons déjà présenté dans la génèse de la technique des Opérateurs Communs en chapitre II. Cet algorithme est couramment utilisé en traitement numérique du signal pour transformer des données discrètes du domaine temporel dans le domaine fréquentiel et éxécuter un nombre conséquent d'opérations tel que Palicot le propose en [PR02].

Si nous considérons, x_0, ..., x_{n-1} des nombres complexes. La transformée de Fourier discrète est définie par les écritures suivantes :

Forme Linéaire :

$$f_j = \sum_{k=0}^{k=n-1} x_k.e^{-\frac{2.\pi.i}{n}.k.j},\ j = 0, ..., n-1 \tag{3.15}$$

Forme Matricielle :

$$\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ . \\ . \\ . \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & w & w^2 & ... & w^{n-1} \\ 1 & w^2 & w^4 & ... & w^{2.(n-1)} \\ & . & & & \\ . & . & . & . & . \\ & . & & & \\ 1 & w^{n-1} & w^{2.(n-1)} & ... & w^{(n-1)^2} \end{pmatrix} . \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ . \\ . \\ . \\ x_{n-1} \end{pmatrix},\ w = e^{-\frac{2.\pi.i}{n}}. \tag{3.16}$$

Comme la transformée de Fourier inverse discrète est équivalente à la transformée de Fourier discrète, à un signe et à un facteur 1/n près, il est possible de générer la transformation inverse de la même manière pour la version rapide. Nous pouvons dénombrer plusieurs algorithmes qui mènent à bien le calcul de la FFT :

Cooley-Tukey L'algorithme de Cooley-Tukey se base sur une approche de type " diviser pour régner " par le biais d'une récursion qui subdivise une transformation de Fourier discrète d'une taille composite n = $n_1.n_2$ en plusieurs transformées de Fourier discrètes de tailles inférieures n1 et n2. L'utilisation la plus classique de l'algorithme de

Cooley-Tukey est une division de la transformation en deux parties de taille identique (n/2) et ceci à chaque étape. Il est aussi possible de mélanger plusieurs types d'algorithmes lors des subdivisions. Ainsi, de nombreuses méthodes utilisées pour effectuer cette algorithme peuvent être dénombrées ; Radix 2 [MBE+05], Radix 4 [ELM04] et Split Radix [JF07]. La subdivision en Radix 2 nécessite le calcul croisé de deux données de chaque subdivision, ce calcul, dit "en papillon" fera l'objet de la mutualisation avec l'algorithme de Viterbi.

Rader-Brenne L'algorithme de Rader-Brenner est une variante de Cooley-Tukey avec des facteurs de rotation purement imaginaires qui améliorent les performances en réduisant le nombre de multiplications mais au détriment de la stabilité numérique et une augmentation du nombre d'additions [Nev77].

Winograd L'algorithme de Winograd met en oeuvre une factorisation en polynômes cyclotomiques, dont les coefficients sont -1,0 et 1, réduisant ainsi, le nombre de multiplications. On peut voir cet algorithme comme la version optimale en termes de multiplications. Toutefois, des additions supplémentaires sont nécessaires ce qui peut être pénalisant sur les processeurs modernes comportant des unités arithmétiques performantes. Un exemple d'implémentation de cet algorithme est donné en [Lav96].

Rader L'algorithme de Rader est quant à lui destiné aux fenêtres dont la taille est un nombre premier. Il profite de l'existence d'une génératrice pour le groupe multiplicatif modulo n. La transformation discrète dont la taille est un nombre premier s'exprime ainsi comme une convolution cyclique d'une taille n - 1.

La transformée de Fourier continue s'applique aux fonctions de carré intégrable et les représente par une intégrale pondérée d'exponentielles complexes. En discrétisant la fonction périodique, nous obtenons une somme finie d'exponentielles complexes :

$$S(w) = \frac{1}{\sqrt{2.\pi}}.\int_{-}^{+} s(t).e^{-i.w.t}.dt,\ X_k = \sum_{n=0}^{n=N-1} x_n.e^{-\frac{2.\pi.i}{N}.k.n},\ x_n = \frac{1}{N}.\sum_{k=0}^{n=N-1} X_k.e^{\frac{2.\pi.i}{N}.k.n} \tag{3.17}$$

L'algorithme original de Cooley-Tukey dit radix-2, s'applique lorsque la dimension de la FFT est une puissance de 2, et la factorisation à l'échelle k fait appel à 2 transformées de Fourier à l'échelle k-1 :

$$X_k = \sum_{n=0}^{n=N-1} x_n.e^{-\frac{2.\pi.i}{N}.k.n} = \sum_{m=0}^{N/2-1} x_{2m}.e^{-\frac{2.\pi.i}{2.N}.2.m.k} + \sum_{n=0}^{N/2-1} x_{2m+1}.e^{-\frac{2.\pi.i}{M}.(2.m+1).k} \tag{3.18}$$

$$X_k = \sum_{m=0}^{M-1} x_{2m}.e^{-\frac{2.\pi.i}{M}.m.k} + e^{-\frac{2.\pi.i}{M}.k}.\sum_{n=0}^{M-1} x_{2m+1}.e^{-\frac{2.\pi.i}{M}.m.k} \tag{3.19}$$

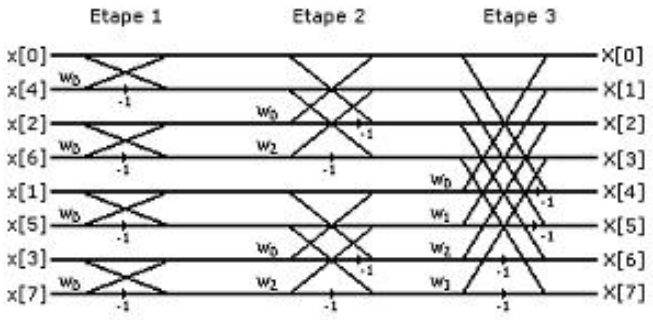

FIGURE 3.24 – Réseau en Treillis de la FFT

Par récurrence une FFT de taille inférieure est réalisée sur les échantillons pairs et une autre sur les impairs à un facteur d'échelle près. La formule du radix-2 met en évidence une cellule de base : le Papillon (butterfly en anglais). Le "Treillis" (figure 3.24) illustre l'utilisation de ce papillon pour résoudre ce problème. Nous pouvons faire apparaître le schéma de ce papillon :

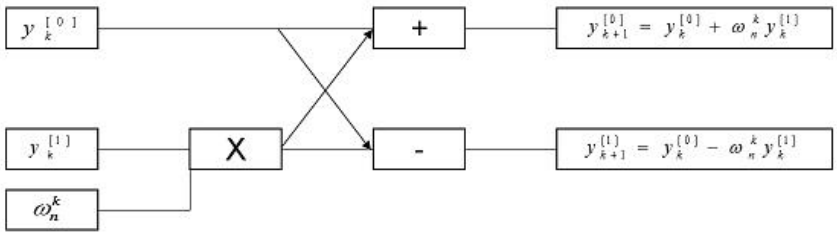

FIGURE 3.25 – Structure en Papillon de la FFT.

3.3.3 Conclusion

L'étude des classes 1 et 2 de notre classification, nous permet de préciser la mise en oeuvre des différentes structures de registres à décalage et de treillis/papillons. Dans le chapitre suivant, nous utilisons notre étude afin de proposer un premier jeu d'opérateurs communs.

Chapitre 4

Proposition d'un premier jeu d'Opérateurs Communs

Sommaire

4.1 Introduction

Dans le chapitre précédent, nous classifions les besoins des différents standards en quatre classes. Nous présentons en détails deux classes et faisons ressortir le besoin en

architectures de type registres à décalages et treillis/papillons. De plus, nous mentionnons dans le chapitre I, les travaux de Palicot et de Wang qui justifient les possibilités de considérer l'algorithme CORDIC et la FFT comme des "opérateurs communs", ces derniers étant utilisés dans un grand nombre d'opérations. Ainsi, nous considérons, de manière générale, le CORDIC et la FFT comme les deux premiers opérateurs communs possibles. Dans ce chapitre, nous présentons, un troisième opérateur commun ; le LFSR et nous définissons un raffinement du papillon FFT pour qu'il intègre les calculs du décodage de Viterbi. Dans chaque cas, nous proposerons plusieurs opérateurs de granularité différente afin d'enrichir le graphe des opérateurs communs et de permettre un choix plus grand à un algorithme de choix.

4.2 Opérateurs traitant les opérations d'Additions et de Multiplications

Ce premier type d'opérateurs communs s'inscrit dans le contexte de la technique des opérateurs communs présentée précédemment et vise à proposer des opérateurs matériels garantissant un fonctionnement en temps réel tout en concevant une forte capacité de reconfiguration. Les opérateurs présentés dans ce chapitre se concentre sur la mutualisation des décodages de Viterbi avec l'algorithme de la FFT qui sont deux fonctions communément utilisées dans les systèmes de communications sans fil. Ces deux fonctions sont très différentes, mais utilisent des structures élémentaires qui présentent certaines similarités. Toutes deux exploitent un système de treillis/papillons. Ainsi, nous proposons deux architectures matérielles de granularité différente permettant de traiter les processus d'addition/soustraction des " papillons " de la FFT et du Décodage de Viterbi.

Les deux architectures que nous proposons relatives aux papillons FFT/Viterbi, permettent d'exécuter plus ou moins les opérations des deux papillons. Concrètement, l'opérateur commun AS^3 pourra remplacer les parties d'addition/soustraction du papillon FFT et du module ACS du Viterbi. L'opérateur FFT/Viterbi quant à lui réalisera l'intégralité du papillon FFT et les modules BMC et ACS du Viterbi. Initiés dans le cadre des Techniques de Paramétrisation et plus particulièrement de la technique des opérateurs communs, ces deux opérateurs utilisent des similarités de fonctionnement et architecturales présentes dans les implémentations de la FFT et de Viterbi. La mutualisation des deux opérateurs nécessite une explication des différents papillons mis en jeux par l'une et l'autre des opération.

4.2.1 Mutualisation des Méthodes

4.2.1.1 Mutualisation des Méthodes : Papillon FFT

En ce focalisant sur les explications du chapitre précédent, nous ne pouvons pas établir de relation évidente entre un papillon FFT et un papillon Viterbi. Néanmoins, si nous posons $y_k^{[1]} = a + j.b$, $w_k^{[n]} = c + j.d$ et $y_k^{[0]} = a + j.b$, en prenant comme papillon

de référence de la FFT, le radix 2 et en développant les expressions du papillon de la FFT, nous obtenons :

$$y_k^{[0]} \pm y_k^{[1]}.w_n^k = (e + j.f) \pm (a + j.b).(c + j.d) = [e \pm (ac - bd)] + j.[f \pm (ad + bc)] \quad (4.1)$$

Nous remarquons que la procédure d'addition d'un papillon de la FFT se ramène à la nécessité de 6 opérations d'Additions/Soustractions. Deux premiers calculs de ac-bd et de ad+bc et quatre autres calculs de e ± ac-bd et f ± ad+bc. Ce qui fait que nous pouvons ramener la partie imaginaire et la partie réelle à trois opérations d'additions/soustractions chacune, tel que si par exemple, nous considérons la partie imaginaire qui traite les trois entrées E_i et les trois sorties S_i, nous obtenons le schéma de la figure 4.1, où S1 = E1 - (E3+E4), S2 = E3 + E4 et S3 = E1 + (E3+E4).

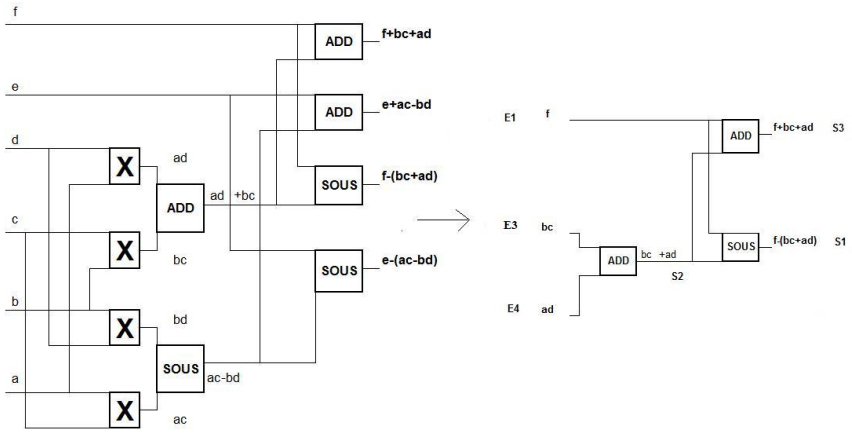

FIGURE 4.1 – Structure en Papillon de la FFT - Décomposée

4.2.1.2 Mutualisation des Méthodes : Papillon Viterbi

La représentation la plus usuelle des codes convolutifs est le diagramme en treillis. Bien que ce ne soit pas la plus compacte, cette forme est souvent utilisée car elle illustre bien le déroulement des algorithmes de décodage. La sortie du codeur dépend uniquement de son entrée et de l'état de son registre à décalage. Le diagramme en treillis (figure 3.20) représente ainsi l'ensemble des transitions possibles de l'état du codeur en fonction du temps. L'ensemble des états possibles du registre à décalage est reporté en ordonnée. En abscisse est représenté le temps. Les transitions en traits pleins correspondent à un élément binaire égal à '1' en entrée du codeur, et inversement, les transitions en pointillé représentent une entrée à '0'. On représente également la valeur des sorties du codeur au dessus de chaque transition. Ces valeurs dépendent de la transition considérée et des polynômes générateurs du code.

La Figure 3.20 ne représente le treillis que pour l'encodage d'un bit d'information, mais le motif du treillis se répète invariablement dans le temps à chaque nouvelle entrée

du codeur. D'après le principe de codage, les mots en sortie d'un codeur convolutif sont corrélés, puisque chaque mot est fonction de K entrées successives. En sortie du codeur, seules certaines séquences binaires sont possibles. Elles correspondent aux différents chemins qui existent dans le diagramme en treillis. Pour décoder une séquence reçue, il est nécessaire de la considérer dans son ensemble. Le décodage d'un code convolutif consiste à rechercher, dans le treillis, la séquence binaire la plus proche de la séquence reçue. Le décodeur estime donc la séquence la plus vraisemblable (most likelihood sequence estimation). Le nombre de séquences possibles étant généralement très important dans le treillis, l'application de cette règle de décodage est d'une complexité prohibitive. En résumé, le déroulement de l'algorithme de décodage est donc le suivant (A chaque instant de décodage t, pour chaque état du treillis) :

- calcul des métriques des deux transitions entrantes
- calcul des deux métriques chemins
- sélection du chemin survivant
- mémorisation du bit de décision en vue de restituer le décodage en fin de trame

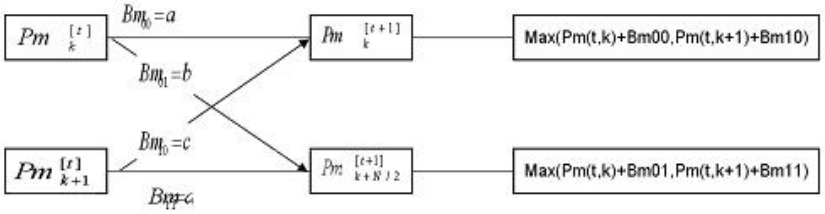

FIGURE 4.2 – Structure en Papillon du Viterbi

Nous pouvons faire apparaître le schéma de ce papillon en figure 4.2. Le papillon de Viterbi nécessite dans un premier temps le calcul des métriques, c'est à dire l'évaluation de la distance entre le message reçu et les différents messages théoriques. Une fois ce calcul effectué, les métriques de branches sont additionnées aux métriques de chemins et sont deux par deux comparées. Dans notre d'étude, nous considérons dans un premier temps uniquement le module ACS qui opère ce choix du chemin survivant. Ce module est composé de deux additioneurs et d'un opérateur de choix. Ce dernier peut être trivialement réalisé par la combinaison d'un soustracteur qui conditionne un multiplexeur à deux entrées. Ainsi, en considérant deux métriques de chemin P0 et P1 et une des deux paires de métriques de branches associées Bm01 et Bm11, nous pouvons définir un des deux opérateurs de choix du papillon Viterbi en figure 4.3. Ainsi, nous définissons les équation du besoin du papillon Viterbi : où S1 = E1 + E2, S2 = E1 + E2 - E5 - E6 et S3 = E5 +E6.

4.2.2 Premier Niveau de Granularité : Opérateur AS^3

A partir des deux décompositions précédentes, nous pouvons noter les caractéristiques importantes des deux papillons FFT et Viterbi (Les notations A, S et A/S cor-

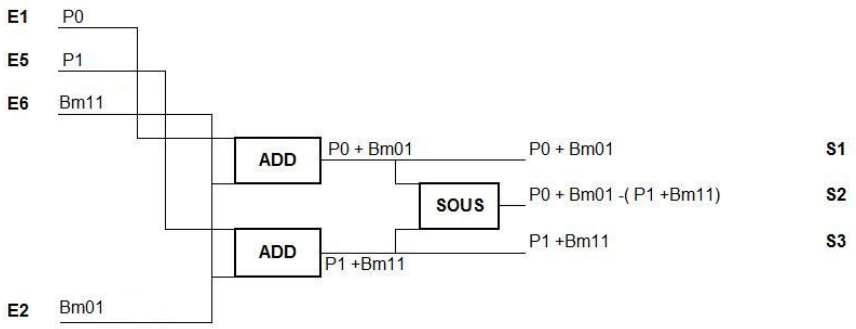

FIGURE 4.3 – Structure en Papillon du Viterbi - Décomposée

respondent à un additionneur, un soustracteur et un additionneur/soustracteur) :

- Le nombre identique de modules A/S mis en oeuvre : 6
- La répartition identique des modules A/S : 3x2
- La liaison systématique entre un même bloc A/S et deux autres blocs A/S

Nous résumons ces trois points en Figure 4.4. Etape par étape, nous pouvons identifier les connexions et les blocs A, S ou A/S nécessaires pour la FFT et le Viterbi seul ou les deux combinés. Finalement, nous posons les bases de notre opérateur générique proposé. Celui-ci met en oeuvre trois modules A/S. A partir du schéma de principe précédent, nous proposons en Figure 4.5, une schématique détaillée de l'opérateur que nous proposons nommé AS^3. Celui-ci comprend trois modules A/S et quatre multiplexeurs à deux entrées. Chacun de ces sept composants est paramétrable par un seul bit (sélection Addition ou Soustraction dans le cas des A/S, choix d'1 parmi 2 chemins de multiplexage dans le cas des Mux). L'équation reliant entrées et sorties d'un tel opérateur est donnée par :

$$\begin{aligned} S1 &= E1 + (-1)^{\alpha 1}.[\beta 1.E2 + (1-\beta 1).S2] \\ S2 &= [\beta 2.E3 + (1-\beta 2).S1] + (-1)^{\alpha 2}.[\beta 3.E4 + (1-\beta 3).S3] \\ S3 &= [\beta 4.E5 + (1-\beta 4).S2] + (-1)^{\alpha 2}.E6 \end{aligned}$$

Ainsi, pour les paramètres β (1,0,0,1) et α (0,1,0), nous retrouvons l'équation du Viterbi et pour (0,1,1,0) et (1,0,0) ou (1,1,0), nous retrouvons l'équation du module la FFT en posant E1=E6.

Il est à noter que chaque bloc A/S sera capable de traiter les données de types différents de la FFT et du Viterbi. En utilisant, une structure d'additionneur réel pour traiter les données de la FFT, la quantification demandera plusieurs bits (en général 16). Cette taille des données nous permet de traiter des données dures du Viterbi c'est-à-dire quantifiées sur 1 bit mais aussi des données " souples ", celles-ci quantifiées sur plus d'un bit (en général 4 à 6). Au-delà des opérations d'additions et de soustractions nécessaires pour le Viterbi et la FFT, notre opérateur est générique et peut réaliser trois

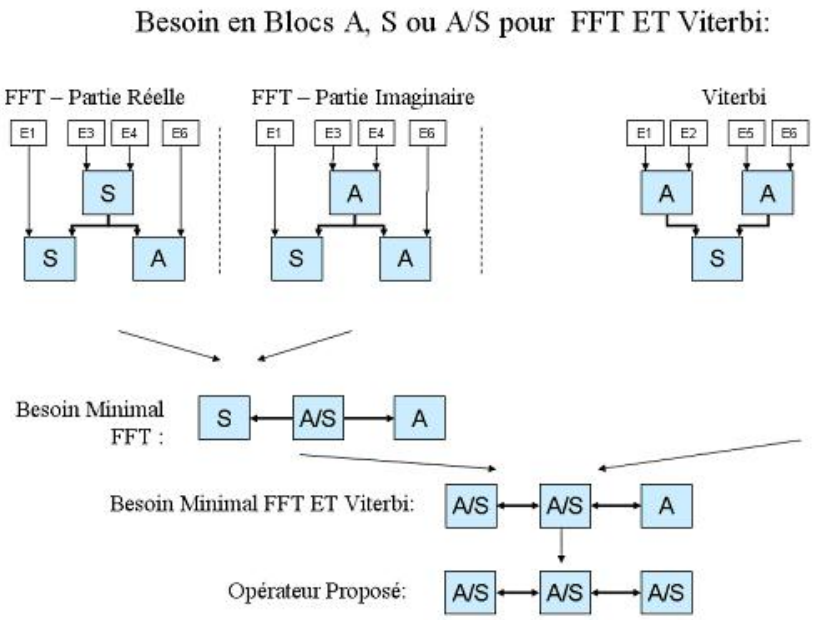

FIGURE 4.4 – Structure de l'Opérateur Commun AS^3 proposé

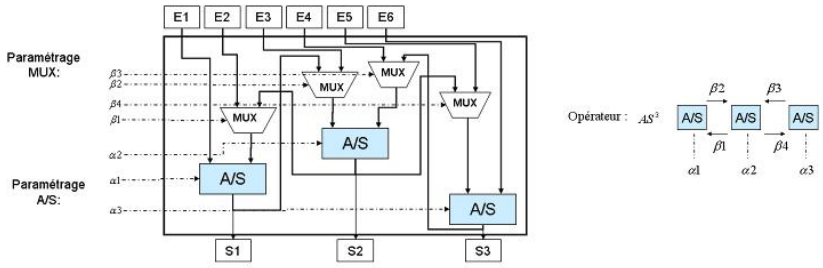

FIGURE 4.5 – Opérateur Commun AS^3 proposé

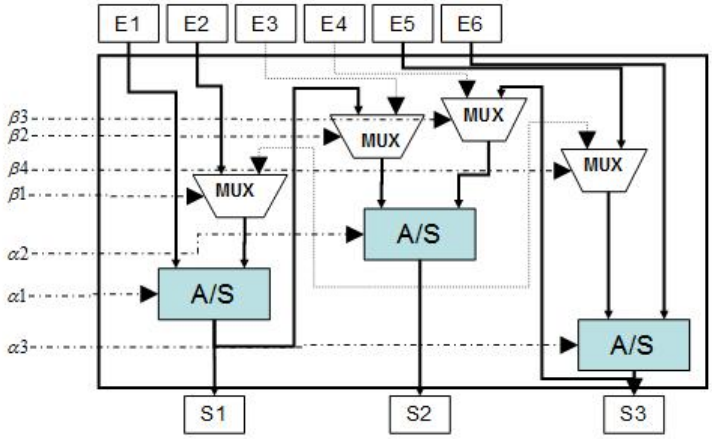

FIGURE 4.6 – Opérateur Commun AS^3 proposé - Paramétrisation Viterbi

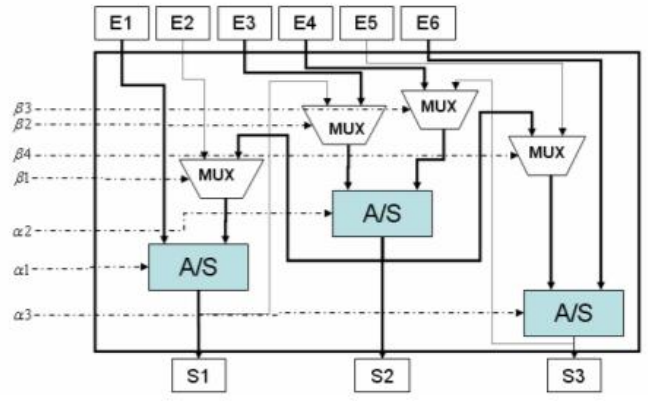

FIGURE 4.7 – Opérateur Commun AS^3 proposé - Paramétrisation FFT

opérations d'addition/soustraction indépendantes (Figure 4.8) ou une cascade de deux ou trois opérations d'additions/soustractions (Figure 4.9).

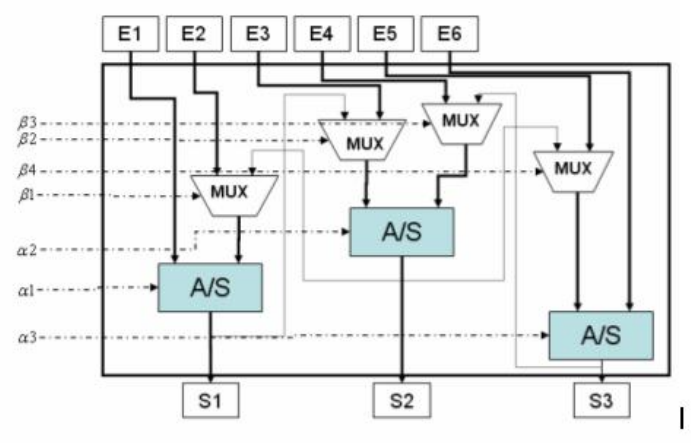

FIGURE 4.8 – Opérateur Commun AS^3 proposé - Paramétrisation Additionneurs Indépendants

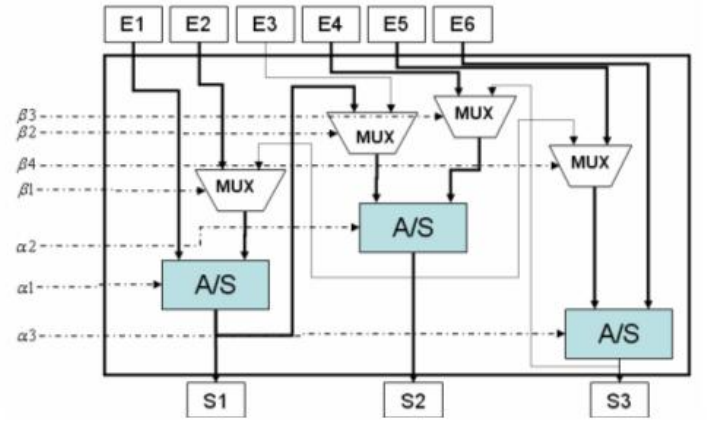

FIGURE 4.9 – Opérateur Commun AS^3 proposé - Paramétrisation Additionneurs en Cascade

4.2.3 Deuxième Niveau de Granularité : Opérateur FFT/Viterbi

Le premier opérateur que nous proposons permet de mutualiser les opérations d'A/S du papillon FFT et du module ACS du Viterbi. Nous étendons ici la factorisation des modules d'A/S au module BMC. Pour ce faire, nous nous focalisons sur ce dit module en rentrant dans les détails de conception afin de définir un besoin spécifique en A/S.

Ensuite, nous modifions le papillon FFT afin que ce dernier puisse tenir compte de ce besoin.

Tout d'abord, nous portons spécifiquement attention au module BMC, nous pouvons remarquer qu'il peut être implémenté de différentes manières selon la distance utilisée. En effet, l'objectif du module BMC est de déterminer le nombre d'"éléments" reçus, différents des bits théoriques potentiellement émis. Nous pouvons voir le problème de deux manières, soit la distance choisie conditionne le type des entrées soit le type des entrées conditionne la distance choisie. Comme nous l'avons mentionné avec l'application du Viturbo, l'usage est aujourd'hui de mettre en oeuvre un décodage de Viterbi à entrée souple pour le Turbo Décodage mais aussi pour les Codeurs Convolutifs NSC. Ce codage à entrées souples permet de considérer les valeurs d'entrées sur plus d'un bit et d'améliorer la quantification. L'entrée du décodeur est alors constituée par une suite d'échantillons. La distance pertinente à utiliser dans ce cas est la distance euclidienne. C'est à dire que nous considérons (s_t^1, s_t^2,...,s_t^r) le r-uplet d'éléments binaires en sortie du codeur a l'instant t (ou R=1/r est le rendement du codeur) ; (y_t^1, y_t^2,...,y_t^r) le r-uplet d'échantillons quantifiés correspondants, reçus en entrée du décodeur et (a_t^1, a_t^2,...,a_t^r) le r-uplet d'éléments binaires associe à une branche donnée du treillis. La distance euclidienne entre la valeur reçue et la valeur théorique associée à la transition est donnée par :

$$d_t = \sqrt{\sum_{i=1}^{r}(y_t^i - a^i)^2} = \sqrt{\sum_{i=1}^{r}(y_t^{i2} - 2.y_t^i.a^i + a^{i2}))} \tag{4.2}$$

Essayons de simplifier les considérations à prendre en compte avec une telle distance. En utilisant une modulation de phase, celle-ci associe à chaque élément binaire s_t^i à valeur dans l'alphabet (0 ;1), un symbole binaire a_t^i à valeur dans l'alphabet (-1 ;+1) selon la formule a_t^i=2.s_t^i-1. Ainsi, les valeurs théoriques des a^i associées à une transition sont à valeurs dans (-1 ;+1). Nous ne cherchons pas à évaluer concrètement la distance mais juste à comparer une succession d'additions de distances, nous pouvons considérer la distance au carré :

$$d_t^2 = \sum_{i=1}^{r}(y_t^i - a^i)^2 = \sum_{i=1}^{r}(y_t^{i2} - 2.y_t^i.a^i + a^{i2}) \tag{4.3}$$

Nous pouvons simplifier cette expressions car les termes au carré seront tous identiques et il nous restera seulement à évaluer dans chacun des cas : $\sum_{i=1}^{r} y_t^i.a^i$. Comme les différents a^i sont égaux à $\pm$ 1 alors, le calcul des métriques se résume à une somme pondérée ($\pm$ 1) des entrées. Maintenant, si nous considérons concrètement, le besoin en codeur convolutif de nos standards qui sont résumés dans le tableau de la figure 3.19, les rendements spécifiés sont soit 1/2 soit 1/3. Ceci correspond à traiter deux ou trois y_t^i à chaque papillon et correspond à opérer deux ou trois additions pour le calcul de chaque métrique, ce qui nécessite soit 4 soit 8 A/S pour chaque papillon.

Si maintenant, nous nous intéressons aux a_t^i, nous pouvons noter que dans le treillis

de Viterbi, les états qui sont reliés par un papillon sont des états qui ne changent qu'au niveau de leurs bits de poids faible (LSB), c'est à dire que c'est l'introduction du nouveau bit dans le registre à décalage qui modifie l'état du registre. Le bit de poids de fort est sorti du registre, les autres bits se décalent et le nouveau bit devient le LSB. Ainsi, l'état des registres aux "extrémités" d'un papillon ne se différencient que par le MSB à t-1 et LSB à t.

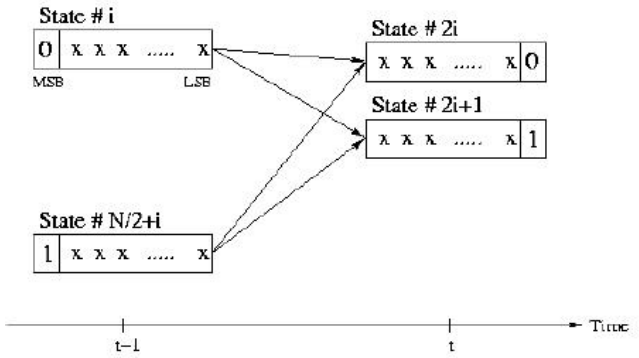

FIGURE 4.10 – MSB et LSB dans la compréhension du Papillon

De plus, pour un codeur convolutif de longueur de contrainte N+1, tous les polynômes générateurs des codeurs convolutifs sont de la forme $x^N + ... + 1$, c'est à dire que systématiquement le MSB au temps t-1 et le LSB au temps t sont additionnés modulo 2. Ainsi, nous pouvons en déduire que vis à vis d'un papillon :

- Les registres dont la somme MSB_{t-1} + LSB_t (modulo 2) est la même auront les valeurs des sorties des codeurs identiques et donc les même métriques de branches.
- Les registres dont la somme MSB_{t-1} + LSB_t (modulo 2) est complémentaire (modulo 2) auront des valeurs de sortie des codeurs complémentaires et donc les métriques de branches opposées.

Ainsi, pour un papillon, une seule métrique de branche sera à calculer, une autre sera identique et les deux autres opposées. Cette propriété réduit le nombre d'A/S requis de 4 et 8 à 1 et 2 pour chaque papillon.

Maintenant que nous avons ramené le calcul du module BMC à un maximum de 2 opérations A/S pour chaque papillon, nous pouvons modifier l'architecture du papillon FFT. Dans la section 4.2.2, nous identifions la papillon de la FFT comme un enchainement de quatre multiplications et six additions. Nous transformons cette décomposition en deux étapes en une architecture en trois étapes (figure 4.11) : Une première étape d'addition, une seconde de multiplication et la troisième de six additions. En effet, l'équation du papillon peut se ramener à une écriture :

$$y_k^{[0]} + y_k^{[1]}.w_n^k = [e + (c.(a+b) - b.(c+d))] + j.[f + (c.(a+b) + a.(d-c))] \quad (4.4)$$

$$y_k^{[0]} + y_k^{[1]}.w_n^k = [e + (c.(a+b) - b.(c+d))] + j.[f + (c.(a+b) + a.(d-c))] \quad (4.5)$$

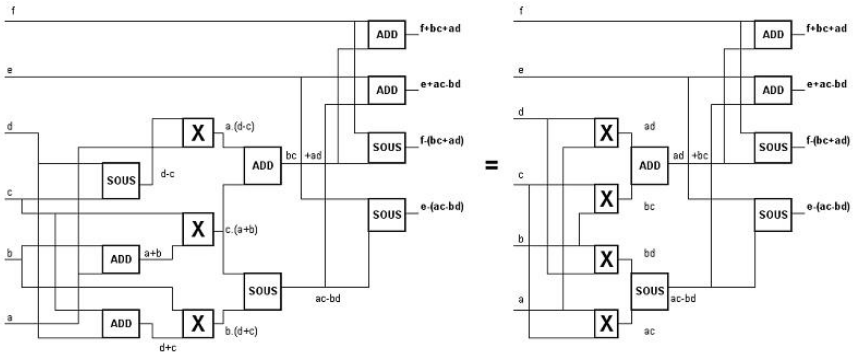

FIGURE 4.11 – Transformation de la FFT

Ainsi, en posant, A=a.(d-c), B=b.(c+d), C=c.(a+b), le troisième étage du papillon se ramène à la nécessité de 6 opérations d'Addition/Soustraction. Deux premiers calculs de AB= (A-B) et AC=(A+C) et quatre autres calculs de e ± AB et f ± AC. Nous remarquons, que la procédure d'addition de ce troisième étage de cette décomposition est équivalente à celle étudiée en 4.2.2 et remplaçable par l'OC AS^3. Nous transformons, donc, le calcul de la FFT de 4 multiplications et 6 additions à 9 additions et 3 multiplications (figure 4.11). Nous créons ainsi, un premier étage de calcul de bloc d'A/S supplémentaire. Ici, nous pouvons utiliser ce bloc ainsi crée pour opérer les opérations du module BMC. Comme le module possède trois A/S, nous pourrons effectuer jusqu'à un décodage Viterbi de codes de rendements 1/4. Nous proposons une schématique de notre opérateur FFT/Viterbi en figure 4.12. Ici, il est à noter que cet opérateur est "philosophiquement" différent du précédent AS^3. Ce dernier est un opérateur générique qui met en jeux une interconnexion spécifique de module A/S afin de réaliser une succession d'opérations d'additions et soustractions dont celles utiles pour la FFT et le module ACS du Viterbi. Ici, l'opérateur FFT/Viterbi n'a pas cette généricité là. Il doit être appréhender comme un papillon en Radix 2 de la FFT qui a été modifié pour réaliser les opérations d'un papillon de Viterbi. Ainsi, l'opérateur FFT/Viterbi sera commun à deux niveaux, d'un part d'un point de vue fonctionnel en raison des fonctions réalisables par la Transformée de Fourier Rapide et d'un point de vue architectural. Ainsi, en figure 4.12, le troisième étage est optimisé par rapport au module AS^3 en sachant que celui-ci peut être utilisé en remplacement.

4.3 Opérateurs traitant les opérations de Registres à Décalages

Dans le chapitre 3.2, nous donnons les définitions et les principales utilisations des registres à décalages et des registres à décalages à rétroactions linéaires, classés selon les fonctionnalités qu'ils peuvent réaliser. Chacune des fonctionnalités trouvées dans les standards peut être mise en oeuvre par des modèles architecturaux qui varient selon le type de codage, d'embrouillage.... L'absence de correspondance entre une architecture

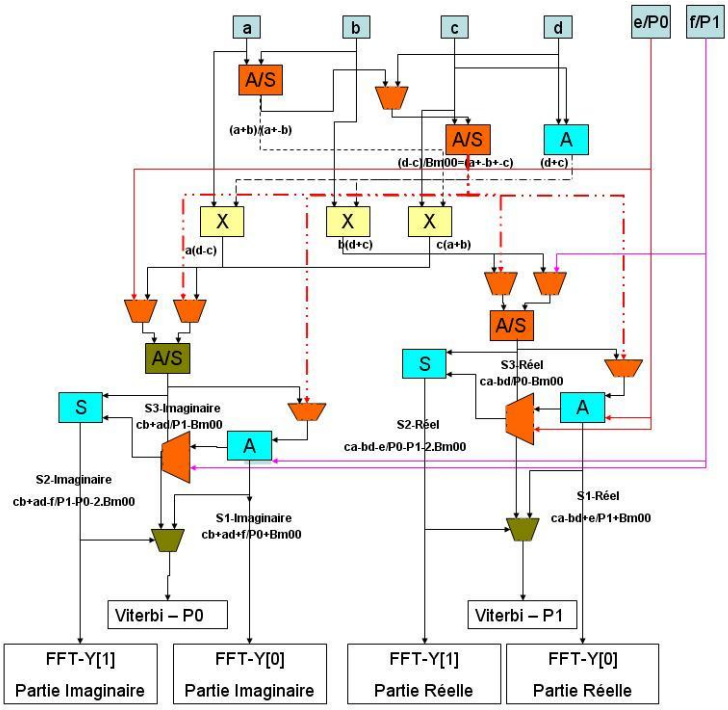

FIGURE 4.12 – Opérateur FFT/Viterbi

spécifique dédiée à une fonction précise est le coeur même de la technique des opérateurs communs qui vise à trouver des architectures "transversales" par rapport à des fonctionnalités distinctes. Néanmoins, des structures récurrentes apparaissent aux travers des différentes classes de fonctions rencontrées. Ainsi, les standards stipulent des architectures que nous qualifions de conventionnelles. Celles-ci obéissent à un modèle générique et ne se différencient que par la taille de leurs registres à décalages et leurs polynômes générateurs. D'autres sont plus complexes à mettre en oeuvre et bien qu'elles utilisent des registres à décalages, elles ne sont utilisées que dans des cas marginaux et nécessitent un agencement des registres qui n'est pas trivial. Ces structures marginales seront ici identifiées comme structures spécifiques. Nous rappelons au lecteur que le terme structure est employé pour définir des architectures mises en oeuvre dans une fonction. Dans le cas des LFSR, cette distinction est souvent très fine car les structures sur lesquelles nous travaillons peuvent constituer l'intégralité d'une fonction donnée.
Nous présentons dans cette partie, quatre architectures permettant de réaliser plus ou moins les opérations mentionnées au chapitre 3.2. Celles-ci divergent par leurs périmètres fonctionnels (figure 4.13), c'est à dire l'ensemble des opérations qu'ils peuvent réaliser. Néanmoins, dans ce cas précis des registres à décalages, les périmètres en question sont gigognes (figure 4.13). Etant donné que la réalisation des différentes opérations se répercutent d'un LFSR à l'autre, pour plus de simplicité, nous présentons les architectures de la moins complexe à la plus complexe. Ainsi, avec quatre registres à décalages, nous enrichissons le graphe des opérateurs communs et augmentons les possibilités de combinaisons des opérateurs communs.

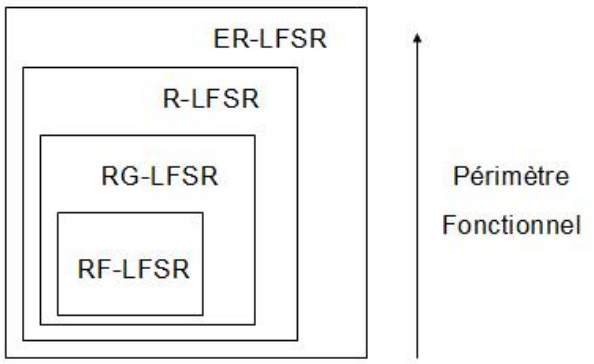

FIGURE 4.13 – Architecture du Scrambler du 802.11 mode OFDM

Les registres à décalages mis en oeuvre dans les exemples précédents différent sur trois grands points principaux :

1. Le nombre de boucles de retour ou d'insertion.
2. Le type de ces boucles (difference notable entre boucle de Galois et de Fibonacci).
3. La taille du registre requis.

Chaque standard va exécuter une même fonction mais avec un polynôme générateur différent. Comme la taille de ce polynôme influe sur la taille du registre à décalages, alors nous proposons dans ce qui suit, uniquement des architectures génériques pour une taille de registre N non déterminée. Concrètement, nous spécifierons dans cette section les architectures proposées par le nombre des boucles de retour et leurs types. Pour ce qui concerne le nombre de registres des OC LFSR, nous définirons celui-ci lors la phase d'implémentation en chapitre 5. La taille des OC proposés dépendra de la manière dont nous considérons l'application de la technique des opérateurs communs (avec réutilisation de l'OC ou en BOC).

4.3.1 Premier Niveau de Granularité : Reconfigurable Fibonacci LFSR (RF-LFSR)

En étudiant les différentes utilisations des registres à décalages pour des opérations binaires tel que l'embrouillage, le codage convolutif ou les CRC, nous nous apercevons vite, que trivialement aucune architecture n'apparait pouvant réaliser toutes les fonctionnalités ensembles ou individuellement. Cette hétérogénéité dans les architectures mises en oeuvre entre les fonctions ou pour une fonction donnée est le point faible de la technique des fonctions communes que nous essayons de résoudre par le biais des opérateurs communs dont le premier est le RF-LFSR (figure 4.14).

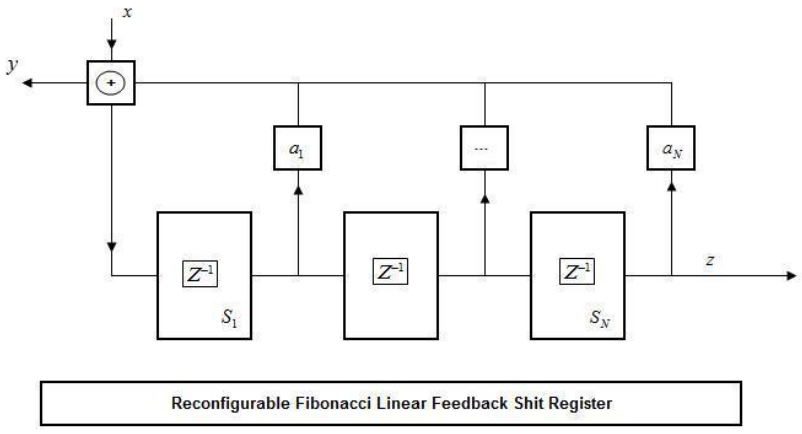

FIGURE 4.14 – Architecture du RF-LFSR

Définition du RF-LFSR. La première architecture commune que nous proposons est intitulée RF-LFSR pour Reconfigurable Fibonacci Linear Feedback Shift Register. Elle lie l'entrée x de l'architecture avec la sortie y tel que pour un registre de taille N à l'itération i :

$$y_i = \sum_{k=1}^{N} a_k \cdot y_{i-k} + x_i \tag{4.6}$$

Nous définissons une deuxième sortie de notre architecture par z tel que :

$$z_i = y_{i-N} = \sum_{k=1}^{N} a_k \cdot y_{i-k-N} + x_{i-N} \tag{4.7}$$

Comme les valeurs des registres ne sont pas modifiées mais seulement décalées, z correspond à la valeur de y à la Nième itération précédente. Nous définissons également l'état du registre à décalage S à l'itération i :

$$S_i = G_F.S_{i-1} + X_i \tag{4.8}$$

Le vecteur S_i représente les états des N registres $s_{j,i}$ à l'itération i et le vecteur X_i représente la valeur du vecteur d'entrée associé à chaque S_i tel que :

$$S_i = \begin{pmatrix} s_{(1,i)} \\ ... \\ s_{(j,i)} \\ ... \\ s_{(N,i)} \end{pmatrix} \qquad X_i = \begin{pmatrix} x_{(1,i)} \\ ... \\ x_{(j,i)} \\ ... \\ x_{(N,i)} \end{pmatrix} = \begin{pmatrix} x \\ 0 \\ 0 \\ ... \\ 0 \end{pmatrix}$$

La matrice G_F représente la matrice caractéristique de la boucle de rétroaction. Ici, seule la valeur $s_{(1,i)}$ est une combinaison linéaire des N $s_{(j,i-1)}$, les autres (N-1) $s_{(j,i)}$ n'étant que décalés par le registre à décalages. Ainsi, la matrice G_F se composera d'une première ligne caractéristique du polynôme générateur de la boucle de rétroaction et d'une sous matrice de taille (N-1) x N qui sera une matrice de Toeplitz significative du décalage. Par la suite, nous identifierons par g_{F1} le vecteur ligne représentatif de la première ligne de G_F et donc des coefficients du polynôme générateur de la rétroaction de Fibonacci tel que : $g_{F(1,j)} = a_j$.

$$G_F = \begin{pmatrix} g_{F1} \\ ... \\ g_{Fj} \\ ... \\ g_{FN} \end{pmatrix} = \begin{pmatrix} g_{F(1,1)} & ... & g_{F(1,k)} & ... & g_{F(1,N)} \\ ... & ... & ... & ... & ... \\ g_{F(j,1)} & ... & g_{F(j,k)} & ... & g_{F(j,N)} \\ ... & ... & ... & ... & ... \\ g_{F(N,1)} & ... & g_{F(N,k)} & ... & g_{F(N,N)} \end{pmatrix}$$

$$G_F = \begin{pmatrix} g_{F1} \\ ... \\ g_{Fj} \\ ... \\ g_{FN} \end{pmatrix} = \begin{pmatrix} g_{F(1,1)} & ... & g_{F(1,k)} & ... & g_{F(1,N)} \\ 1 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & 0 \\ 0 & 0 & ... & ... & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} g_{F1} \\ --- \\ Matrice de \\ Toeplitz \\ --- \end{pmatrix}$$

Ainsi, nous pouvons définir par récurrence les valeurs de S_i vis à vis du vecteur initial S_0. Nous donnons en Annexe E, un procédé simplifié de calcul des puissances de G_F.

$$S_i = G_F^i.S_0 + \sum_{p=0}^{i} G_F^{i-p}.X_p \tag{4.9}$$

Architecture du RF-LFSR. x et y représentent respectivement l'entrée série et la sortie du RF-LFSR prise au niveau du module d'addition binaire modulo 2. Ce module permet d'additionner les N valeurs de la boucle de retour et peut être vu comme composé de (N-1) XOR à deux entrées. Nous définissons également, z, la sortie série du RF-LFSR qui est une vision de y mais avec un retard de N itérations. Concrètement, cette architecture reprend la structure de la boucle de rétroaction d'un LFSR de Fibonacci (Annexe D) dont nous posons les coefficients a_j comme paramétrables. Ainsi, vis à vis de la structure du LFSR de Fibonacci, nous ajoutons la possibilité d'insérer une entrée x dans le registre à décalage. La sortie y représentera l'addition modulo 2 entre x et la valeur de la boucle de rétroaction du LFSR. Le registre contiendra N valeurs de y de y_{i-1} à y_{i-N} pour l'entrée x_i. D'un point de vue implémentation le RF-LFSR (Figure 4.14) sera donc définie par :

- Une entrée Binaire x
- Une entrée sur N bits pour paramétrer les coefficients a_k
- Une entrée sur N bits pour insérer la séquence initiale du LFSR
- Une sortie Binaire y
- Une sortie Binaire z

Utilisation du RF-LFSR. Vis à vis des opérations requises dans les normes et décrites précédemment, l'utilisation principale du RF-LFSR sera en tant que générateur de séquences aléatoires. La structure de Fibonacci permet une implémentation triviale des générateurs de séquences de type LFG ou LFSR par simple paramétrage des coefficients a_j par les polynômes générateurs adéquats. Le RF-LFSR peut donc être utilisé dans l'embrouillage et le désembrouillage de telles séquences. De plus, le possibilité d'insérer directement des données dans le LFSR de Fibonacci par le biais de l'entrée x, nous permet de récréer facilement des scramblers de séquences auto-synchronisées, comme dans le cas du 802.11 High Rate DSSS.

Conclusion sur le RF-LFSR. Bien que défini comme un opérateur dédié à l'embrouillage, certaines de ces opérations (figure 3.3) ne peuvent pour autant être réalisées via le RF-LFSR (séquences de Gold du 3GPP LTE ou désembroillage du scramblers de séquences auto-synchronisées du 802.11 High Rate DSSS). Outre l'embrouillage, les différents types de codages ne sont pas traités par ce premier OC, nous proposons par conséquent, un deuxième OC de même granularité.

4.3.2 Premier Niveau de Granularité : Reconfigurable Galois LFSR (RG-LFSR)

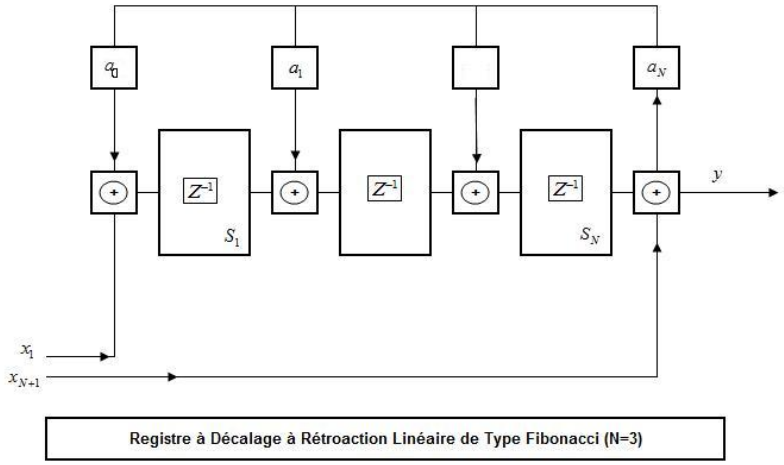

FIGURE 4.15 – Architecture du RG-LFSR

Définition du RG-LFSR. La deuxième architecture commune que nous proposons est intitulée RG-LFSR pour Reconfigurable Galois Linear Feedback Shift Register. Elle lie l'entrée x de l'architecture avec la sortie y tel que pour un registre de taille N à l'itération i :

$$y_i = \sum_{k=1}^{N} a_{N-k} \cdot y_{i-k} + x_{(1,i-N)} + x_{(N+1,i)} \tag{4.10}$$

Nous définissons également l'état du registre à décalage S à l'itération i :

$$S_i = G_G.S_{i-1} + X_i \tag{4.11}$$

Le vecteur S_i représente les états des N registres $s_{j,i}$ à l'itération i et le vecteur X_i représente la valeur du vecteur d'entrée associé à chaque $s_{j,i}$ tel que :

$$S_i = \begin{pmatrix} s_{(1,i)} \\ \dots \\ s_{(j,i)} \\ \dots \\ s_{(N,i)} \end{pmatrix} \qquad X_i = \begin{pmatrix} x_{(1,i)} \\ \dots \\ x_{(j,i)} \\ \dots \\ x_{(N,i)} \end{pmatrix} = \begin{pmatrix} x_{(1,i)} + g_{G(1,N)}.x_{(N+1,i)} \\ g_{G(2,N)}.x_{(N+1,i)} \\ \dots \\ \dots \\ g_{G(N,N)}.x_{(N+1,i)} \end{pmatrix}$$

La matrice G_G représente la matrice caractéristique de la boucle de rétroaction de type Galois. Ici, toutes les valeurs $s_{(j,i)}$ sont dépendantes de la valeur $s_{(j-1,i-1)}$ et de la valeur $s_{(N,i-1)}$. Ainsi, la matrice G_G se composera d'une première sous matrice

de Toeplitz caractéristique du décalage et d'une deuxième sous matrice (la dernière colonne), notée g_{GN}, caractéristique du polynôme générateur de la boucle de rétroaction.

$$G_F = \begin{pmatrix} g_{G1} \\ ... \\ g_{Gj} \\ ... \\ g_{GN} \end{pmatrix} = \begin{pmatrix} g_{G(1,1)} & \cdots & g_{G(1,k)} & \cdots & g_{G(1,N)} \\ ... & ... & ... & ... & ... \\ g_{G(j,1)} & \cdots & g_{G(j,k)} & \cdots & g_{G(j,N)} \\ ... & ... & ... & ... & ... \\ g_{G(N,1)} & \cdots & g_{G(N,k)} & \cdots & g_{G(N,N)} \end{pmatrix}$$

$$G_G = \begin{pmatrix} g_{G1} \\ ... \\ g_{Gj} \\ ... \\ g_{GN} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & g_{G(1,N)} \\ 1 & 0 & 0 & ... & ... \\ 0 & 1 & 0 & ... & g_{G(j,N)} \\ 0 & 0 & ... & ... & ... \\ 0 & 0 & 0 & 1 & g_{G(N,N)} \end{pmatrix}$$

Ainsi, nous pouvons définir par récurrence les valeurs de S_i vis à vis du vecteur initial S_0

$$S_i = G_G^i.S_O + \sum_{p=0}^{i} G_G^{i-p}.X_p \tag{4.12}$$

Architecture et utilisation du RG-LFSR. Comme le RF-LFSR reprenait l'architecture la plus utilisée pour générer des séquences pseudo aléatoires, le RG-LFSR reprend l'architecture générique utilisée pour définir le codage des CRC. Ainsi, vis à vis des explications sur le calcul des codes cycliques binaires et plus précisément des CRC, nous définissons une architecture générique permettant de réaliser la division dans GF(2). En 3.2.2, nous détaillons le principe des CRC et expliquons que le codage des N bits de parité rajoutés au message pour former le mot code comme le calcul des syndromes permettant de détecter les erreurs vis à vis du message entrant où le calcul se fait par le biais du division polynomiale dans GF(2). En 3.2.2, nous spécifions l'architecture permettant de réaliser ces opérations. Nous reprenons ici cette architecture sous une forme générique paramétrable. Le RG-LFSR se définit donc par :

- Une entrée Binaire $x_{(N+1)}$
- Une entrée Binaire $x_{(1)}$
- Une entrée sur N+1 bits pour paramétrer les coefficients a_k
- Une entrée sur N bits pour insérer la séquence initiale du LFSR
- Une sortie Binaire y
- Une sortie sur N bits pour récupérer la valeur des registres.

Conclusion sur le RG-LFSR. Tel que défini dans le paragraphe précédent, le RG-LFSR est présenté initialement comme un diviseur polynomiale dans GF(2), permettant de réaliser le calcul des CRC pour un polynôme générateur donné. Néanmoins, dans notre recherche d'opérateurs communs, notre objectif est d'étendre au maximum l'utilisation de nos architectures et ne

pas les dédier à un type d'opérations. Ainsi, dans le cas d'étude de la génération de séquences aléatoires, nous pouvons transposer les calculs réalisables par le RF-LFSR au RG-LFSR. Pour ne pas surcharger ce chapitre du rapport nous détaillons l'utilisation du RG-LFSR en RF-LFSR en Annexe E. Nous précisons deux points essentiels : le passage d'une architecture à l'autre (Annexe E.2.1) et d'une séquence initiale à l'autre (Annexe E.2.2). Ainsi, le RG-LFSR présente la particularité de pouvoir réaliser les mêmes opérations que le RF-LFSR. Néanmoins, ces opérations basées sur les LFSR ne concernent concrètement que les opérations de calcul des CRC et de générations de séquences aléatoires de type LFG et LFSR. Les générations de séquences plus évoluées comme les codes GOLD ou les opérations de désembrouillage des scramblers auto-synchronisés ne peuvent être menées à bien par l'une ou l'autre de ces architectures. De plus, l'ensemble des codeurs convolutifs ne sont pas non plus pris en compte. Le RF et le RG-LFSR ne permettent pas de factoriser une partie des opérations à bases de registres à décalages que nous avons énumérées dans les tableaux 3.3, 3.10 et 3.19. Nous proposons par conséquent, un troisième OC.

4.3.3 Deuxième Niveau de Granularité : Reconfigurable LFSR (R-LFSR)

Nous proposons une troisième architecture commune capable d'étendre le gamme des opérations réalisables par les LFSR aux codages convolutifs tout en reprenant les possibilités d'implémentations des RF et RG-LFSR. Cet opérateur nommé Reconfigurable LFSR se base sur un système IIR dans sa forme transposée (Annexe D), rapportée au calcul dans GF(2) (Figure 4.16).

Définition du R-LFSR. La troisième architecture commune que nous proposons, reprend l'architecture d'un système de filtre IIR dans sa forme transposée où les calculs s'opèrent dans GF(2). Ainsi, dans GF(2) addition et soustraction se confondent, ce qui donne la relation liant l'entrée série x et la sortie série y tel que pour un registre de taille N à l'itération i :

$$y_i = \sum_{k=0}^{m=N} b_{N-k} \cdot x_{i-k} + \sum_{k=1}^{m=N} a_{N-k} \cdot y_{i-k} \tag{4.13}$$

Le R-LFSR peut se voir comme la répétition de N fois la même cellule dite cellule en "E". La cellule en E est composée d'un registre et d'une unité de calcul à cinq entrées (Figure 4.17). Pour les entrées A, B, C, a_N et b_N, la sortie D de la cellule en E est égale à :

$$D = A + B \cdot b_N + C \cdot a_N \tag{4.14}$$

Nous définissons également la relation de récurrence entre les valeurs des sorties parallèles de l'architecture. Celles-ci correspondent à l'état du registre à décalages S_i à

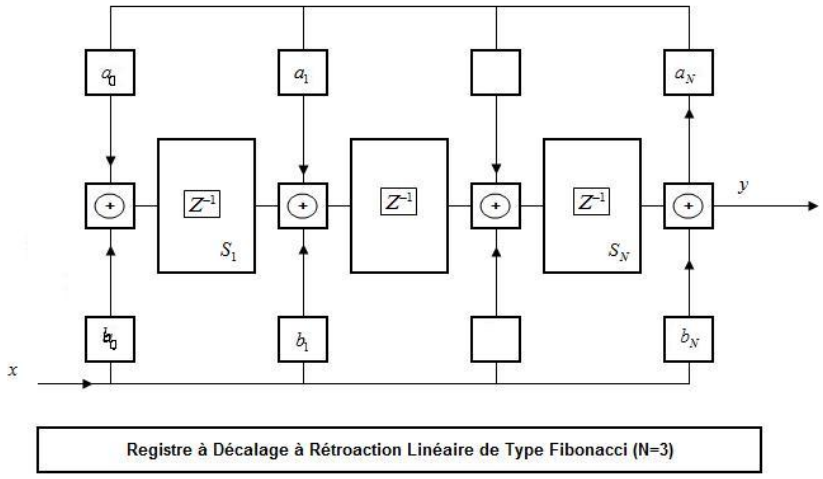

FIGURE 4.16 – Architecture du R-LFSR

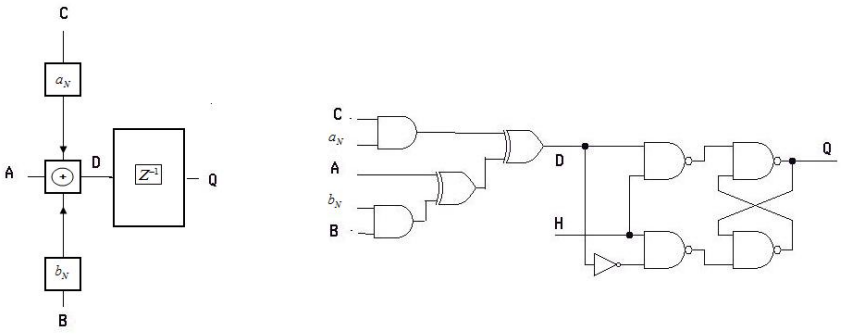

FIGURE 4.17 – Architecture de la cellule en E du R-LFSR

l'itération i.

$$S_i = G_R.S_{i-1} + T_i \tag{4.15}$$

Le vecteur S_i représente les N états $s_{j,i}$ du registre à décalage.

$$S_i = \begin{pmatrix} s_{(1,i)} \\ ... \\ s_{(j,i)} \\ ... \\ s_{(N,i)} \end{pmatrix},\ T_i = \begin{pmatrix} t_{(1,i)} \\ ... \\ t_{(j,i)} \\ ... \\ t_{(N,i)} \end{pmatrix}$$

La matrice G_R caractérise la rétroaction de type de Galois qui associe à chaque $s_{j,i}$, $s_{j-1,i-1} + a_{j-1}.s_{N,i-1}$. Ainsi, la matrice G_R sera équivalente à la matrice G_G du RG-LFSR (4.3.2). Elle se compose d'une première sous matrice de Toeplitz caractéristique du décalage et d'une deuxième sous matrice (la dernière colonne), caractéristique du polynôme générateur de la boucle de rétroaction. Comme en 4.3.2, les coefficients a_j du polynôme générateur de la boucle de Galois seront représentés par le vecteur g_{GN}, tel que $g_{G(j,N)} = a_{j-1}$.

$$G_R = G_G = \begin{pmatrix} g_{G1} \\ ... \\ g_{Gj} \\ ... \\ g_{GN} \end{pmatrix} = \begin{pmatrix} g_{G(1,1)} & \cdots & g_{G(1,k)} & \cdots & g_{G(1,N)} \\ ... & ... & ... & ... & ... \\ g_{G(j,1)} & \cdots & g_{G(j,k)} & \cdots & g_{G(j,N)} \\ ... & ... & ... & ... & ... \\ g_{G(N,1)} & \cdots & g_{G(N,k)} & \cdots & g_{G(N,N)} \end{pmatrix}$$

$$G_R = \begin{pmatrix} g_{G1} \\ ... \\ g_{Gj} \\ ... \\ g_{GN} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & g_{G(1,N)} \\ 1 & 0 & 0 & ... & ... \\ 0 & 1 & 0 & ... & g_{G(j,N)} \\ 0 & 0 & ... & ... & ... \\ 0 & 0 & 0 & 1 & g_{G(N,N)} \end{pmatrix}$$

Le vecteur T_i représente les N entrées associées à chaque registre, formées à partir des (N+1) versions de l'entrée série x pondérée par les coefficients b_i. Concrètement chaque entrée d'une cellule j reçoit, une version de x pondérée par b_{j-1} plus une deuxième version de x due à la boucle de rétroaction. Celle-ci est doublement pondérée par b_N et a_{j-1}. Nous considérons le vecteur B de dimension (1 x N+1) caractéristique des b_i, ce qui nous conduit à définir le vecteur T_i

$$T_i = x \cdot B^T + x \cdot b_N \cdot G_{GN} = \begin{pmatrix} x \cdot b_0 + x \cdot b_N \cdot g_{G(1,N)} \\ x \cdot b_1 + x \cdot b_N \cdot g_{G(2,N)} \\ ... \\ ... \\ x \cdot b_{N-1} + x \cdot b_N \cdot g_{G(N,N)} \end{pmatrix}$$

Ainsi, nous pouvons définir par récurrence les valeurs de S_i vis à vis du vecteur initial S_0

$$S_i = G_G^i.S_O + \sum_{p=0}^{i} G_G^{i-p}.T_p \tag{4.16}$$

Choix d'une Forme Transposée de Filtre IIR. Vis à vis des deux précédents LFSR et des besoins présentés en 3.2, nous devons développer une structure capable d'opérer des opérations de codages tel que mentionné en 3.2.3. Les différentes architectures présentées en 3.2.3, ne nécessitent pas systématiquement des registres à décalages à rétroactions linéaires :

- Chaque sortie des codeurs NSC 1/n (3.2.3), peut être vue comme la sortie d'un filtre FIR, toutes basées sur le même registre à décalage.
- Les codeurs RSC quant à eux présentent une boucle de rétroaction mais l'équation régissant la ou leurs sorties est celle d'un filtre IIR (D.1).
- Les Codeurs Convolutionnels de type k/n sont quant à eux plus complexes et nécessitent n sorties obtenues à partir de k.n polynômes générateurs et de k registres à décalages. Néanmoins, comme expliqué en 3.2.3, chacune de ces sorties peut être vue comme la combinaison linéaire de différentes sorties de filtres FIR.

A partir de ces explications, nous devons compléter notre gamme d'opérateurs communs par une architecture capable de satisfaire aux besoins en FIR et autres IIR. Nous présentons en annexe D.1, les trois formes génériques des filtres IIR, (1) Forme Direct 1, (2) Forme Directe 2 et (3) Forme Transposée.
Parmi, ces trois architectures, nous choisissons de développer une architecture commune basée sur la Forme Transposée qui sera appelée Reconfigurable Linear Feedback Shift Registers (R-LFSR). Le choix de cette troisième forme n'est pas fortuite par rapport aux deux autres. En effet,

- Vis à vis de la Forme "Directe 1", les formes "Directe 2" et "Transposée" présentent deux fois moins de registres à implémenter ce qui allège la structure à considérer.
- Vis à vis de la Forme "Directe 2", la "Forme Transposée" possède l'avantage de présenter dans son architecture, un système de rétroaction équivalent à celui de Galois. Comme nous cherchons un opérateur commun, nous voulons maximiser les opérations qu'il peut réaliser. De part son architecture de Galois, la "Forme Transposée" devient capable de générer les éléments du corps de Galois et d'opérer les divisions polynomiales dans GF(2), celles-ci étant nécessaires au calcul des CRC. Nous retrouvons ici, la même différence fondamentale entre "Forme Directe 2" et "Forme Transposée" qu'entre le type de Fibonacci et de Galois. Même si les deux parviennent à obtenir les même sorties séries pour une même entrée série, les valeurs de leurs registres respectifs ne sont pas identiques. Or dans ce nombreux calculs (3.2.2), nous nous intéressons aux sorties parallèles de nos architectures.

Ainsi, nous développons l'architecture du R-LFSR (figure 4.16). Il est à noter que dans notre cas d'étude, nous travaillons sur des opérations de GF(2), c'est à dire qu'addition et soustraction sont équivalentes et réalisées par un XOR, d'où le changement

de signe dans l'équation de 4.13 avec celle de l'Annexe D.1. Le R-LFSR se définit donc par :

- Une entrée série x.
- Une première entrée parallèle sur N+1 bits pour paramétrer les coefficients a_k.
- Une deuxième entrée parallèle sur N+1 bits pour paramétrer les coefficients b_k.
- Une troisième entrée parallèle sur N bits pour insérer la séquence initiale du LFSR.
- Une sortie série y.
- Une sortie parallèle sur N bits pour récupérer la valeur des registres.

Utilisation du R-LFSR Le R-LFSR est donc défini à partir d'une architecture de Filtre IIR. Comme illustré en Figure 4.16, il peut être présenté comme la concaténation d'un LFSR de Galois et d'une structure en Filtre FIR dans sa forme Transposée. Comme il existe une forme transposée du filtre IIR, il existe une forme transposée du filtre FIR. Le passage d'une forme à l'autre se fait simplement par "inversion" des coefficients comme dans le cas du LFSR de type Fibonacci au LFSR de type de Galois. En effet, pour le type FIR dans sa forme directe, nous avons :

$$y_j = \sum_{k=0}^{N} b_k \cdot x_{j-k} \tag{4.17}$$

Pour le type FIR en forme transposée :

$$y_j = \sum_{k=0}^{N} b_{N-k} \cdot x_{j-k} \tag{4.18}$$

Ainsi, la correspondance s'obtient en intervertissant les coefficients du polynôme générateur.

Cas d'un codage NSC 1/n. De part sa structure en FIR, le R-LFSR peut réaliser les différents codeurs NSC. Comme présenté en 3.2.3, chaque sortie d'un codeur convolutif est défini par un l'équation d'un filtre FIR. Ainsi, l'équivalent avec le R-LFSR sera obtenu par le R-LFSR de même taille mais avec les coefficients de la boucle de rétroaction nuls et ceux de la boucle d'insertion "inversés" par rapport à ceux du code convolutif. Nous donnons un exemple en figure 4.18. Nous implémentons un codeur de taille 3 et de polynôme générateur $x^3 + x^2 + 1$. L'équivalent obtenu par le R-LFSR est obtenu par le paramétrage à zero de tous les coefficients de la boucle de retour et avec $b_i = [1, 1, 0, 1]$. Nous illustrons également en figure 4.18, l'équivalence des premiers résultats obtenus avec MATLAB.

Ainsi, nous devrons dupliquer n fois le R-LFSR, pour un coder NSC 1/n afin d'obtenir les n sorties séries pour la même entrée série qui sera distribuée sur les n architectures.

Cas d'un codage RSC. Contrairement au codage NSC précédent, le codage RSC nécessite une boucle de rétroaction. Chaque sortie du codeur étant une fonction

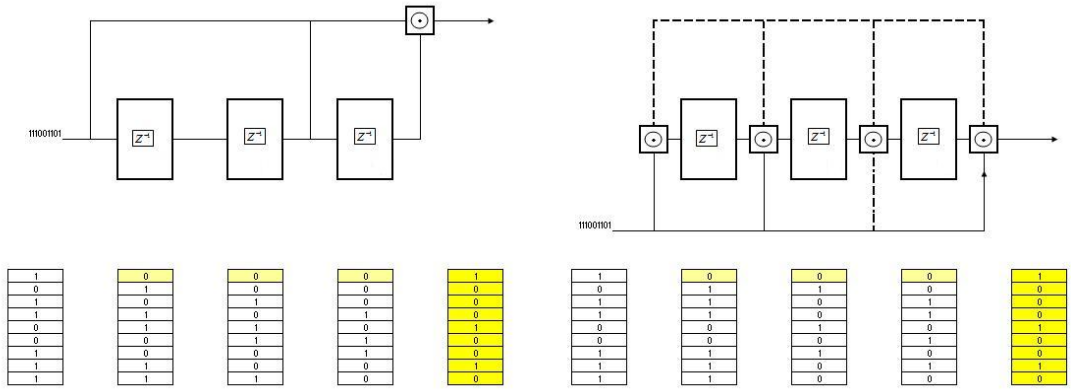

FIGURE 4.18 – Application du R-LFSR pour le codeur Convolutif $x^3 + x^2 + 1$

des N entrées et sorties précédentes. Comme précédemment, pour réaliser le codage d'un codeur RSC dont les coefficients des polynômes générateurs sont définis par rapport à la structure en Forme Directe 2 du filtre IIR, nous devons "inverser" ces coefficients pour le paramétrage du R-LFSR. Nous prenons en figure 4.19, l'exemple du codeur RSC de taille 3 utilisé dans le cas du turbo codage du 3GPP LTE (3.19). Ce turbo codeur possède deux polynômes générateurs $x^3 + x + 1$ pour la boucle de retour et $x^3 + x^2 + 1$ pour la boucle de sortie. Ainsi, le paramétrage équivalent pour le R-LFSR est donné par a_i = [1,1,0,1] et b_i = [1,0,1,1]. Nous illustrons également en figure 4.19, l'équivalence des premiers résultats obtenus avec MATLAB.

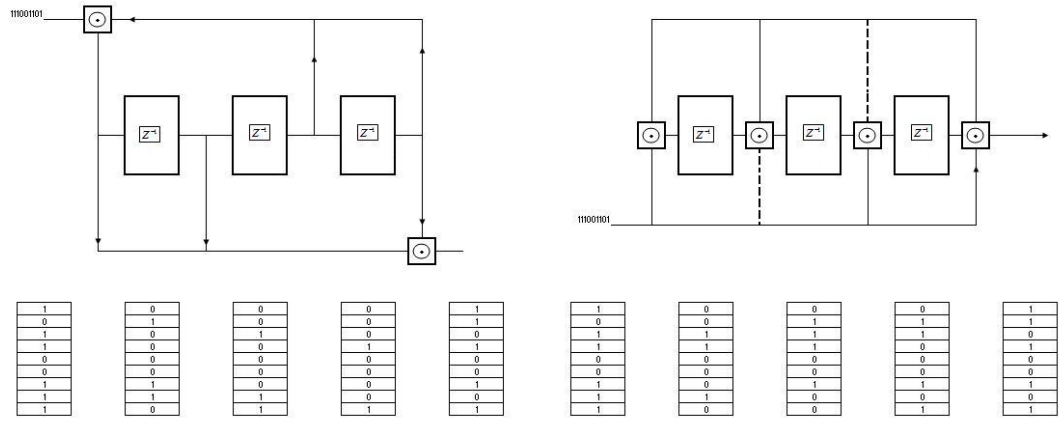

FIGURE 4.19 – Application du R-LFSR pour le codeur Convolutif $x^3 + x + 1/x^3 + x^2 + 1$

Cas d'un codage NSC k/n Nous expliquons le principe des codages NSC de rendement k/n en 3.2.3. L'équation 3.11 régit la relation entre les k entrées e_i et les n sorties s_i. Si nous définissons les matrices E des entrées, S des sorties et G des k.n polynômes générateurs du code alors nous avons l'équation :

$$S = G.E,\ G = [g_{ji}] \tag{4.19}$$

Pour mieux expliquer cette relation et l'implémentation sous-jacente requise par le R-LFSR, nous prenons comme exemple le codeur NSC 2/3 du mode ERP du 802.11. Ce codeur possède 2 entrées et trois sorties avec comme indiqué en 3.2.3, k.v = k.(L-1) = 2 x 4 = 8 registres.

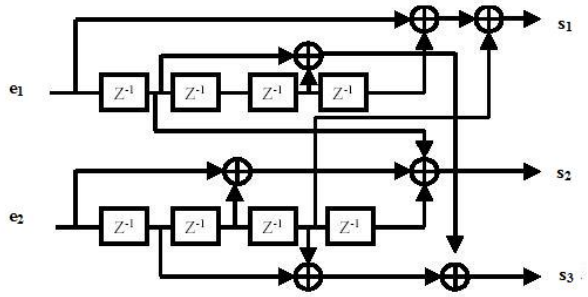

FIGURE 4.20 – Exemple de Codeur NSC 2/3 - Norme 802.11 ERP

Dans ce cas précis, la matrice G devient égale à :

$$\begin{pmatrix} g_{11}, g_{12} \\ g_{21}, g_{22} \\ g_{31}, g_{32} \end{pmatrix} \quad avec \quad \begin{cases} g_{11} = x^4 + 1 & et & g_{12} = x^3 \\ g_{21} = x & et & g_{22} = x^4 + x^2 + 1 \\ g_{31} = x^3 + x & et & g_{32} = x^{32} + x \end{cases}$$

Nous avons donc concrètement :

$$\begin{cases} s_{1(t)} = e_{1(t)} + e_{1(t-4)} + e_{2(t-3)} \\ s_{2(t)} = e_{1(t-1)} + e_{2(t)} + e_{2(t-4)} + e_{2(t-2)} \\ s_{3(t)} = e_{1(t-1)} + e_{1(t-3)} + e_{2(t-1)} + e_{2(t-3)} \end{cases}$$

Ainsi, chaque multiplication polynomiale de l'équation 3.11 de la forme $g_{ji}.e_i$ peut être mise sous la forme :

$$g_{ji} \cdot e_i(t) = \sum_{k=1}^{N} g_{(ji)(k)} \cdot e_{i(t-k)} \tag{4.20}$$

Ainsi chacune des multiplications représentent l'équation du système FIR d'une sortie d'un codeur NSC, à la différence que les sorties obtenues doivent être additionnées selon les couples (ji).

Le R-LFSR peut donc être utilisé pour chaque codeur de polynôme g_{ji}. Nous aurons donc besoin d'autant de R-LFSR que de polynômes, c'est à dire k.n R-LFSR. Nous présentons cette architecture dans le cas précédent du codeur NSC 2/3 du mode 802.11 ERP. La figure 4.21 représente la décomposition en k.n=6 codeurs convolutionnels interconnectés et l'équivalence obtenue avec les R-LFSR.

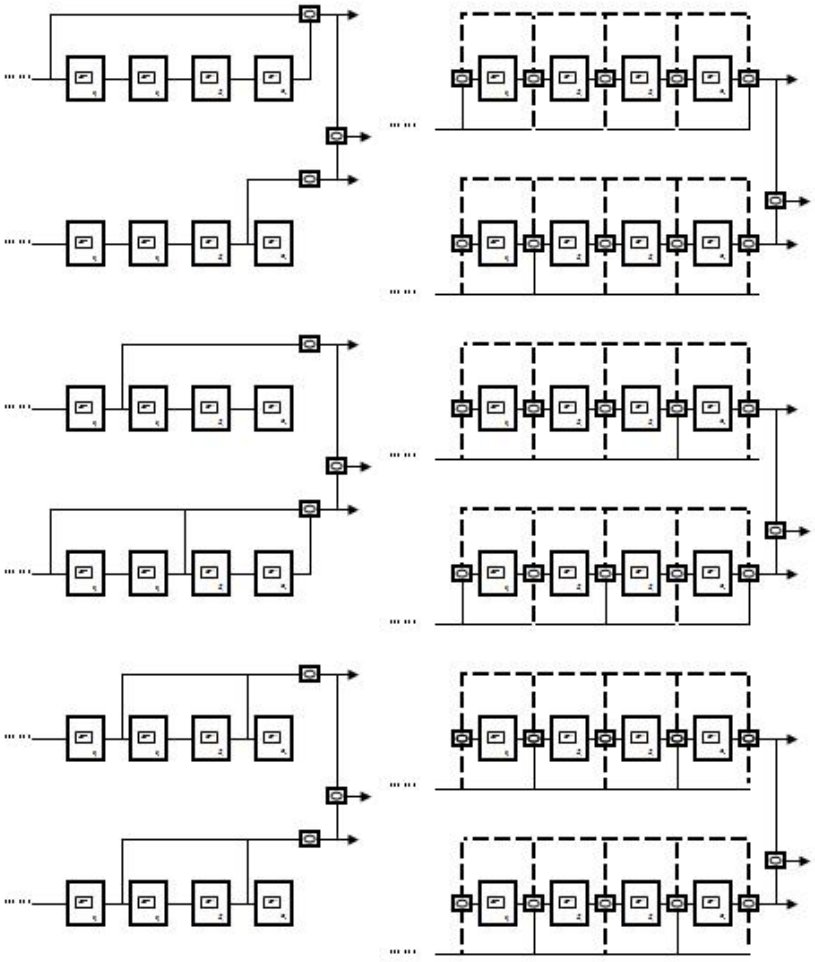

FIGURE 4.21 – Application du R-LFSR pour le codeur NSC 2/3

Conclusion sur le R-LFSR **Parmi toutes les structures identifiées, le R-LFSR permet de réaliser les différents codages convolutifs et en reprenant l'architecture du RG-LFSR, il est capable d'exécuter les opérations de CRC et d'embrouillage. Seuls deux cas particuliers restent à traiter. La génération des séquences de GOLD et le turbo codage utilisant un Codeur Récursif m-Circulaire (m-CRSC). Pour le premier, nous donnons en Annexe E.3, une**

explication de la génération de séquences identiques entre un R-LFSR et un système IIR, utilisé par les codes GOLD. Quant aux codeurs m-CRSC, ils nous obligent à définir une quatrième architecture de granularité supérieure afin de pouvoir factoriser 100% des besoins en LFSR.

4.3.4 Troisième Niveau de Granularité : Extended R-LFSR (ER-LFSR)

Dans la section précédente, nous avons présenté trois architectures génériques basées sur les LFSRs. Le RF, RG et R- LFSR sont régis par des équations mathématiques et reprennent des modèles génériques de registres à décalages fortement utilisés dans la littérature. Ces trois opérateurs agencés dans des configurations spécifiques peuvent réaliser la majorité des opérations de décalages nécessaires dans les normes. Néanmoins, au niveau des opérations requises par les trois standards IEEE 802.16, GPP LTE et IEEE 802.11, il est apparu qu'une seule des opérations parmi l'ensemble de celles étudiées dans le codage convolutif, turbo codage, CRC et génération de séquence aléatoires, ne peut être réalisée par aucune des trois architectures réalisées précédemment. Cet inconvénient, nous a poussé à développer une quatrième architecture basée sur les LFSR : Extended Reconfigurable LFSR (ER-LFSR). Celle-ci nous permettra non seulement d'implémenter les codeurs convolutionnels RSC nécessaires au turbo codage WIMAX mais aussi de comparer l'impact d'une structure plus générale et à fortiori plus complexe sur notre design.

Les trois architectures présentées précédemment sont définies par rapport à des opérations binaires travaillant sur des bits, c'est à dire des blocs de taille 1. Néanmoins comme nous le présentons en 3.2.3, des types de codages convolutifs par blocs sont définis et mis en application dans les standards étudiés. Concrètement, seul le 802.16 (WIMAX) présente dans sa spécification des modes OFDM et OFDMA, l'utilisation d'un Turbo codeur RSC m Binaires Circulaires. Celui-ci est optionnel et est défini pour être exécuté en tant que code complémentaire (Inner code) du code Reed Solomon sur GF(8). Le type de codage circulaire n'influe pas directement sur l'architecture du codeur RSC mais sur son utilisation répétée pour le pré-codage de la séquence initiale. Néanmoins, l'aspect m Binaires quant à lui redéfinit une toute nouvelle architecture qui nous oblige à proposer une quatrième architecture commune. Dans le chapitre précédent 4.3, nous avons réussi à lister la quasi totalité des structures à bases de registres à décalages sous des formes conventionnelles qui pouvaient être exécutées par une architecture générique de type R-LFSR. Dans le cas présent et comme stipulé dans le tableau 3.10, nous devons définir une nouvelle architecture pour un seul type de structures à remplacer.

4.3.4.1 Architecture du ER-LFSR(m,n).

Cette quatrième architecture est définie par deux paramètres m et n. Le premier représente le nombre d'entrées binaires considérées dans l'architecture permettant ainsi les traitements bits ou par blocs. Le deuxième correspond au nombre de branches de sorties de l'architecture et définit le rendement "1/n". Dans un premier temps, nous

présentons notre architecture en ER-LFSR(m,n) et dans un second temps nous exposerons l'architecture en ER-LFSR (m=2 et n=1) adaptée à notre technique des opérateurs communs. L'architecture m-binaire CRSC présentée en 3.2.3, peut être vue comme une entité paramétrable étant donné que les nombres m et n ne sont pas définis. Par rapport à cette définition fondatrice, nous définissons l'architecture en ER-LFSR(m,n). Celle-ci présentée figure 4.23 peut donc être vue soit comme une évolution du R-LFSR soit comme du m-CRSC.

L'architecture du ER-LFSR(m,n) se décompose en trois parties distinctes :

- La génération des N entrées du Registres à décalages à rétroactions linéaires à partir des m entrées.
- La génération des valeurs du registre à rétroactions Linéaires.
- La génération des n sorties à partir des N valeurs du registres.

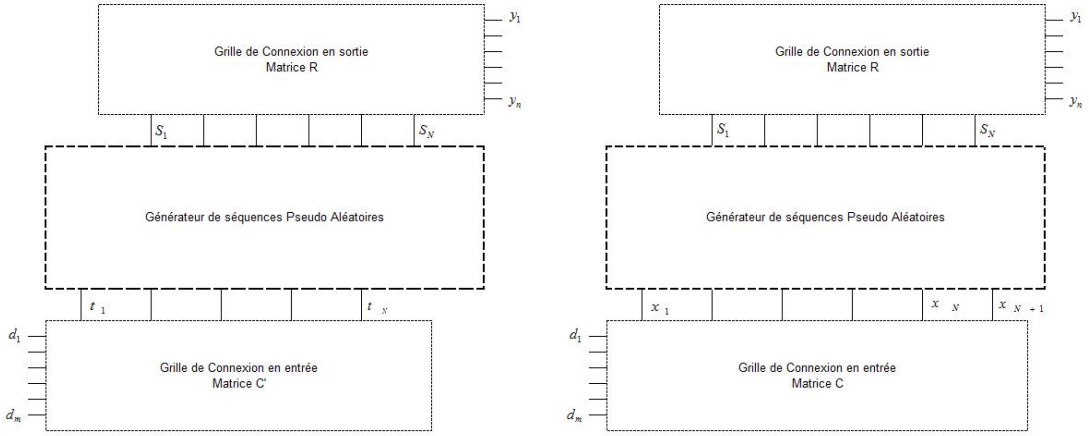

FIGURE 4.22 – Schématique des composantes du ER-LFSR(m,n)

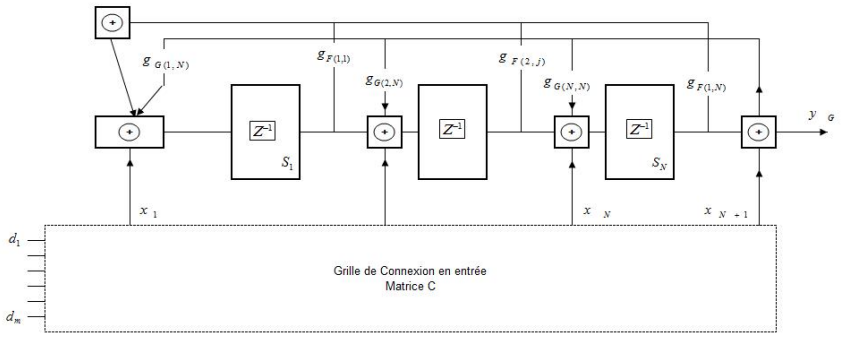

FIGURE 4.23 – Schématique du générateur de séquence du ER-LFSR(m,n)

Nous illustrons ces trois éléments en figure 4.22. Nous pouvons représenter notre système via deux schématiques équivalentes. En figure 4.22, la schématique de gauche correspond à la définition des m-CRSC tel que Berrou les défini en [BC99], la schématique de droite est spécifique à notre architecture ER-LFSR(m,n) mais peut se ramener à celle de Berrou (Annexe E). Notre objectif est de pouvoir définir l'ER-LFSR(m,n) comme un m-CRSC, pour cela nous devons être capable de définir les valeurs S_i des registres sous la forme linéaire :

$$S_i = G_{ER}.S_{i-1} + T_i \tag{4.21}$$

Même si architecturalement, notre architecture se différencie de celle proposée en [BC99], nous pouvons la définir via le même type de récurrence, identique à celle définie pour le R-LFSR. Ici, T_i est un vecteur de dimension 1 x N, caractéristique des combinaisons linéaires de m entrées d_i de l'architecture. G_{ER} est la matrice carré de taille N x N, spécifique de la double boucle de rétroaction de l'architecture proposée en figure 4.23. G_{ER} est définie par :

$$G_{ER} = \begin{pmatrix} g_{ER(1,1)} & \cdots & g_{ER(1,k)} & \cdots & g_{ER(1,N)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ g_{ER(j,1)} & \cdots & g_{ER(j,k)} & \cdots & g_{ER(j,N)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ g_{ER(N,1)} & \cdots & g_{ER(N,k)} & \cdots & g_{ER(N,N)} \end{pmatrix}$$

$$G_{ER} = \begin{pmatrix} g_{F(1,1)} & \cdots & g_{F(1,k)} & \cdots & g_{F(1,N)} + g_{G(1,N)} \\ 1 & 0 & \cdots & \cdots & \cdots \\ 0 & 1 & 0 & \cdots & g_{G(j,N)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & 1 & g_{G(N,N)} \end{pmatrix}$$

Ainsi la matrice G_{ER} représente la rétroaction simultanée des deux LFSR de Galois et Fibonacci combinés. Ce système de rétroaction est le coeur même de notre structure et permet ainsi de reproduire par paramétrage des deux polynômes g_{F1} et g_{GN}, les trois architectures présentées précédemment que sont le RF, RG ou R - LFSR.

En ce qui concerne les entrées T_i de notre nouveau générateur de séquences aléatoires, celles-ci sont plus complexes à définir que dans les trois cas précédents. En effet, nous définissons concrètement (N+1) entrées $x_{j,i}$ qui formeront les (N+1) composantes du vecteur d'entrée X_i du ER-LFSR(m,n) mais qui pourront se ramener aux N composantes du vecteurs d'entrées T_i. Chacune des $x_{j,i}$ est définie comme une combinaison Linéaire des m entrées d_i de notre système tel que :

$$X_i = C \cdot D_i \tag{4.22}$$

$$X_i = \begin{pmatrix} x_{1,i} \\ \dots \\ x_{j,i} \\ x_{N,i} \\ x_{N+1,i} \end{pmatrix} = \begin{pmatrix} c_{(1,1)} & \dots & c_{(1,q)} & \dots & c_{(1,m)} \\ \dots & \dots & \dots & \dots & \dots \\ c_{(j,1)} & \dots & c_{(j,q)} & \dots & c_{(j,m)} \\ \dots & \dots & \dots & \dots & \dots \\ c_{(N+1,1)} & \dots & c_{(N+1,q)} & \dots & c_{(N+1,m)} \end{pmatrix} \cdot \begin{pmatrix} d_{1,i} \\ \dots \\ d_{j,i} \\ d_{m,i} \end{pmatrix}$$

A partir des (N+1) entrées $x_{j,i}$, nous pouvons déterminer les N composantes $t_{j,i}$ de l'entrée T_i de notre système.

$$T_i = X_i + x_{N+1,i} \cdot g_{GN} \tag{4.23}$$

$$T_i = \begin{pmatrix} t_{1,i} \\ \dots \\ t_{j,i} \\ t_{N,i} \end{pmatrix} = \begin{pmatrix} x_{1,i} + g_{G(1,N)} \cdot x_{N+1,i} \\ \dots \\ x_{j,i} + g_{G(j,N)} \cdot x_{N+1,i} \\ x_{N,i} + g_{G(N,N)} \cdot x_{N+1,i} \end{pmatrix}$$

Dans le cas d'un m-CRSC [BC99], le réseau formé par la matrice C est appelée grille de connections. Nous pouvons également utiliser ce terme pour définir une grille de connections de sortie afin de relier les n sorties binaires $y_{k,i}$ au valeurs des registres $s_{j,i}$. Ainsi,

$$y_{k,i} = r_k \cdot S_{i-1} + s_{1,i} + t_{1,i} \tag{4.24}$$

r_j est une matrice de taille 1 x N, correspondant à la jième ligne de la matrice R de taille n x N, caractéristique de la grille de sortie reliant les N registres aux n sorties $y_{k,i}$. **Nous proposons une comparaison détaillée des architectures de Berrou et celle du ER-LFSR(m,n) en Annexe E.4**.

4.3.4.2 Choix de l'Architecture du ER-LFSR(2,1).

L'architecture du ER-LFSR(m,n) permet donc d'effectuer les opérations propres à un m-CRSC et au R-LFSR. Néanmoins, cette architecture est "générique" mais pas encore "commune". Nous devons spécifier m et n afin de proposer une version définie et implémentable. Nous choisissons ainsi m = 2 et n = 1. Pour justifier ce choix, nous étudions les besoins non seulement dans les standards IEEE 802.16, IEEE 802.11 et 3GGP LTE mais aussi dans la littérature. Comme mentionné précédemment, le seul m-CRSC normalisé est celui du DVB-RCS [ETS00] repris dans les 802.16 qui est un duo-binaire CRSC, c'est à dire où m=2. Ainsi à partir de cet état de fait en considérant que ce codeur (Figure 4.25) est le seul m-CRSC dans les normes étudiées, nous nous basons sur son architecture (Figure 4.25) pour choisir n=1 et m=2.

En effet, même si pour m=2, la matrice C définit par Berrou est de la forme :

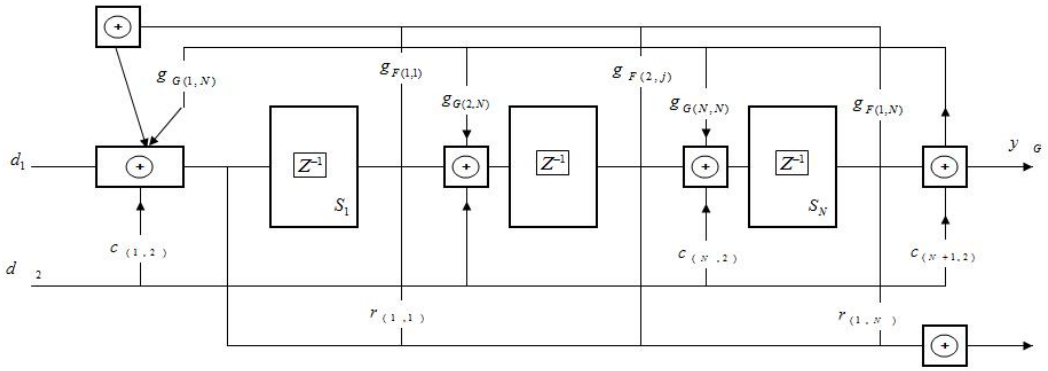

FIGURE 4.24 – Schématique du générateur de séquence du ER-LFSR(2,1)

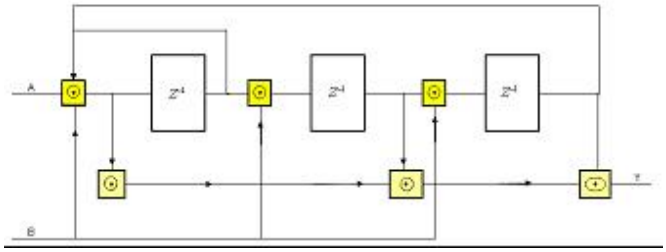

FIGURE 4.25 – Codeur CRSC du 802.16 m=2 et n=1

$$C_{Berrou} = \begin{pmatrix} 1,1 \\ 0,1 \\ 0,1 \end{pmatrix} \; avec \; C = \begin{pmatrix} 1,1 \\ 0,1 \\ 0,1 \\ 0,0 \end{pmatrix}$$

Concrètement, la première entrée d_1 ne devra être injectée que dans le système d'addition du premier registre. Par contre, l'entrée d_2, sera requise pour tous les registres. Ainsi, dans le cas du 802.16, l'entré d_1 peut architecturalement être injectée au niveau de l'additionneur du premier registre, d'où notre choix de m=2 mais qui équivaut à m=1 étant donné que les premiers coefficients $c_{j,1}$ sont figées.

En ce qui concerne la valeur n. Etant donnée qu'une structure à n entrées n'est utilisée que dans le cas des codeurs convolutionnels et non dans le cas des CRC et des générateurs de séquences aléatoires qui présentent des polynômes générateurs de degrès bien plus elevés, nous posons n=1 en connaissance de l'obligation de devoir dupliquer la structure dans le cas de rendement 1/n ou k/n. L'ER-LFSR(2,1) ou simplement ER-LFSR est donc défini par :

- Deux entrées binaires d_1 et d_2 (entrées séries)(m=2).
- Quatre entrées sur N+1 bits pour paramétrer les quatre branches paramétrables coefficientées.
- Une entrée sur N bits pour insérer la séquence initiale du LFSR
- Une sortie série du registre à décalage y_G
- Une sortie série y_1 (n=1).
- Une sortie sur N bits pour récupérer la valeur des registres.

Comme pour le R-LFSR, nous pouvons présenter la cellule de base de l'architecture qui sera repétée N fois, cette cellule est dite en "H" (figure 4.26)

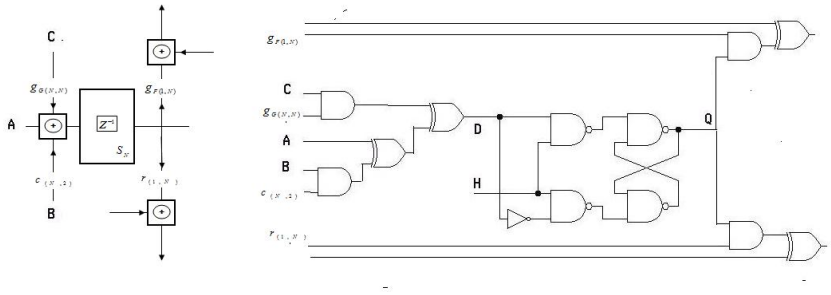

FIGURE 4.26 – Cellue de base du ER-LFSR

4.3.4.3 Utilisation de l'ER-LFSR.

L'ER-LFSR peut être vu comme la concaténation de deux filtres IIR, une en forme directe, l'autre en forme transposée. Ainsi, même si l'ER-LFSR est une version évoluée

du R-LFSR, nous ne sommes pas obligé de nous contraindre à utiliser le paramétrage défini dans le cas du R-LFSR pour les systèmes mettant en oeuvre des LFSR de type de Fibonacci. Nous pouvons directement utiliser les composantes en forme directe.

Grâce à la structure en ER-LFSR, nous pouvons directement implémenter les codeurs convolutionnels et les générateur de séquences aléatoires (LFG ou LFSR) sans besoin de modifier le paramétrage adéquate ni les séquences initiales. Cet état de fait est d'autant plus utile dans le cas du générateur de séquence Gold du 3GPP LTE. Dans le chapitre précédent 4.3.3, nous avons établi que nous pouvions modifier le paramétrage et la séquence initiale pour réaliser, cette architecture via le R-LFSR, mais sous la condition de dupliquer les registres à décalages, c'est à dire d'utiliser quatre LFSR de taille 25 ou 17 au lieu de deux dans chacun des cas. Ici, comme le générateur des séquences de Gold du 3GPP LTE est un système IIR, il sera facilement implémentable avec l'ER-LFSR de part sa structure intrinsèque en IIR et de ces deux sorties y_G et y_1. Nous résumons en figure 4.27, le perimètre de chaque architecture c'est à dire les 35 opérations en LFSR des trois standards considérés et les possibilités de réalisation par les différents LFSR. Nous spécifions pour chaque opération le nombre de chaine (N_{ch}) et la taille de ces chaines (N_r). Si le nombre de chaine est différent pour un des LFSR, nous le précisons dans le tableau en indiquant celui-ci. Les désignations sont utilisées en figure 5.2 du chapitre suivant.

Standard	n°	Désignation	Fonction	Nr	Nch	RF-LFSR	RG-LFSR	R-LFSR	ER-LFSR
WIFI	1	CRC 16	CRC- Codage	16	1		X	X	X
	2	CRC 16	CRC - Décodage	16	1		X	X	X
	3	CRC 32	CRC- Codage	32	1		X	X	X
	4	CRC 32	CRC - Décodage	32	1		X	X	X
	5	Scramb 7	Embrouilleur	7	1	X	X	X	X
	6	Scramb 7	Dé-Embrouilleur	7	1	X	X	X	X
	7	Scramb 7	Embrouilleur	7	1	X	X	X	X
	8	Scramb 7	Dé-Embrouilleur	7	1		X	X	X
	9	NSC 1/2	Codeur Convolutif NSC 1/2	6	1			X/2	X/2
	10	NSC 2/3	Codeur Convolutif NSC 2/3	4	2			X/6	X/6
3GPP	11	CRC 24	CRC- Codage	24	1		X	X	X
	12	CRC 24	CRC - Décodage	24	1		X	X	X
	13	CRC 16	CRC- Codage	16	1		X	X	X
	14	CRC 16	CRC - Décodage	16	1		X	X	X
	15	CRC 12	CRC- Codage	12	1		X	X	X
	16	CRC 12	CRC - Décodage	12	1		X	X	X
	17	CRC 8	CRC- Codage	8	1		X	X	X
	18	CRC 8	CRC - Décodage	8	1		X	X	X
	19	Scramb 25	Embrouilleur	25	2			X/4	X
	20	Scramb 17	Dé-Embrouilleur	17	2			X/4	X
	21	NSC 1/2	Codeur Convolutif NSC 1/2	8	1			X/2	X/2
	22	NSC 1/3	Codeur Convolutif NSC 1/3	8	1			X/3	X/3
	23	Turbo Code (RSC 1/2)	Turbo Code RSC 1/2	4	2			X	X
WIMAX	24	CRC 32	CRC- Codage	32	1		X	X	X
	25	CRC 32	CRC - Décodage	32	1		X	X	X
	26	CRC 16	CRC- Codage	16	1		X	X	X
	27	CRC 16	CRC - Décodage	16	1		X	X	X
	28	Scramb 22	Embrouilleur	22	1	X	X	X	X
	29	Scramb 22	Dé-Embrouilleur	22	1	X	X	X	X
	30	Scramb 15	Embrouilleur	15	1	X	X	X	X
	31	Scramb 15	Dé-Embrouilleur	15	1	X	X	X	X
	32	Scramb 11	Embrouilleur	11	1	X	X	X	X
	33	NSC 1/2	Codeur Convolutif NSC 1/2	6	1			X/2	X/2
	34	Turbo Code (1/2)	Turbo Code 2-CRSC 1/2	3	2				X
	35	Turbo Code (1/3)	Turbo Code 2-CRSC 1/3	3	2				X/4

FIGURE 4.27 – Périmètre Fonctionnel de chaque LFSR

4.4 Conclusion

A partir des explications précédentes, nous pouvons désormais établir qu'une grande partie du terminal peut être factorisée en utilisant trois types d'opérateurs communs ; un type LFSR, un type FFT et un type CORDIC. Le fait de proposer différents niveaux de granularité pour chaque opérateur commun autorise à obtenir un terminal optimisé en combinant différents opérateurs de chaque type. Au contraire, si nous cherchons la généricité de ce terminal, nous pouvons dès lors définir un premier jeu d'opérateurs communs en se basant à chaque fois sur celui de chaque type qui opère le plus d'opérations. Ainsi, ce premier jeu d'opérateurs communs serait constitué de l'ER-LFSR, du papillon FFT/Viterbi et d'un opérateur CORDIC.

Finalement, il est à noter que pour valoriser nos travaux, nous proposons une évolution architecturale en Annexe H, valable pour le R et l'ER-LFSR afin qu'ils puissent constituer un élément de base d'un module CORDIC. Nous pourrons ainsi, évaluer l'impact du niveau de granularité entre OC dans le chapitre suivant.

Chapitre 5

Résultats d'Implémentations

Sommaire

5.1 Introduction

Dans ce chapitre, nous proposons les résultats d'implémentation afin de valider la technique des opérateurs communs. Pour évaluer l'impact d'un terminal complet construit à partir des trois familles d'OC, nous focalisons principalement notre attention sur la famille des LFSR. Ces opérateurs présentent des architectures simples, nous permettant ainsi d'étudier l'application concrète de la technique sur un cas particulier.

Dans un premier temps, nous étudierons les différentes possibilités de l'opérateur commun LFSR que nous proposons. En effet, l'OC peut être implémenté pour être

réutilisé par différentes fonctions en considérant l'agencement des OC et des données dans le temps ou être implémenté sous forme de Banc qui relâche les contraintes d'ordonnancement. Nous définissons, jusqu'à présent les architectures en LFSR sans tenir compte du nombre de registres. Nos implémentations nous permettront de montrer que cette taille varie selon le type d'implémentation que nous choisissons. Comme nous développons, quatre OC en LFSR chacun factorisant un nombre d'opérations différent, nous évaluerons, en termes de complexité matérielle, l'intérêt de factoriser plus ou moins le design. Pour cela, la première partie se divisera en quatre études :

1. Nous étudierons l'opérateur commun $OC_{ER-LFSR}$ défini afin d'être ordonnancé pour réaliser 100% des besoins en LFSR.
2. Nous étudierons l'opérateur commun défini en BOC pour réaliser 100% des besoins en LFSR.
3. Nous diminuerons le périmètre de factorisation en réalisant les deux études précédentes en utilisant les différents LFSR définis en 4.3.
4. Nous proposerons une interconnexion spécifique des LFSR (Annexe G) afin de diminuer le nombre de registres à considérer dans les trois cas précédents et de maximiser le gain en complexité (figure 5.1).

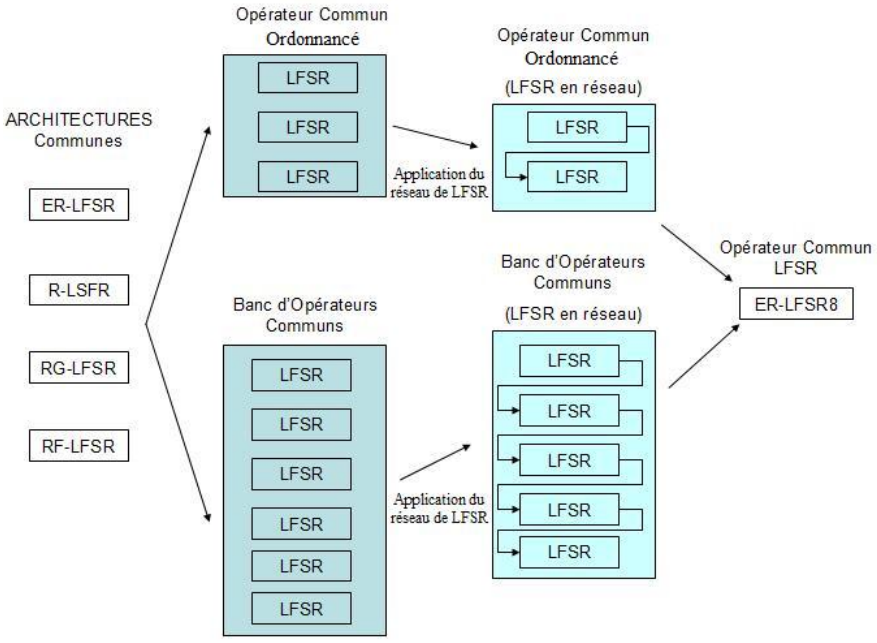

FIGURE 5.1 – Démarche dans l'Etude des LFSR en tant qu'OC

Dans un second temps, nous appliquerons la méthodologie proposée par GUL à nos OC ([GAP+10]) pour étudier l'impact de différentes combinaisons d'OC LFSR. La technique des opérateurs communs propose de déterminer le meilleur ensemble possible

selon un critère donné. Ici, en se référant à la complexité matérielle, nous déterminerons cet ensemble.

Finalement, nous conclurons ce chapitre par une discussion sur le choix de l'OC et l'application du BOC. En annexe H, nous définissons une évolution du R-LFSR de telle sorte que ce dernier puisse réaliser non seulement les opérations d'un R-LFSR tel que défini en chapitre 4 mais aussi devenir l'élément de base d'un module CORDIC et par conséquent d'un module FFT. Ces granularités distinctes dans la construction d'un OC FFT, nous permettront de conclure sur l'utilité et l'impact de construire un OC à partir d'autres OC de plus faible granularité. Le dernier point abordé nuancera les résultats des BOC en LFSR en évaluant les Banc d'Opérateurs Communs appliqués aux OC en FFT que nous spécifions en chapitre 4.

5.2 Impact de l'implémentation des OC LFSR

5.2.1 Opérateurs Cadencés et Bancs d'Opérateurs

5.2.1.1 Eléments de Comparaison

Les systèmes de LFSR que nous développons présentent des architectures binaires simples. Or, ces architectures sont à rétroactions linéaires et nécessitent les N entrées précédentes pour calculer une sortie au temps t. Cette linéarité induit un traitement des données par flots et rend complexe la réutilisation d'une même structure pour des fonctions successives ou en parallèle de la chaîne de traitement. Ainsi, la définition de l'OC LFSR dépendra de la manière dont celui-ci est utilisé. Comme nous travaillons dans une approche pragmatique, nous définissons cette taille vis à vis des standards que nous considérons. Ces standards présentés en Annexe F ont des besoins en LFSR différents que ce soit le type, le nombre ou la taille. Nous résumons ces besoins dans les trois tableaux 3.10, 3.3 et 3.19 en 3.2. Ainsi, nous devons déterminer la taille de la plus grande instance à implémenter et son nombre, ces deux paramètres pouvant ne pas être dépendants du même standard. Le standard 802.11 ne définit pas de voies montantes et descendantes et s'implémente par une chaine d'émission et réception "classique". Le 802.16 quant à lui définit dans ses spécifications une liaison FDD FULL Duplex. En ce qui concerne le 3GPP LTE, qui est toujours à l'état de "draft", l'aspect Duplex est autorisé par l'utilisation de deux modulations différentes pour les voies montantes et descendantes. La considération de ces deux derniers standards dans des communications full duplex nécessite de distinguer des chaines de traitements spécifiques pour l'émission et la réception des données et de pouvoir les utiliser en simultané. Nous résumons ce besoin pour les structures en LFSR en figure 5.2. La figure 5.2 définit pour chacun des standards, les différentes opérations nécessaires et la taille des polynômes générateurs s'y référant. Les désignations reprennent celle de la figure 4.27 du chapitre précédent. Les couleurs permettent d'identifier quelle sera l'architecture en LFSR minimale pour réaliser ces opérations (RF, RG, R ou ER-LFSR). Considérons le WIMAX en mode single carrier FULL DUPLEX. Nous distinguerons une chaine pour la voie montante et une autre pour la voie descendante. Pour l'Ul, la chaîne nécessitera un CRC 32, un

scrambler de degrés 15 pour l'embrouillage des données et un codeur NSC 1/2. Pour le DL, la chaine comprendra le décodage du CRC 32, un scrambler de degrés 15 pour le désembrouillage des données. Maintenant, il reste possible d'utiliser une modulation supplémentaire à étalement de spectre afin d'accroitre la robustesse du transfert de données. Celle-ci peut être réalisée de manière optionnelle par un scrambler de polynôme générateur de degrés 22 pour l'Ul et le Dl. Ainsi, il y aura 7 fonctions à réaliser dont le LFSR de taille maximale à implémenter nécessite 32 registres.

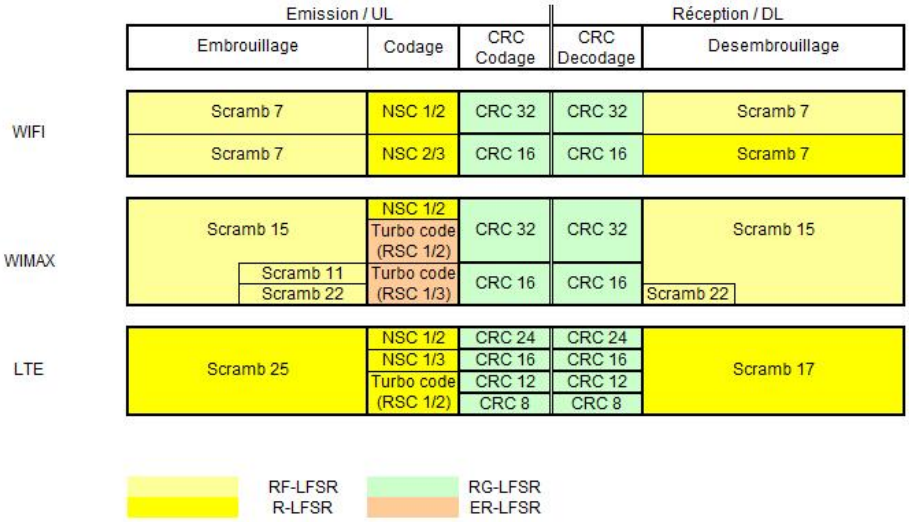

FIGURE 5.2 – Répartition des Besoins en LFSR par Standard

Afin de pouvoir évaluer l'implémentation des architectures en LFSR, nous prenons comme référence de nos comparaisons, une implémentation Velcro des fonctions à implémenter. Pour chaque LFSR, nous définissons un périmètre fonctionnel qu'il peut exécuter (figure 4.27), ainsi pour chaque évaluation de nos résultats, nous prendrons comme référence ce périmètre associé réalisé par une approche Velcro. Afin de présenter une comparaison réaliste des différentes implémentations, les fonctions "identiques" se retrouvant dans les différents standards (CRC16, CRC32 et codeur NSC 1/2) ne sont implémentées qu'une seule fois. Nous résumons les quatre implémentations en Velcro que nous considérons en tableau 5.1. Toutes nos implémentations à l'exception de celle de l'Annexe H, sont effectuées sur un Altera Cyclone II, utilisant les outils de synthèse Quartus. Les résultats en termes de compléxité sont donnés en Logic Cells (LC), ce logic element de base du Cyclone II possède principalement une bascule et une LUT à quatre entrées.

TABLE 5.1 – Etude des différentes Implémentations Velcro

Périmètre Fonctionnel	ER-LFSR	R-LFSR	RG-LFSR	RF-LFSR
Nombre structures Velcro :	28	26	19	8
Nombre registres cumulés :	429	417	297	106
Nombre de LCs :	515	509	361	113

5.2.1.2 Définition de l'OC Ordonnancé

Afin de définir un OC qui pourra être utilisé par chacune des 7 fonctions, l'opérateur commun devra pourvoir réaliser un LFSR de taille maximale 32, tout en permettant un accès aux contraintes de débit de 2/3 du codeur NSC 2/3. Ce codeur à lui seul, nécessite 6 instances différentes et en simultané pour construire l'architecture à partir de l'ER-LFSR. Ainsi, si nous considérons l'ER-LFSR qui peut factoriser 100% des besoins en registres à décalages du terminal tri-standard considéré, nous définissons notre premier opérateur commun $OC_{ER-LFSR}$ composé de six architectures en ER-LFSR de taille 32. Cet opérateur commun nécessite 192 registres. Il est localement plus complexe que n'importe laquelle des structures que nous devons exécuter. Si nous considérons uniquement, les structures en LFSR à remplacer, le nombre de registres maximal à implémenter est de 50 registres dans le cas de l'embrouilleur de l'Uplink du 3GPP LTE et au minimum de 7 dans le cas de celui du IEEE 802.11. Maintenant, si nous réalisons une première implémentation sur une cible Altera/Cyclone II en utilisant les outils Quartus nous obtenons de grandes différences en termes de complexité. L'opérateur implémenté représente 384 logic cells (LC) dont 180 uniquement réservées à des LUTs. Les deux structures précédentes donnent à titre d'exemple une implémentation coûtant 52 et 7 LC dont aucune LC n'est réservée à des LUTs. Ainsi, localement, l'opérateur $OC_{ER-LFSR}$ est plus complexe que les structures à remplacer. En ce qui concerne la rapidité de l'opérateur, les systèmes de LFSR que nous utilisons seront tributaires des différentes opérations "entre" les registres. Dans le cas de l'ER-LFSR, le chemin critique se situe au niveau du système FIR de la boucle de sortie qui réalise l'addition pondérée modulo 2 des valeurs des différents registres. Plus la taille du registre sera importante plus ce chemin critique sera long. Ici, la fréquence maximale de $OC_{ER-LFSR}$ est de 163 MHz. Cette fréquence est à comparer avec les structures à remplacer sur la même cible, la fréquence du système à remplacer le plus lent est de 184 MHz pour l'embrouilleur 3GPP/Ul et de 197Mhz pour le plus rapide dans le cas de l'embrouilleur IEEE 802.16. Ainsi, l'opérateur commun LFSR spécifié $OC_{ER-LFSR}$ présente une forte complexité hardware par rapport à chaque structure que nous visons à remplacer. Si maintenant, nous implémentons ces 28 structures (tableaux 3.10, 3.3 et 3.19), le coût est de 515 LCs. Ainsi, sachant que nous devons traiter, dans le pire des cas, 7 fonctions distinctes pour un mode donné, nous pouvons estimer que le besoin de seulement deux $OC_{ER-LFSR}$ engendrera une complexité supérieure à celle d'une approche Velcro sans compter l'ajout d'un système de contrôle inhérent à la réutilisaiton de la même instance physique.

5.2.1.3 Définition du BOC

Afin de relâcher les contraintes sur le cadencement des OC et de faciliter leurs gestions, nous étudions l'implémentation sous forme de Banc. Par conséquent, nous devons dupliquer le même LFSR (type et taille) en nombre suffisant afin d'allouer les différentes fonctions pour les trois standards précédents exécutés en alternance. Contrairement au cas précédent, le fait d'utiliser un BOC, permet de répartir les différentes instances d'un OC par fonctions. Ainsi, toujours en considérant un LFSR de 32 registres, la chaîne qui requiert le plus grand nombre d'instances est celle d'un mode WIFI en mode ERP, c'est à dire utilisant le codeur NSC 2/3 (6 instances), plus quatre autres instances nécessaires pour le codage et décodage des CRC et des scramblers. Le BOC sera alors constitué de 10 instances.

Pour réaliser 100% des opérations à base de LFSR du terminal tri-standard considéré, nous devons implémenter un ER-LFSR de taille 32 qui sera dupliqué 10 fois dans le banc d'opérateurs. Nous réalisons cette implémentation sur la même cible Altera/Cyclone II en utilisant les outils Quartus. Dans le même temps, nous implémentons l'ensemble des fonctions à remplacer via une méthode Velcro. Nous comparons les résultats en termes de surface en Logic Cells. L'implémentation de 10 structures en LFSR de taille 32, pour un nombre de 320 registres donne un résultat de 623 Logic Cells. A périmètre fonctionnel équivalent et dans les même conditions, toutes les fonctions en LFSR requièrent 515 Logic Cells. C'est à dire que le terminal obtenu en remplaçant 100% des besoins en LFSR est 20% supérieur à une implémentation Velcro. Ce surcoût dû à la duplication d'une même entitée localement complexe était une possibilité envisagée dès la définition de l'architecture en bancs. Ainsi, l'OC résoud le problème de l'ordonnancement et du temps de développement du terminal mais peut entrainer un coût supérieur en complexité.

5.2.1.4 Etude sur le périmètre factorisé

Nous développons dans ce rapport 4 OC LFSR chacun factorisant un nombre de structures différent. L'ER-LFSR a été proposé pour exécuter un type d'opérations supplémentaires par rapport au R-LFSR, c'est-à-dire le turbo codage m-CRSC du WIMAX. Ce turbo codeur se décline en deux versions de débit 1/2 ou 1/3. Nous pouvons dès lors réitérer les comparaisons précédentes en considérant les trois autres OC LFSR et leurs périmètres associés.

Tout d'abord, nous définissons l'OC_{R-LFSR}, construit à partir du R-LFSR. Concrètement, cet opérateur pourra réaliser les mêmes opérations que l'$OC_{ER-LFSR}$ à l'exception des deux turbo codeurs du WIMAX. Comme en 5.2.1.2, l'OC_{R-LFSR} devra pourvoir réaliser un LFSR de taille maximale 32 tout en permettant l'implémentation d'un codeur de rendement de 2/3, ce qui nécessitera six architectures de taille 32 pour un besoin total de 192 registres, équivalent à celui de l'ER-LFSR. Si nous considérons, l'implémentation FPGA sur un Altera/Cyclone II, le coût unitaire d'un tel OC est de 198 LCs. Pour comparaison, l'équivalent en Velcro des structures à remplacer donne un résultat de 509 LCs. Le coût d'un OC_{R-LFSR} est de 38% par rapport à l'équivalent Velcro alors que celui de l'$OC_{ER-LFSR}$ est de 74% par rapport à son propre équivalent

TABLE 5.2 – Etude selon le périmètre factorisé

-	ER-LFSR	R-LFSR	RG-LFSR	RF-LFSR
Taille :	32	32	32	22
Nombre :	10	13	6	4
Registres :	320	416	192	88
Complexité (%) :	120	81	65	96

Velcro. Pour que la comparaison soit équitable, nous devons noter que les deux turbo codeurs du WIMAX devront être systématiquement implémentés en plus de l'OC_{R-LFSR} ramenant ce coût à 211 LCs soit 41% pour le périmètre fonctionnel équivalent à celui de l'ER-LFSR. Ainsi, en diminuant la factorisation du terminal, nous obtenons une plus grande marge de manoeuvre quant à la possible duplication du même OC.

Nous pouvons également réitérer la comparaison des BOC/Velcro, en diminuant le périmètre factorisé et en nous focalisant uniquement sur les périmètres fonctionnels des trois autres opérateurs R-LFSR, RG-LFSR et RF-LFSR. Les résultats sont donnés pour chaque BOC en précisant la taille de l'instance en LFSR dupliquée, le nombre de ces instances, le nombre de registres total pour chaque BOC et le rapport en % entre l'implémentation en BOC et leurs équivalents Velcro. Il est à noter que dans le cas du R-LFSR, la chaine qui requiert le plus grand nombre d'intances est celle du 3GPP LTE. En effet, comme expliqué en 4.3.3, le R-LFSR doit être dupliqué pour réaliser les générateurs de séquences Gold, ce qui ramène le nombre d'intances à 13 (une de plus pour chacune des quatre chaînes).

Conclusion sur l'OC ordonnancé et le Banc d'Opérateurs : ces premières implémentations sont instructives quant à l'application de la technique des OC sur un terminal tri-standard. Premièrement, la définition d'un seul et même OC qui devrait être ordonnancé est unitairement moins complexe que l'équivalent Velcro mais comme nous pouvions nous y attendre, localement supérieur à n'importe laquelle des structures à remplacer. Un besoin de dupliquer cet OC pourrait engendrer un surcoût de la méthode. L'impact de la factorisation est également à souligner dans la mesure où ne pas considérer seulement deux opérations permet de diviser par deux le coût unitaire de l'OC. Deuxièmement, l'application du BOC n'engendre pas automatiquement un gain en complexité. Dans le cas de l'ER-LFSR, ce coût reste supérieur à un équivalent Velcro. Nous montrons ainsi, qu'éviter le cadencement des OC a un coût à payer. Cependant, ce surcoût est à nuancer car la définition du BOC dans les trois autres cas engendre un gain jusqu'à 35% et permet également d'utiliser les OC à leurs vitesses maximales de fonctionnement alors que les deux $OC_{ER-LFSR}$ et OC_{R-LFSR} doivent fonctionner sept fois plus vite que les OC du banc. Dans le cas particulier des LFSRs, nous proposons dans la section suivante une interconnexion spécifique des différentes instances des LFSR afin d'optimiser la complexité engendrée.

5.2.2 Réseau de LFSR Interconnectés

5.2.2.1 Principe de l'interconnexion en LFSR proposée

Nous avons vu précédemment que la technique des opérateurs communs implémentée en Banc d'Opérateurs Communs pouvait s'avérer plus complexe que l'ensemble des opérations que le banc vise à remplacer. Ici, nous sommes dans un cas très particulier qui est celui des registres à décalages à rétroactions linéaires, c'est-à-dire que pour définir un banc d'opérateurs communs nous implémentons le registre à décalage de la taille maximale requise. Dans le cas précédent, ce registre est dû au CRC du WIFI et/ou du WIMAX qui est de taille 32. Nous devons ensuite le dupliquer pour satisfaire le standard qui demande le plus d'instances indépendantes, ici 10 pour le WIFI utilisant un codeur NSC 2/3. Or, l'ER-LFSR de taille 32 dupliqué est surdimensionné en taille pour la majeure partie des opérations à exécuter. Ainsi, nous proposons une structure spécifique dédiée au LFSR, permettant une optimisation de la taille des opérateurs communs à implémenter.

Considérons les standards/modes S_j, chaque standard requiert P fonctions f_i chacune nécessitant N_{ij} registres. Ce nombre N_{ij} est posé égal à $N_{ch,ij}$ ·$N_{r,ij}$, où $N_{ch,ij}$ est égal au nombre de registres à décalages de taille $N_{r,ij}$ requis. Un banc d'opérateurs tel que défini jusqu'à présent va considérer l'instance de taille maximum, c'est à dire $Max_j(N_{r,ij})$ qu'il va dupliquer selon le besoin maximum d'un standard, c'est à dire $Max_j(\sum_i(N_{ch,ij}))$. Ainsi, le nombre de registres concrètement implémenté sera de :

$$Max_j(\sum_i(N_{ch,ij})) \cdot Max_j(N_{r,ij}). \tag{5.1}$$

Or, le nombre de registres réellement requis (NTS) pour le terminal multi-standard sera celui requis par le standard qui en demande le plus et sera suffisant pour exécuter les autres standards. Ce Nombre NTS est égal à $Max_j(\sum_i(N_{ch,ij} \cdot N_{r,ij}))$, c'est à dire $Max_j(\sum_i(N_{ij}))$. La structure que nous proposons va utiliser le principe de linéarité d'un LFSR. Ainsi, nous proposons de construire une structure de LFSR de tailles identiques T_r, interconnectés de manière à ce que le nombre de registres réellement implémentés se rapproche de NTS et que par conséquent, nous puissions développer des opérateurs communs qui soient les plus adaptés à la taille des structures à remplacer.

La figure 5.3 donne un exemple de cette structure. Chaque opérateur de taille T_r est relié via ces entrées et sorties aux opérateurs suivants et précédents. Ces liaisons travaillant au niveau binaire dans notre cas d'étude, elles sont paramétrées par un simple bit. Ainsi, pour minimiser le nombre de registres nous devons définir T_r qui minimise l'équation 5.2 :

$$Max_j(\sum_i(N_{ch,ij}.E(\frac{N_{r,ij}}{T_r}).T_r)) = N_{T_r}.T_r \tag{5.2}$$

Où N_{T_r} représente le nombre de registres à décalage de taille T_r requis. Si maintenant,

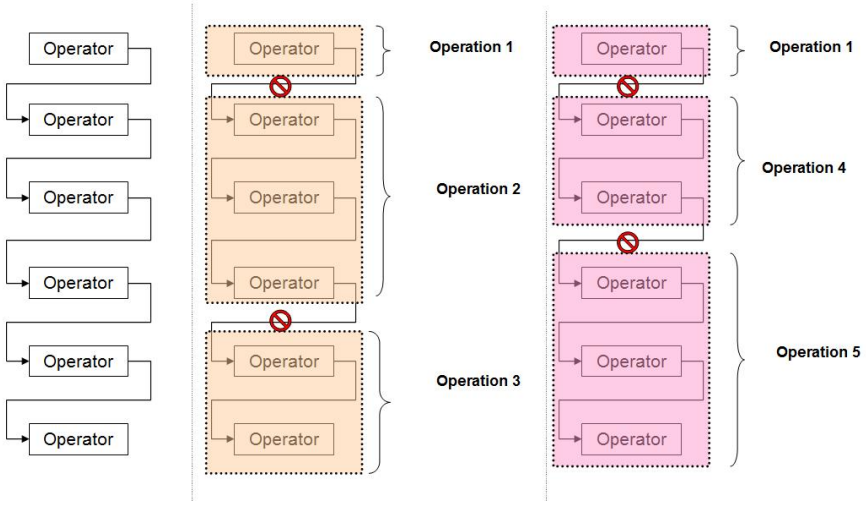

FIGURE 5.3 – Structure en LFSR Proposée

nous associons le coût de surface N_s définit en chapitre II pour un OC LFSR de taille T_r, et que nous considérons le coût $N_{s,liaison}$ de chaque liaison, alors nous devons minimiser :

$$N_{T_r}.[N_s + N_{s,liaison}] \tag{5.3}$$

Même si cette structure est développée en priorité pour diminuer le nombre de registres dans un système de Banc, elle peut être utilisée par n'importe quel OC utilisant des LFSR. Ainsi, nous pouvons appliquer ce réseau d'interconnexions aux deux premiers OC LFSR, $OC_{ER-LFSR}$ et OC_{R-LFSR}. Dans ce cas, nous ne cherchons plus à tenir compte de l'exécution de différentes fonctions sur le Banc, étant donné que ces deux OC ne sont dédiés qu'à l'exécution d'une seule opération à la fois. Pour évaluer T_r, nous devons évaluer chaque fonction indépendamment et trouver T_r qui minimise :

$$Max_{ij}((N_{ch,ij}.E(\frac{N_{r,ij}}{T_r}).T_r)) = N_{T_r}.T_r \tag{5.4}$$

En Annexe G, nous recherchons le meilleur compromis entre la granularité qui permet de s'adapter aux structures à remplacer et le surcoût résultant des interconnexions. Ainsi, nous déterminons le taille optimale de T_r égale à 8 pour 23 instances du BOC en ER-LFSR (noté $BOC_{ER-LFSR_{23}}$) et 8 instances pour l'$OC_{ER-LFSR}$ qui sera noté l'$OC_{ER-LFSR_{reseau}}$. En Annexe G, nous détaillons également, le mapping des différentes fonctions de chaque mode du terminal tristandard sur le BOC ainsi obtenu. Pour la suite de nos travaux, nous prendrons systématiquement T_r égal à 8.

5.2.2.2 Application du réseau de LFSR interconnectés sur le terminal tristandard

TABLE 5.3 – Impact du réseau de LFSR sur les OC ordonnancés

-	$OC_{ER-LFSR}$	$OC_{ER-LFSR_{reseau}}$	OC_{R-LFSR}	$OC_{R-LFSR_{reseau}}$
Taille Opérateur :	32	8	32	8
Nombre (N_{T_R}) :	6	8	6	16
Registres :	192	64	192	128
Complexité (%) :	74	24	38	31

Application aux Opérateurs Communs $OC_{ER-LFSR}$ et OC_{R-LFSR} : Les OC $OC_{ER-LFSR}$ et OC_{R-LFSR} sont construits à partir de la duplication de la même chaine de taille maximale par rapport aux structures à exécuter et en nombre suffisant pour exécuter les rendements adéquates. Dans le cas du 5.2.1, la taille était imputable au CRC 32 et le nombre d'instances au codeur NSC 2/3. En appliquant le réseau d'interconnexions nous pouvons diminuer le nombre de registres pour qu'il colle au plus près du besoin et étudier l'impact sur la complexité engendrée.

En se basant sur l'Annexe G, nous construisons les deux nouveaux OC utilisant le réseau $OC_{ER-LFSR_{reseau}}$ et $OC_{R-LFSR_{reseau}}$ mettant en oeuvre des registres de taille 8. Ici, le nombre de registres diffère selon que nous considérons l'ER-LFSR et le R-LFSR. En effet, pour l'$OC_{ER-LFSR}$, la taille maximale du registre à traiter est de 32 mais pour une seule chaine (N_r=32, N_{ch}=1). Le nombre d'instances maximal, quant à lui, est de 6, mais pour une taille de registre de 4 (N_r=4, N_{ch}=6). Dans l'application du réseau de LFSR, ceux ne sont pas ces deux valeurs extrêmes qui vont dimensionner $OC_{ER-LFSR_{reseau}}$ mais le codeur de séquences de Gold qui demande deux instances de 25 registres (N_r=25, N_{ch}=2) soit 8 ER-LFSR de taille 8 . De la même manière et selon les explications de l'Annexe E, l'$OC_{R-LFSR_{reseau}}$ nécessitera 16 R-LFSR de taille 8.

Ainsi, en implémentant ces deux nouveaux OC sur une cible FPGA Altera Cyclone II (tableau 5.3), en utilisant les outils de simulation Quartus, nous obtenons une diminution significative du nombre de LC requises par OC. Pour l'$OC_{ER-LFSR_{reseau}}$, nous diminuons par 3 le nombre de registres requis et la complexité passant de 384 LC à 124 pour une complexité de 74% à 24% par rapport à l'équivalent Velcro. Ce résultat permet de constater que la diminution du coût en LC permet d'envisager une duplication jusqu'à trois reprises de l'$OC_{ER-LFSR_{reseau}}$ si besoin lors de l'ordonnacement tout en présentant de meilleurs résultats qu'une approche Velcro. En ce qui concerne l'$OC_{R-LFSR_{reseau}}$, le nombre de registres n'est diminué que d'1/3 (de 192 à 128) pour une diminution de 20% en LC (de 198 à 161). En appliquant, le réseau de LFSR interconnectés, nous parvenons à diminuer la compléxité de chacun des deux OC, $OC_{ER-LFSR}$ et OC_{R-LFSR}. Dans le cas présent, l'OC de plus forte granularité obtient au final, les meilleurs résultats en termes de complexité matérielle.

Implémentation selon le niveau de Granularité de l'OC présent dans le BOC : a partir des éléments précédents, nous implémentons sur une cible FPGA Cyclone II via les éléments Quartus, les différentes réseaux pour les quatre LFSR. Pour chaque type de LFSR, nous prenons T_r égal à 8 et en déduisons le nombre N_{T_r}, d'instances

TABLE 5.4 – Impact du réseau de LFSR sur les BOC ER-LFSR et R-LFSR

-	ER-LFSR	23ER	R-LFSR	37R
Taille Opérateur :	32	8	32	8
Nombre (N_{T_r}) :	10	23	13	37
Registres :	320	184	416	296
Complexité (%) :	120	60	81	68

TABLE 5.5 – Impact du réseau de LFSR sur les BOC RG-LFSR et RF-LFSR

-	RG-LFSR	18RG	RF-LFSR	10RF
Taille Opérateur :	32	8	22	8
Nombre (N_{T_r}) :	6	18	4	10
Registres :	192	144	88	80
Complexite (%) :	65	47	96	75

interconnectées. Encore une fois, dans le cas du R-LFSR, ce nombre est élevé car pour exécuter la génération des séquences de Gold, nous devons dupliquer chaque chaîne des quatre LFSR considérés. Nous définissons ainsi un $BOC_{ER-LFSR_{23}}$ (noté 23ER dans le tableau 5.4), constitué de 23 ER-LFSR de taille 8. De la même manière, nous proposons les $BOC_{R-LFSR_{37}}$ (37R), $BOC_{RG-LFSR_{18}}$ (18RG) et $BOC_{RF-LFSR_{10}}$ (10RF). Nous résumons les résultats dans les deux tableaux 5.4 et 5.5, où pour chaque type de LFSR, nous comparons un BOC sans réseau de LFSR interconnectés et l'équivalent avec un réseau de LFSR interconnectés. Dans chaque cas la référence à 100% est prise pour l'équivalent des deux BOC réalisé par une implémentation Velcro. Ainsi, l'application du réseau pour chacun des LFSR (tableau 5.5) permet de diminuer le nombre de Logic Cells requises. Ici, nous diminuons pour chaque type de LFSR la complexité de l'implémentation. Cependant, une question reste en suspend, à savoir est il préférable de factoriser au maximum le design par l'utilisation d'OC plus complexe ?

Pour répondre à cette question, nous étudions la combinaison d'un BOC basé sur le R-LFSR, RG-LFSR ou le RF-LFSR avec les structures qu'ils ne peuvent pas exécuter afin d'atteindre le périmètre fonctionnel de l'ER-LFSR. Concrètement, nous implémentons trois cas différents, dans chacun, nous remplaçons les structures initiales par un BOC de LFSR présenté en section 5.2.2.2 et les structures qui ne peuvent pas être substituées par les LFSR, sont laissées en Velcro. Le tableau (tableau 5.6), résume l'implémentation sur cible FPGA de ces différentes combinaisons de Bancs en LFSR et des structures implantées indépendamment. Pour chaque cas, nous énumerons le nombre d'entités physiques requises et le nombre de registres correspondant. L'ER-LFSR organisé par un BOC en réseau interconnecté, permet de factoriser l'ensemble des opérations tout en obtenant des résultats meilleurs que les autres opérateurs, localement moins complexes mais ne permettant pas de satisfaire à l'intégralité du besoin en registres à décalages.

TABLE 5.6 – Etude de l'impact du réseau de LFSR sur 100% du besoin

		ER-LFSR (23ER)	R-LFSR (37R)	RG-LFSR (18RG)	RF-LFSR (10RF)
BOC	Taille OC :	8	8	8	8
	Nombre OC :	23	37	18	10
	Registres OC :	184	296	144	80
Velcro	Nombre Structures :	0	2	9	20
	Registres Structures :	0	12	132	323
Complexité (%) :		60	69	63	95

Evaluation de l'impact sur la consommation dynamique Outre l'aspect surface dans l'implémentation de nos LFSR, il nous a semblé pertinent d'analyser la consommation engendrée par la mise en place d'une telle méthode. En effet, comme nous le montrons dans ce rapport les LFSRs sont utilisés dans les standards de télécommunications. Ainsi, ils ont fait l'objet de nombreuses études relatives à leurs consommations. Le travail de référence est celui de Lowy qui en [Low96] définit une méthode de parallélisation des registres afin de diminuer la consommation dynamique du LFSR. L'idée développée dans ces travaux est que dans le cas d'une structure en LFSR classique, toutes les bascules sont "cadencées" sur la même horloge pour une seule donnée produite en un seul cycle d'horloge. Lowy propose par son architecture de ne faire varier qu'une seule bascule par cycle d'horloge est de diminuer ainsi la consommation dynamique du système. Cette méthode n'a été développé pour le moment que pour des LFSR de type Fibonacci présentant des architectures peu complexes. **La consommation est divisée par deux au détriment d'une multiplication par 5 de la surface requise**. Dans nos travaux, nous ne cherchons pas à proposer en soi, une architecture à faible consommation mais il est tout de même intéressant de pouvoir comparer la structure spécifique en Banc d'Opérateurs Communs que nous développons avec les structures à remplacer vis à vis de la consommation. Pour ce faire, nous suivrons le même protocole qu'en [Low96]. Pour estimer la consommation dynamique nous utiliserons les même notations que Hamid et Chen en [HC98]. Ainsi, la consommation dynamique est donnée par la formule :

$$P = (\frac{1}{t_d}) \cdot C \cdot V_{dd}^2 \cdot P_a \tag{5.5}$$

Où t_d est la période de l'horloge, C, la capacité de l'élément considéré, V_{dd} la tension d'alimentation et P_a, le pourcentage d'activité égal à 50%. Dans notre cas d'étude, le fait d'utiliser des Opérateurs Communs en Banc via un réseau d'interconnexions nous permet de créer l'exact répliqua des structures que nous avons à remplacer. Une fois le banc d'opérateurs paramétré, chaque sous bloc du BOC constitue l'équivalent architectural des structures requises. C'est à dire que chaque bloc considéré pourra fonctionner à la même fréquence que la structure initiale. Pour effectuer ces comparaisons, nous décomposons les LFSR en trois entités différentes, cellule en E ou H (4.3.4.1), cellule

TABLE 5.7 – Décomposition de l'ER-LFSR

$Element\ ER-LFSR:$	$Bascule$	XOR	AND	INV	OR
Cellule H	1	4	4	0	0
Cellule Feedback	0	2	3	0	0
Cellule Liaison	0	0	6	2	2

TABLE 5.8 – Décomposition du R-LFSR

$Element\ R-LFSR:$	$Bascule$	XOR	AND	INV	OR
Cellule H	1	2	2	0	0
Cellule Feedback	0	1	1	0	0
Cellule Liaison	0	0	3	1	1

de Feedback, assurant la rétroaction et cellule de liaison, connectant deux registres à décalages de taille 8. Nous évaluons ainsi le surcoût de consommation des cellules de liaisons en nous basant sur les tableaux 5.7 et 5.8.

Pour effectuer cette évaluation, nous effectuerons une comparaison entre la chaîne en Velcro qui requiert le plus de registres à savoir celle du 3GPP LTE et les deux implémentations en BOC qui permettent de la réaliser à savoir celle de l'ER-LFSR ($BOC_{ER-LFSR_{23}}$) et celle du R-LFSR ($BOC_{R-LFSR_{37}}$). Pour pouvoir réaliser cette évaluation, nous considérons trois points :

1. Les LFSR permettent d'aboutir à l'exact replica des structures à remplacer. Ainsi, une fois le LFSR paramétré, nous supposons que sa consommation dynamique est la même que celle de la structure qu'il remplace. Cette consommation représentera notre référence de 100%.
2. La consommation dynamique supplémentaire résultera par conséquent des liaisons entre les différents LFSR.
3. Dans certains cas, nous devons dupliquer k.n fois le LFSR pour des débits en k/n ou deux fois dans le cas des codes Gold du 3GPP LTE. Dans ces cas précis, nous devrons ajouter un surcoût de consommation dynamique aux BOC en comptabilisant en plus la consommation des structures dupliquées.

Ainsi, le $BOC_{ER-LFSR_{23}}$ comporte 23 instances de LFSR, 22 liaisons dont seulement 14 seront actives lors de la paramétrisation de la chaine du 3GPP LTE. Or, pour réaliser l'exécution du codeur NSC 1/3, le BOC nécessitera deux chaines du codeur NSC 1/3 supplémentaires. Dans le cas du $BOC_{R-LFSR_{37}}$, 37 instances sont nécessaires pour 36 liaisons dont 24 actives. Au niveau de la consommation dynamique, en plus du codeur NSC 1/3 dupliqué, cette implémentation nécessite la duplication des deux chaînes des

TABLE 5.9 – Capacité des Elements Considérés

–	$Bascule$	XOR	AND	INV	OR
$Capacite:$	0.0027 pf	0.0042 pf	0.00215 pf	0.0027 pf	0.0026 pf

TABLE 5.10 – Estimation de la Consommation Dynamique

BOC:	Référence	Cellules (Liaison)	(Duplication)	Total	%
$23ER$:	28 μW	+ 16 μW	+ 8 μW	+ 22 μW	+ 78
$37R$:	28 μW	+ 13 μW	+ 23 μW	+ 36 μW	+ 128

générateurs de GOLD. Pour le calcul, nous reprenons les même données qu'en [HC98] avec V_{dd} = 1,8 V et $\frac{1}{t_d}$ = 30 Mhz en considérant les capacités du tableau 5.9.

Ainsi, en se basant sur les considérations précédentes, nous pouvons estimer (tableau 5.10) que l'implémentation du $BOC_{ER-LFSR_{23}}$ a un coût en consommation dynamique 1,8 fois supérieur à l'équivalent Velcro et qui reste inférieure à celle du $BOC_{R-LFSR_{37}}$, deux fois plus consommatrice.

Maintenant, nous pouvons comparer cette évaluation de consommation avec celle de l'$OC_{ER-LFSR_{reseau}}$ qui doit être ordonnancé et être utilisé jusqu'à 7 fois pour exécuter 7 fonctions différentes. L'$OC_{ER-LFSR_{reseau}}$ est concrètement un réseau de 8 ER-LFSR de taille 8, interconnectés. Ainsi, nous pouvons considérer que l'$OC_{ER-LFSR_{reseau}}$ aura une fréquence au minimum sept fois supérieure à celle du $BOC_{ER-LFSR_{23}}$ pour une complexité, environ divisée par trois (nombre de chaines). Par rapport à l'équation 5.5, la consommation dynamique de l'$OC_{ER-LFSR_{reseau}}$ sera donc environ deux fois celle du $BOC_{ER-LFSR_{23}}$. En effet, si nous reprenons les calculs précédents, en considérant la consommation dynamique maximale possible par les deux architectures, nous obtenons une évaluation de 293 μW pour le $BOC_{ER-LFSR_{23}}$ et de 708 μW pour l'$OC_{ER-LFSR_{reseau}}$.

Conclusion sur le réseau d'interconnexions : Le réseau d'interconnexions permet de limiter au maximum le nombre de registres non utilisés pour chaque allocation de fonction. Ainsi, le gain en logic elements est notable, notamment au niveau de l'ER-LFSR qui devient l'OC à privilégier.

5.3 Choix du meilleur ensemble d'OC LFSR pour un terminal tristandard donné

5.3.1 Evaluation théorique d'une Optimisation par combinaisons d'OC

Dans le chapitre II de définition de la méthode, nous présentons la technique des opérateurs communs en deux étapes, l'identification des opérateurs communs et le choix du meilleur ensemble d'opérateurs pour un standard donné. Dans notre étude, nous développons une approche dite pragmatique. En se basant sur les différents standards, nous définissons des opérateurs communs pouvant réaliser un plus ou moins grand nombre des opérations requises. Le choix est quant à lui effectué suite à des implémentations effectives sur cible. Parallèlement à nos travaux et dans la continuation des travaux de Rodriguez [MPRG06], GUL développe en [GMP07], une implémentation logicielle de la technique de Rodriguez, permettant à partir d'une décomposition graphique d'obtenir

l'ensemble optimal d'opérateurs communs.

Nous avons par conséquent, intégré ([GAP+10]) nos travaux à ceux de GUL en mettant en pratique l'exécutable développé au laboratoire SCEE. L'approche graphique proposée par Rodriguez ne tient pas compte de l'agencement temporel des opérateurs communs or comme les banc de LFSRs que nous proposons sont une solution à cette même problématique d'organisation temporelle, nous pouvons définir une décomposition graphique basée sur les BOC dont le seul paramètre en prendre en compte sera le coût de surface. Ainsi, dans l'approche de Rodriguez, chaque opérateur à un cout monétaire fixe et les liaisons reliant les opérateurs caractérisent un appel de la même entité implémentée une seule fois. Comme les banc d'opérateurs implémentent physiquement un nombre spécifiques d'OC, nous devons faire apparaître dans le graphe TOUS les BOC possibles, chacun avec un coût spécifique et dont le nombre d'appel sera systématiquement égal à 1. Cette approche devenant rapidement complexe, pour illustrer l'utilisation de l'exécutable de GUL via les LFSR, nous restreignons arbitrairement notre cas d'étude à trois modes des standards considérés :

- Un mode 3GPP LTE (FULL DUPLEX), nécessitant les deux scramblers pour le Ul et le Dl, le codage et décodage des CRC de polynôme générateur de degrés 24 et un codage NSC 1/2.
- Un mode 3GPP LTE (FULL DUPLEX), nécessitant les deux scramblers pour le Ul et le Dl, le codage et décodage des CRC de polynôme générateur de degrés 16 et un codage NSC 1/2.
- Un mode WIFI, dont nous implémentons simultanément une chaine de réception et d'émission, nécessitant un scrambler/déscrambler de polynôme générateur de degrès 7, le codage et décodage des CRC de polynôme générateur de degrés 16 et un codage NSC 1/2.

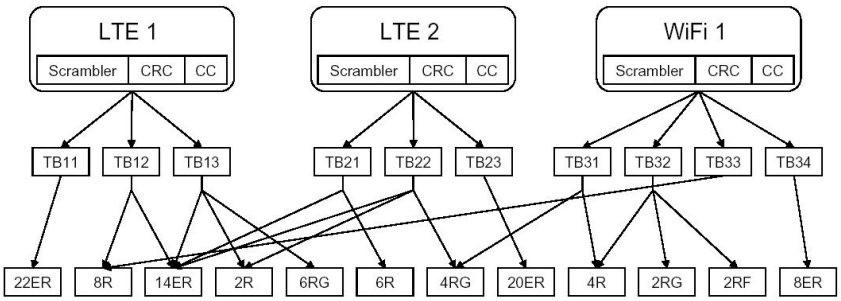

FIGURE 5.4 – Intégration des LFSR dans la décomposition Graphique

Les différentes possibilités d'allocations sont représentées en figure 5.4. En figure 5.4, comme nous considérons une application des bancs d'opérateurs, les fonctions de Codage (CC), CRC et embrouillage (scrambler) doivent pouvoir être réalisées en même temps.

(Nous relions ainsi par le biais "d'opérateurs intermédiaires" (TB) les standards à l'ensemble des Bancs). Nous conservons le même système de notation pour les différents Bancs (22ER désigne un Banc d'Opérateurs Communs composé de 22 ER-LFSR de 8 registres). Nous appliquons le processus développé par Gul et obtenons que le meilleur ensemble est l'association des BOC : 14ER, 8R, 2R et 4 RG. **L'application de cette méthode laisse supposer qu'une association de différents Banc d'Opérateurs Communs pourrait s'avérer plus pertinente que l'ER-LFSR seul. L'étude complète est donnée en [GAP+10]**.

5.3.2 Implémentation pratique d'une combinaison d'OC

Pour étudier l'impact d'une combinaison d'OC, nous pouvons proposer une implémentation de chaque opérateur pour les opérations pour lequel il est le plus approprié en termes de complexité. Par exemple, le R-LFSR est utile pour le Scrambling, le CRC et le Codage Convolutif mais il est surdimensionné pour le CRC car un simple RG-LFSR suffirait. Nous pouvons dès lors étudier, la complexité d'une combinaison de plusieurs bancs d'opérateurs communs, chacun utilisé pour les opérations qui lui sont propres. C'est-à-dire que chaque opérateur ne pourra être utilisé uniquement pour son périmètre spécifique. Celui-ci correspondra aux turbo codeurs du WIMAX et aux générateurs de séquences de Gold pour l'ER-LFSR, aux fonctions de codages pour le R-LFSR plus la fonction de desembrouillage synchrone, au calcul des CRC pour le RG-LFSR et aux fonctions d'embrouillages pour le RF-LFSR. Dans ce cas d'étude on segmente les possibilités d'allocation et chaque banc d'opérateurs devra être dimensionné pour le standard le plus contraignant :

- $BOC_{ER-LFSR_{14}}$: L'ER-LFSR devra être dimensionné pour les deux générateurs de séquences de Gold soit 14 LFSR pour 112 registres.
- BOC_{R-LFSR_3} : Le R-LFSR pour le codeur NSC 1/3 du 3GPP LTE soit 3 LFSRs pour 24 registres.
- $BOC_{RG-LFSR_8}$: Le RG-LFSR, pour les deux CRC 32 (Up et Dl) du WIMAX soit 8 LFSRs pour 64 registres.
- $BOC_{RF-LFSR_{10}}$: Le RF-LFSR pour les deux embrouilleurs et desembrouilleurs (Up et Dl) du WIMAX de polynôme générateur de degrés 15 et 22 soit 10 LFSR pour 80 registres

Si nous diminuons le nombre de LFSR évolués (ER et R-LFSR), nous augmentons le nombre de registres nécessaires de 184 pour le $BOC_{ER-LFSR_{23}}$ à 280 dans cette combinaison des quatre BOC ($BOC_{ER-LFSR_{14}} + BOC_{R-LFSR_3} + BOC_{RG-LFSR_8} + BOC_{RF-LFSR_{10}}$). Une implémentation sur la même cible Altera Cyclone II donne une complexité en Logic Cells de 416 soit 81% par rapport à l'équivalent Velcro. Ce résultat est à comparer avec les 60% de complexité engendrée par le $BOC_{ER-LFSR_{23}}$ seul. En effet, au delà du nombre total de registres, les différentes fonctions considérées ne présentent pas le même besoin et ne correspondent pas systématiquement à un seul type de LFSR. Ainsi, si nous considérons un mode WIMAX utilisant une modulation à étalement de spectre optionnelle, le RF-LFSR devrait être dimensionné pour gérer

deux scrambleurs/déscrambleurs de tailles 15 et 22 alors que le WIFI ne nécessite qu'un scrambleur de degrès 7. La différence serait inutilisée lors de l'exécution du WIFI. L'ER-LFSR, quant à lui, permet de répartir le besoin sur la même structure interconnectée.

Conclusion sur les combinaisons d'OC : On peut en déduire que l'ER-LFSR étant plus complexe que le RF, RG et R-LFSR, il n'est pas localement le mieux adapté à remplacer un grand nombre de structures. Néanmoins, sa capacité à factoriser la totalité du besoin en LFSR permet de profiter des allocations hétérogènes induites par les standards.

5.4 Discussion sur la complexité matérielle de la technique des Opérateurs Communs

Dans ce chapitre, nous avons pu implémenter différents cas relatifs aux opérateurs en LFSR. Nous rappelons que le choix d'implémenter les LFSRs sur une cible FPGA a été un choix arbitraire et qui ne présume pas qu'un tel OC "doit" être implémenté sur une telle cible. Le traitement au niveau binaire peu complexe des architectures LFSR aurait laissé supposer une facilité d'implémentation. Cependant, la diversité en taille et en nombre des chaînes des structures à exécuter impose une analyse détaillée de l'implémentation en Banc d'Opérateurs Communs, nous obligeant à proposer une structure spécifique d'interconnexions afin d'ajuster au mieux la taille des registres. Ce premier point met en lumière l'aspect non trivial dans l'implémentation d'un OC. La définition initiale d'une architecture commune ne suffit pas pour avérer de la pertinence de la méthode mais requière une réelle implémentation sur cible pour en appréhender les contraintes.

Granularité de l'OC Dans notre cas d'étude, en se focalisant sur les opérateurs à registres à décalages, nous avons pu valider la pertinence de l'ER-LFSR en Banc comme l'opérateur à implémenter pour un terminal tristandard. Ainsi, si nous nous focalisons sur la surface engendrée par les différentes implémentations. Dans chacun des cas l'implémentation d'un ER-LFSR en BOC présente des résultats équivalents ou meilleurs que les implémentations concurrentes. Même si l'implémentation est spécifique sous la forme de bancs interconnectés, nous validons bien l'approche qu'un opérateur localement plus complexe permet d'apporter une optimisation globale sur le terminal multistandard. Dans notre cas d'étude, le R-LFSR représente une complexité en LC inférieure à l'ER-LFSR pour un même nombre de registres, il apparait que ce surcoût local de l'OC permet néanmoins de gagner lors de l'implémentation de l'ensemble des opérations à exécuter d'une chaîne de traitement.

Maintenant si nous sortons du cadre du BOC, et que nous considérons les OC, $OC_{ER-LFSR}$ et OC_{R-LFSR} alors nous obtenons le résultat opposé. Comme pour le BOC, deux nouveaux OC ($OC_{ER-LFSR_{reseau}}$ et $OC_{R-LFSR_{reseau}}$) doivent être définis, utilisant un réseau de LFSR interconnectés afin de diminuer la complexité. Le deuxième

TABLE 5.11 – Comparaison de la complexité du $BOC_{ER-LFSR_{23}}$ selon le nombre de standards

	$\text{BOC}_{ER-LFSR_{23}}$	TRI	$LTE/WIMAX$	$LTE/WIFI$	LTE
Logic Cells :	312	515	450	300	235
Complexité (%) :	100	165	144	96	75

opérateur construit à partir du R-LFSR est plus complexe que le premier obtenu à partir de l'ER-LFSR, tout en se rappelant qu'il ne permet pas de réaliser les deux opérations de turbo codage du WIMAX. Ainsi, comme dans le cas du BOC, l'OC de granularité supérieure permet de factoriser 100% du besoin tout en étant le plus optimal en termes de surface.

Finalement dans les deux cas, la structure de base de nos architectures est un ER-LFSR de taille 8, utilisé dans un réseau de chaines interconnectées que ce soit pour un OC ordonnancé ou un BOC. Ainsi, l'ER-LFSR de taille 8 est l'OC que nous proposons pour la classe 2 des registres à décalages. Nous discutons dans le reste de cette conclusion de l'intérêt général du BOC et du niveau de granularité à prendre en compte entre les différentes classes.

Banc d'OCs Ici, outre l'ER-LFSR, l'implémentation vise également à tester le système de banc d'opérateurs communs que nous proposons. Ces résultats démontrent que nous pouvons proposer une architecture qui évite les conflits de ressources entre fonctions tout en limitant la complexité matérielle. De plus, si nous comparons le $BOC_{ER-LFSR_{23}}$ et l'$OC_{ER-LFSR_{reseau}}$, nous remarquons que la complexité de ce dernier est environ la moitié de celle du BOC. C'est-à-dire que sachant que nous devons exécuter au maximum sept fonctions distinctes utilisant des registres à décalages, la simple nécessité de dupliquer l'$OC_{ER-LFSR_{reseau}}$ résulterait en un design équivalent voire plus complexe que celui du $BOC_{ER-LFSR_{23}}$. Le système en BOC est donc à privilégier pour le cas des LFSR.

Les résultats que nous obtenons démontrent un gain en surface sur une cible FPGA ALTERA/Cyclone II. Ce résultat provient de la diminution du nombre de structures de 28 à 23 opérateurs communs. En effet, dans notre cas de figure, il est à noter que les opérations que nous visons à remplacer pour chaque standard se distinguent par leurs architectures, ce qui nécessiterait de toutes les dupliquer dans une implémentation classique en Velcro. La technique des OC permet d'exploiter cette accumulation de structures et de diminuer la complexité engendrée. Si nous considérions des standards supplémentaires, sous la condition que 23 ER-LFSR suffisent à l'implémentation, le gain serait encore plus grand. Or au contraire, si nous diminuons le nombre de standards considérés, la tendance s'inverse et l'utilisation d'un BOC n'engendre plus systématiquement un gain en surface.

Pour illustrer ce point, nous considérons toujours, l'architecture en $BOC_{ER-LFSR_{23}}$. Ce banc d'opérateurs est valable pour un terminal tristandard, mais son dimensionnement est lié au 3GPP LTE. Si maintenant, nous diminuons le nombre de standards considéré

pour se focaliser sur deux terminaux bi-standard (LTE + WIMAX, LTE + WIFI) et un terminal mono standard, uniquement LTE, alors nous comparons le coût du même banc d'ER-LFSR avec les opérations à exécuter pour ces différents cas de figure (tableau 5.11) . Il apparait que dans le cas d'un terminal monostandard, le système de bancs est plus complexe que l'équivalent en Velcro et que dans une approche bi-standard, une fois sur deux, le BOC apporte un gain en complexité. Ainsi, notre approche en Banc d'Opérateurs Communs est bien tributaire du nombre de standards à considérer et dans ce cas précis de l'accumulation des opérations à exécuter.

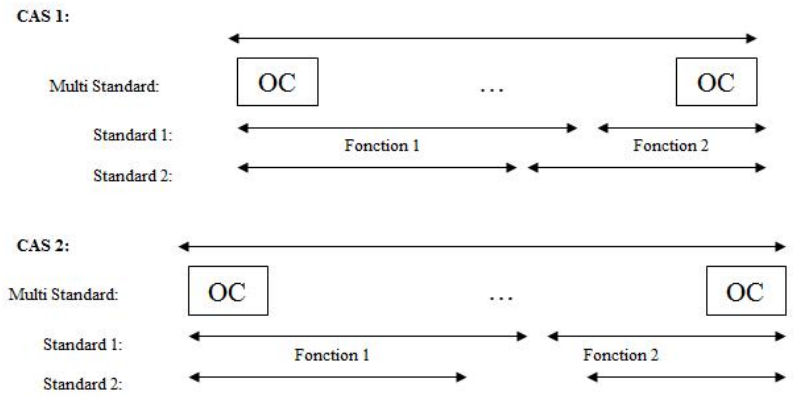

FIGURE 5.5 – Allocation des Fonctions sur un BOC

Le nombre de structures à remplacer affecte l'impact en termes de surface d'un BOC mais un deuxième point qui conditionne la coût d'un banc est la répartition de ces structures sur les fonctions à exécuter. En effet, l'utilité d'un banc d'opérateurs est de permettre une allocation optimale des ressources entre les différentes fonctions d'un BOC. La figure 5.5, illustre notre propos. Dans le cas 1, la fonction 1 nécessite le maximum d'OC pour le standard 1 et la fonction 2 pour le standard 2, ainsi la taille du banc pourra être minimisée, dans le cas 2, les deux fonctions ont leurs maxima d'OC pour le même standard, alors le Banc d'OC sera tributaire du besoin de ce standard et l'optimisation en nombre d'opérateurs sera réduite. Dans le cas des LFSRs, nous retrouvons cette disparité par fonction, notamment entre le WIMAX et le LTE. Les fonctions de CRC nécessitent respectivement une taille maximale de 32 et 24 et les fonctions d'embrouillage de 15 et 25. Ainsi, comme les maxima des besoins sont répartis entre les standards nous pouvons diminuer le nombre d'OC du Banc. La figure 5.5 illustre cette problématique.

Dans le cas du $BOC_{ER-LFSR_{23}}$, l'optimisation venait de deux propriétés : l'accumulation de structures différentes à remplacer et la répartition hétérogène de ces structures dans les fonctions des standards. Les structures en LFSR diffèrent d'un standard à un

autre dans leurs tailles et dans leurs répartitions et permettent, ainsi, d'avancer le BOC comme une solution pour diminuer la complexité, il n'en est pas de même pour tous les autres OC. Pour chaque standard les papillons de FFT ou de Viterbi respectifs sont identiques et le 3GPP LTE définit les deux maxima pour chaque opération (tableau 5.4). Nous sommes dans le cas 2 de la figure 5.5, dans lequel la quantité d'opérateurs communs et leur répartition par fonctions, dépendent d'un seul et même standard. Le BOC sera défini pour les contraintes du LTE à savoir un besoin de 1024 papillons pour la FFT et 128 pour le Viterbi. Si l'OC que nous proposons est unitairement plus complexe que les papillons à remplacer alors le BOC résultant le sera aussi. Or si nous reprenons les deux OC, AS^3 et FFT/Viterbi, bien que possédant le même nombre de module A/S que leurs équivalents respectifs FFT et Viterbi, le système de multiplexeurs gérant les données rendra la structure plus complexe et par conséquent le BOC.

TABLE 5.12 – Nombre de Papillons FFT et Viterbi

	$N_{FFT}/Nombre\ Papillons$	$N_{Viterbi}/Nombre\ Papillons$
LTE :	2048/1024	8/128
$WIFI$:	64/32	6/32
$WIMAX$:	2048/1024	6/32

Construction de l'OC Maintenant, la question se pose de savoir jusqu'où il est souhaitable de chercher à "construire" un OC de forte granularité systématiquement à partir d'OC de granularité plus faible au lieu de l'implémenter directement. Dans une approche théorique telle que développée par Rodriguez [MPRG06], l'opérateur étant implémenté qu'une fois, plus la factorisation est poussée, plus la réutilisation de l'OC permettra un gain en complexité. Or dans une approche pratique, les OC sont dupliqués et doivent obéir à des contraintes de temps et de dépendance des données. Dans le cadre du projet ANR INFOP, pour aller plus loin dans la factorisation des opérateurs LFSR proposés en chapitre IV, nous proposons de définir un nouveau LFSR qui deviendrait l'élément de base d'un module CORDIC et donc d'un papillon de FFT. En Annexe H, nous définissons les modifications à apporter aux OC, R-LFSR et ER-LFSR afin de les utiliser dans la définition d'un module CORDIC. Ici, les structures sont modifiées pour travailler sur des mots de 16 bits et réaliser simultanément une division par une puissance de 2 et une addition (Annexe H). Nous développons ces modifications sur l'OC le plus simple à savoir le R-LFSR et définissons un R-LFSR CORDIC qui constituera l'élément de base du module CORDIC. Nous implémentons un module CORDIC traditionnel composé de 12 étages en pipeline et un deuxième module CORDIC composé cette fois-ci de 24 R-LFSR CORDIC répartis en 12 étages. Cette implémentation donne un surcoût en terme de slice 3 fois plus important que la version initiale du CORDIC passant de 500 à près de 2000 slices. Cette implémentation est faite sur une cible FPGA Xilinx Virtex 4 (spécificité du projet INFOP) mais permet de remarquer que pour un seul module CORDIC implémenté le surcoût est supérieur à l'ensemble du gain obtenu par l'ensemble des opérations de registres à décalages présentées en début de

ce chapitre. Or, ce coût est à revoir à la baisse dans la mesure, où les implémentations ont été réalisées avec un R-LFSR et non un R-LFSR CORDIC, plus complexe. Il est trivial d'envisager que l'implémentation en BOC du R-LFSR CORDIC demanderait une surface plus importante que le R-LFSR. Dans ce cas, nous proposons un OC qui se situe entre deux niveaux de granularité. Il est l'élément de granularité supérieur pour les systèmes en LFSR et devient l'élément de base pour les systèmes à base de FFT. Or, dans le premier cas, il présente une complexité bien supérieure aux opérations à registres à décalages qu'il doit réaliser en tant qu'OC LFSR de type SISO travaillant sur des flots binaires et dans le second cas, il constitue également un élément de base, de complexité excessive par rapport à un module d'A/S de base. Dans les deux cas, les designs sont complexifiés. Ainsi, il apparait que chercher à factoriser au maximum les opérations peut s'avérer pénalisant en termes de coût d'implémentation. En effet, en continuant notre raisonnement, nous pourrions utiliser le R-LFSR CORDIC pour n'importe quel module A/S que ce soit pour l'opérateur CORDIC, le module AS^3 ou bien encore les opérations de multiplications du FFT/Viterbi. Or, l'opérateur CORDIC appartient à la classe 1 des opérations d'additions et multiplications sur des mots, le LFSR à la classe 2 des opérations à décalages sur des bits. Ainsi, dans notre étude, nous distinguerons d'un coté un opérateur commun traitant les systèmes à registres à décalages sur des données binaires et de l'autre un opérateur commun CORDIC, construit "directement". Nous validons ainsi, la classification initialement proposée en 3.2.

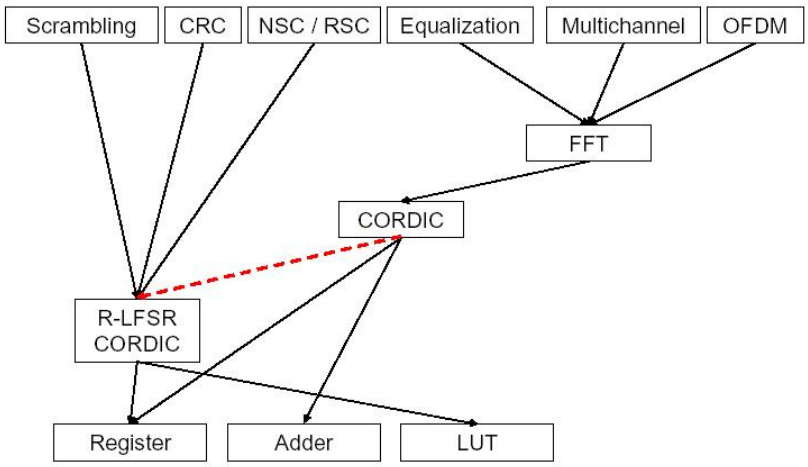

FIGURE 5.6 – Décomposition de la Granularité de l'opérateur FFT

Outre cet aspect, il est à noter que le banc d'opérateurs, a pour principale utilité d'apporter une solution à l'ordannancement global et laissant la possibilité d'un ordonnancement local. Dans les cas que nous abordons, nous implémentons les LFSRs, soit en banc interconnectés, soit en bancs organisés en pipeline. Ici, nous obtenons l'exact réplica des

structures à remplacer. Pour autant, les OC développés présentent des fréquences de fonctionnement déjà inférieures aux fréquences de fonctionnements des structures utilisées par les fonctions à exécuter. Pour diminuer la taille du design utilisant l'OC, nous pourrions diminuer le nombre d'instances des OCs et réutiliser ces derniers au coeur même d'une fonction ou entre différentes fonctions. Or il est à noter que les structures que nous remplaçons, mettent en oeuvre dans leurs conceptions des architectures où les opérateurs sont déjà dupliqués, cela pour atteindre des fréquences de fonctionnement élevées. Ainsi, le fait de construire un opérateur commun de haut niveau à partir d'une succession d'opérateurs communs de bas niveau va diminuer cette fréquence. En effet, si nous reprenons le cas de la FFT, en utilisant le R-LFSR CORDIC comme élément constitutif du module CORDIC, nous perdons en fréquence de fonctionnement au niveau du CORDIC et de la FFT. Si maintenant, nous voulons diminuer le nombre d'instances du R-LFSR CORDIC de 24 en pipeline à 1, itéré 24 fois, les données équivalentes à celles obtenues avec une architecture pipeline ne seraient obtenues qu'après 24 itérations. Et si nous opérons le même principe à chaque niveau de granularité, l'opérateur commun de granularité supérieure verra sa fréquence minimale de fonctionnement augmentée pour chaque niveau. Ce point est contradictoire avec la volonté d'utiliser un OC (ici la FFT) pour le maximum de fonctions possibles, car celui-ci devra pouvoir atteindre des fréquences qui lui permettent une utilisation pour différentes exécutions. Ainsi, indépendamment des classes que nous définissons en 3.2, nous concluons cette étude en spécifiant que la définition d'un OC, visant à être ordonnancé devra se faire en considérant, l'ensemble des OC de granularité supérieure qui l'utilise, étant donné qu'à chaque niveau supplémentaire, l'OC de plus haut niveau mettra plus de temps à exécuter la même opération. Ce constat aboutit à définir "directement", un OC de granularité supérieure, utilisé directement par les fonctions, comme un module FFT ou un ER-LFSR.

5.5 Conclusion

Ainsi, nous pouvons conclure que l'OC ER-LFSR implémenté en banc est à privilégier par rapport à son implémentation en OC ordonnancé. En termes de complexité matérielle et de consommation dynamique les résultats sont meilleurs. Outre cet aspect, le système de banc facilite l'implémentation dès la conception, et permet d'envisager une réutilisation plus efficace quant à un nouveau standard de part la duplication du même élément régulier.

Outre ce premier aspect, le banc permet d'apporter flexibilité et gain en complexité dans le cas des LFSR alors que traditionnellement, la flexibilité est concomitante à une hausse de la complexité. Ce résultat est à nuancer car dans le cas des OC FFT/Viterbi, nous ne pouvons pas attendre un gain en implémentant un système de BOC.

Finalement, le dernier point de cette conclusion est relatif à la granularité de l'OC. En termes de fréquences, construire un OC à partir d'une succession de niveaux d'OC n'est pas pertinent quant à la fréquence d'utilisation de l'OC de plus haute granularité. De plus en termes de complexité, chercher à factoriser au maximum et à un obtenir l'OC

le plus générique conduit à un réel surcout comme dans le cas du R-LFSR CORDIC. Ainsi, un compromis doit être trouvé entre généricité et complexité.

Conclusion

Tout d'abord, rappelons l'objectif de départ de notre étude. Pour faire face à la multiplication des normes de télécommunications, nous recherchons une architecture reconfigurable permettant d'exécuter tour à tour différents standards qui ne sont pas (obligatoirement) prédéfinis à l'avance. La méthode classique utilisée pour implémenter de tels terminaux est la méthode Velcro qui peut être vue comme la coexistence de l'ensemble des chaines de télécommunication nécessaire et d'un simple commutateur pour sélectionner celle voulue. Cette méthode, utile pour les applications temps réel reste dépendante des standards donnés et ne permet pas l'évolution de l'architecture définie vers un standard non prévu à l'avance. Ainsi, nos objectifs sont de :

- proposer une architecture reconfigurable à faible temps de reconfiguration.
- proposer une architecture aussi indépendante que possible des standards utilisés pour sa définition.

Conclusions relatives à la méthode

Le premier résultat de nos recherches concerne la méthode que nous définissons, intitulée " La technique des Opérateurs Communs (OC) ". Placée dans le cadre des techniques de Paramétrisation, nous proposons ainsi de définir des éléments communs, utilisés non seulement entre les différents standards mais aussi à plusieurs reprises dans l'enchainement des fonctions de la chaîne de télécommunication d'un même standard. Un OC est défini à un niveau de granularité inférieur à celui d'une fonction de télécommunication et selon un jeu de paramètres spécifiques, il peut s'adapter pour l'exécution de la totalité ou d'une partie de ladite fonction. Les OC ne sont " reconfigurés " qu'en modifiant un jeu de paramètres, sans modifier pour autant l'architecture. Ce concept présente deux avantages notables :

1. Il définit les OC, indépendamment de la cible visée et autorise l'implémentation conjointe sur des technologies différentes.
2. Il définit des OC qui sont eux-mêmes indépendants des fonctions les exécutant (à l'instar d'une bascule (flip flop) ou d'un additionneur multiplieur (MAC)). Les OC apparaissent comme des éléments génériques et confère au terminal un caractère évolutif.

La définition de tels Opérateurs Communs génériques et paramétrables répond à la problématique de notre étude ; la reconfiguration est bien obtenue en temps réel par modification de paramètres sur une architecture

reconfigurable générique.

Le deuxième résultat de nos recherches est la validation de la méthode en déterminant et implémentant les premiers OC. Pour ce faire, nous étudions les structures requises dans les différentes fonctions de la chaîne de traitement et les répartissons en quatre classes selon les types de données qu'elles traitent et les structures requises pour ces traitements :

1. Les fonctions utilisant les opérations de multiplications et d'additions à l'instar de la FFT, des FIRs, de la corrélation...
2. Les fonctions réalisant des opérations de décalages et des opérations logiques telles que le codage convolutionel, la randomisation, le turbo codage...
3. Les fonctions réalisant des opérations sur la localisation des bits comme " l'interleaving " ou le " puncturing ".
4. Les fonctions réalisant des associations de données comme les différentes modulations.

Parmi ces quatre classes, nous proposons trois familles d'OC :
- La première utilisant l'opérateur CORDIC.
- La deuxième travaillant sur les systèmes de treillis/papillons.
- La troisième relative aux registres à décalages.

Nous spécifions pour chaque famille différents niveaux de granularité pour les OC. Le périmètre des fonctions pouvant être réalisées par chaque OC, n'est pas disjoint et permet ainsi un nombre élevé de combinaisons possibles pour la réalisation du terminal multistandard. **Ces trois premières familles d'OC permettent de factoriser la majorité des fonctions de télécommunication et de valider ainsi le concept de la technique des Opérateurs Communs proposée.**

Une fois la méthode définie et un premier jeu d'OC disponible, nous étudions sa mise en application. A cette fin, nous centrons nos travaux sur une cible hardware en définissant l'Opérateur Commun matériel et la problématique de l'utilisation dans le temps d'un nombre d'instances virtuelles supérieures au nombre (limité) d'instances physiques réellement implémentées. En effet, comme l'OC pourra être utilisé à plusieurs reprises par des fonctions distinctes de la chaînes de traitements, il doit répondre aux contraintes de temps et aux dépendances de données inhérentes à chaque fonction de télécommunication. Nous identifions ainsi, quatre points cruciaux dans l'ordonnancement des OC :

1. La définition (difficulté et temps) d'une telle gestion lors de la conception du terminal.
2. La complexité de sa mise en oeuvre lors de l'exécution d'un standard.
3. Le coût potentiellement élevé de l'infrastructure de contrôle des OC.

4. L'inadéquation possible entre le nombre d'instances physiques requis pour l'exécution d'un terminal multistandard et celles nécessaires à l'exécution d'une nouvelle norme.

Afin de répondre à cette problématique, nous concluons sur l'utilité d'un système de Banc d'Opérateurs Communs (BOC). Celui-ci consiste à attribuer à chaque fonction d'un standard, le nombre d'OC dont elle aura besoin pour son exécution et permet d'éviter les conflits de ressources entre deux fonctions appelant la même instance physique. **Cette architecture en banc permet en priorité de résoudre le problème d'ordonnancement des opérateurs, attribués à des fonctions qui pourraient s'exécuter successivement ou en parallèle et dont le fonctionnement pourrait de surcroit être altéré par une dépendance des données entre fonctions.**

Conclusions relatives aux implémentations

Pour évaluer l'impact de la technique, nous nous concentrons sur un terminal tri-standard, pouvant exécuter alternativement l'IEEE802.11, IEEE802.16 et le 3GPPLTE. En définissant la technique des Opérateurs Communs, nous proposons deux méthodes d'implémentations distinctes :

- La première où l'OC est ordonnancé dans le temps.
- La deuxième où l'OC est organisé en banc.

Que ce soit dans le cadre des OC FFT/Viterbi ou LFSR, la première méthode aboutit à la définition d'OC unitairement plus complexes que les structures qu'ils doivent remplacer et un gain en complexité final ne pourra être envisagé que si ces OC sont capables d'être exécutés à des vitesses supérieures à celles des structures qu'ils remplacent. Outre la complexité matérielle, utiliser des OC à des fréquences plus élevées entrainera des consommations dynamiques, elles-aussi plus élevées. Ces aspects résultants des premières implémentations ne sont pas les seuls freins à l'implémentation d'OC ordonnancés. Nous devons prendre en compte le temps de conception d'une telle architecture et le fait que l'agencement proposé pour un ensemble de standards donné ne puisse pas satisfaire aux contraintes d'un nouveau standard, non prévu initialement. **Ainsi, notre première conclusion relative aux implémentations est de privilégier l'implémentation en banc**. Celle-ci raccourci le temps de conception en proposant autant d'OC que nécessaires par fonction et évite une gestion complexe de l'ordonnancement. De plus, une conclusion majeure de notre étude reste que dans le cas des LFSR, la duplication régulière du même OC permet d'apporter une flexibilité à l'architecture tout en obtenant un gain en complexité par rapport à une implémentation Velcro. Ce résultat est significatif car là, où l'ajout de flexibilité était synonyme habituellement de hausse de complexité, la technique des OC en bancs apporte un gain. Cette caractéristique est d'autant plus remarquable qu'elle est couplée à une gestion simplifiée des OC. Cependant ce résultat n'est valable que dans le cas des LFSR, car nos travaux ont montré que d'autres OC comme l'OC FFT/Viterbi induit une hausse de la complexité matérielle. **Ainsi, non seulement la technique des OC répond à notre problématique d'architecture reconfigurable générique en temps réel**

mais dans certains cas, elle peut s'accompagner d'un gain en complexité.

La deuxième conclusion importante de nos travaux concerne le choix des OC. Dans la vision que nous proposons de l'OC, nous pouvons supposer à priori que chercher l'OC qui factorise au maximum les structures de la chaine de traitement serait l'optimum à privilégier. Si nous reprenons la représentation en graphe, du terminal multistandard, cette recherche consisterait à diminuer le niveau de granularité de l'OC et à "descendre dans le graphe" afin d'identifier un OC vers lequel converge la majorité des flèches provenant des opérations à exécuter. Nous avons déjà conclu que dans le cadre d'un OC ordonnancé, construire un OC à partir de plusieurs étages de granularité inférieure diminuait "dangereusement" la vitesse d'exécution possible pour l'OC de haut niveau en lien direct avec les fonctions à exécuter. Dans le cadre des BOC, nous constatons que développer un OC le plus générique possible n'est pas pertinent en termes de complexité. En effet, le R-LFSR CORDIC est défini pour l'exécution des opérations en registres à décalages et pour l'exécution des fonctions dans le domaine fréquentiel par l'intermédiaire d'un module CORDIC et FFT, or son implémentation est plus complexe qu'un module CORDIC et un R-LFSR, implémenté indépendamment. **Nous concluons par conséquent qu'un compromis est à faire sur le choix de l'OC entre généricité et "implémentabilité"**. Dans notre cas de figure, ce choix sera de privilégier d'un coté un OC CORDIC et de l'autre un OC LFSR. Cette distinction en deux OC, rejoint la distinction de classes que nous proposons et atteste de l'intérêt d'une recherche d'OC suivant les architectures et les données mises en jeu. Néanmoins, il est à noter que l'ER-LFSR qui présente une architecture plus complexe que les autres LFSR que nous proposons, est l'OC à privilégier et justifie ainsi, qu'au delà des premiers résultats empiriques que nous obtenons, une méthode formelle apparait comme nécessaire pour déterminer les OC à sélectionner.

Perspectives

Nous proposons un premier jeu d'OC et étudions différents cas d'implémentations afin de faire ressortir les premières conclusions quant à l'impact de la technique des OC. Suite à nos travaux, trois axes d'études peuvent être envisagés :

- Le premier concerne une méthode formelle de détermination du meilleur ensemble d'OC. Dans ce rapport, nous démontrons que privilégier une implémentation plutôt qu'une autre n'est pas trivial et que pour un terminal multistandard, où un grand nombre de combinaisons est possible, une méthode formelle de détermination des OC à choisir reste indispensable.
- Le deuxième point rejoint le premier et concerne l'implémentation de l'OC sur les différentes cibles technologiques possibles de la plateforme hybride et la définition d'une méthode de gestion du même OC mais défini sur des cibles distinctes.
- Finalement le dernier point est également relatif à la plateforme hybride et se focalise sur l'implémentation des OC sur cette dernière. Dans le cas des LFSR, nous définissons l'ER-LFSR pour seulement une opération à exécuter en plus du R-LFSR. Dans notre étude, cet OC apparait comme l'OC en registres à décalages à privilégier. Or, cette caractéristique peut ne pas être systématique et nous n'af-

firmons pas dans nos recherches qu'un terminal dans son intégralité puisse être factorisé par des OC, ainsi certaines structures devront être laissées en Velcro. Une étude de la plateforme dans son ensemble reste donc à mener.

Publications de l'auteur

Revues

- **Alaus L**, Palicot J, Roland C, Louët Y and Noguet D, Promising Technique of Parameterization For Reconfigurable Radio, the Common Operators Technique : Fundamentals and Examples, march 2009, Revue Journal of Signal Processing Systems, Éditeur Springer New York ISSN 1939-8018 (Print) 1939-8115 (Online), DOI 10.1007/s11265-009-0353-4.

- Gul S T, **Alaus L**, Moy C, Palicot J, Noguet D, Optimal set of LFSR Common Operators for Multi-Standards Cognitive Radio Terminals, International Journal of Autonomous and Adaptive Communications ; Special Issue on Cognitive Radio Systems (2009).

Brevets

- **Alaus L**, Noguet D, Opérateur d'addition et de soustraction générique adapté à la réalisation de fonctions FFT et Viterbi, Brevet Français - 09 58501.

- **Alaus L**, Noguet D, Processeur de traitement de données numériques à opérateur papillon pour l'exécution d'une FFT/IFFT et terminal de télécommunication, Brevet Français - 09 58495

Conférences Internationales

- Naoues M, **Alaus L**, Noguet D, A Common Operator for FFT and Viterbi algorithms, 13th EUROMICRO Conference on Digital System Design, 1-3, September 2010, University of Lille, Lille, France.

- Wang H, **Alaus L**, Louet Y, Palicot J, Noguet D, Memory-efficient FFT architecture using R-LFSR based CORDIC common operator, International Worhshop on Cognitive Information Processing 2010, 15-16, June 2010, Elba Island, Italy.

- **Alaus L**, Noguet D, Palicot J, A Common Operator Bank to Resolve Scheduling Issue on a SDR Terminal, 6th Advanced International Conference in Telecommunication, 9-15, May 2010, Barcelona, Spain.

- Wang H, **Alaus L**, Louet Y, Palicot J, Noguet D, A new CORDIC architecture based on R-LFSR Common Operators in a Software Radio context, 6th Karlsruhe Workshop on Software Radios, 11-13 March 2010.

- **Alaus L**, Noguet D, Palicot J, A New Reconfigurable Linear Feedback Shift Register Organization To Improve SDR Design, 3rd international conference on Signals, Circuits and Systems (SCS). 9-11 november 2009, Jerba, Tunisia.

- **Alaus L**, Noguet D, Palicot J, A Reconfigurable Linear FeedBack Shift Register Operator for Software Defined Radio Terminal, IEEE International Symposium on Wireless Pervasive Computing (ISWPC-2008),6-9 may 2008, Santorini, Greece.

- **Alaus L**, Noguet D, Palicot J, Extented Reconfigurable Linear FeedBack Shift Register Operators for Software Defined Radio, 10th International Symposium of Spread Spectrum Techniques and Applications 2008, 25-28 August 2008,Bologna, Italia.

- **Alaus L**, Noguet D, Palicot J, A Reconfigurable LFSR for tri-standard SDR

transceiver, architecture and complexity analysis, 11th EUROMICRO Conference on Digital System Design, 3-5,September 2008, University of Parma, Parma, Italy.

Conférences Nationales

- **Alaus L**, Palicot J, Noguet D, Réalisation d'un Terminal Multi-Standards : Performances et Avantages de la Technique des Opérateurs Communs, illustrés par l'ER-LFSR, XXIIe Colloque GRETSI - Traitement du Signal et des Images, - Dijon - 8 au 11 septembre 2009.

Conférences Internationales - Workshops

- Gul S T, **Alaus L**, Noguet D, Moy C, Palicot J, The Common Operator Technique : An Optimization Process to Identify and Design a Set of Common Operators to Perform SDR Equipment, International Conference in Telecommunications Mobile Summit'09, 10-12 June 2009, Santander, Spain.

- **Alaus L**, Noguet D, Palicot J, A Bank of Coarse Grain Common Operators for Flexible Multistandard SDR Terminal, European Reconfigurable Radio Technologies Workshop and Exibition'10, 23-25 June 2010, Mainz, Germany.

Séminaires

- **Alaus L**, Noguet D, Palicot J, "Techniques de Paramétrisation et Opérateurs Communs : Apports et Limites, Exemples des Opérateurs à Registres à Décalages ", Séminaire SCEE du 29 Mai 2008, SUPELEC, Campus de Rennes.

- **Alaus L**, Noguet D, Palicot J, " Techniques des Opérateurs Communs pour la Conception d'un Terminal Multistandard Reconfigurable : Apports et Limites, Exemples des LFSRs", Commission Télécommunication du Conseil Scientifique, SUPELEC, 22 Octobre 2008.

- **Alaus L**, Noguet D, Palicot J, "Architecture Reconfigurable pour un Equipement Radio Multistandard.", Séminaire SCEE du 20 Mai 2010, SUPELEC, Campus de Rennes.

Projets

Projet ANR INFOP

- Lot 4 - Livrable 1 : Rapport sur les Fonctions Communes aux différents standards.
- Lot 4 - Livrable 2 : Rapport sur le Graphe Fonctionnel.

Projet NEWCOM ++

- WPRC - DC 1 : Report on the state for the art on hardware architectures for flexible radio and intensive signal processing.
- WPRC - DC 2 : Intermediate report on flexible architectures using various type of flexibility and reconfigurability.

Annexes

Annexe A

Le FPGA Détaillé

Sommaire

A.1 Préambule

Depuis son invention par Xilinx en 1984, le FPGA ne se contente pas de son rôle d'interfaçage d'appoint mais remplace les ASIC (circuits intégrés à application spécifique) et les processeurs personnalisés dans des applications de contrôle et de traitement de signaux. Aujourd'hui, le FPGA apparait comme un élément indispensable à la Radio logicielle Restreinte. Voici encore une quinzaine d'années, configurer une carte électronique entière, ou plusieurs cartes d'une même plate-forme pour répondre aux besoins de tous les éléments d'un système, concevoir rapidement des composants matériels système tels que : microprocesseurs, périphériques, filtres, boucles de contrôle, UART afin de répondre exactement aux besoins d'une application semblait inimaginable. La majorité des blocs logiques standards comme les contrôleurs Ethernet, CAN ou USB, ainsi que les contrôleurs de microprocesseurs et de mémoire, et enfin les UART, étaient considérés comme de simples puces en logique fixe.

Aujourd'hui, les FPGA atteignent des tailles et des prix très compétitifs, à tel point que même un microprocesseur 32 bits ne représente plus qu'une petite partie du coût et de la taille d'un FPGA. Ils sont donc une solution viable pour des applications qu'il s'agisse d'intégrer des FPGA dans des matériels, de les utiliser pour tester des machines, ou encore d'employer du matériel à base de FPGA pour contrôler et fabriquer des produits. Un FPGA est un circuit en silicium reprogrammable. À l'aide de blocs logiques préconstruits et de ressources de routage programmables, ce circuit peut être configuré afin de mettre en oeuvre des fonctionnalités matérielles personnalisées. Des tâches de traitement numérique sont développées par logiciel et ensuite compilées sous forme de fichier de configuration ou de flux de bits contenant des informations sur la

manière dont les composants doivent être reliés. Les FPGA sont omniprésents et, grâce à la disponibilité d'un nombre incalculable de blocs d'IP ciblant les FPGA, ils sont capables de prendre en charge plusieurs applications avec une seule et même carte.

Les solutions alternatives sur carte sont généralement composées d'un microprocesseur ou d'un ASSP (Application-Specific Standard Product) et d'une logique fixe associée, le tout dans une architecture rigide, ce qui limite les possibilités d'atteindre de hautes performances. Une carte basée sur FPGA présente une architecture pouvant être personnalisée, pour des performances accrues. Les tâches peuvent y être réparties de façon optimale entre matériel et logiciel, et peuvent y être implémentées de manière parallèle en ajoutant des microprocesseurs softcore, en dupliquant les blocs de fonctions matérielles, ou en ajoutant des coprocesseurs directement au microprocesseurs eux-mêmes.

A.2 Intérêt de la Technologie FPGA

D'un point de vue pratique, les FPGA réunissent le meilleur des ASIC et des systèmes basés processeurs. Ainsi, ils offrent un cadencement par matériel qui leur assure vitesse et fiabilité, mais sont plus rentables que les ASIC personnalisés. Les circuits reprogrammables jouissent également de la même souplesse d'exécution logicielle qu'un système basé processeur et sont moins affectés par le nombre de coeurs de traitement disponibles. Contrairement aux processeurs, les FPGA sont vraiment parallèles par nature, de sorte que plusieurs opérations de traitement différentes ne se trouvent pas en concurrence pour l'utilisation des ressources. Chaque tâche de traitement indépendante est affectée à une section spécifique du circuit, et peut donc s'exécuter en toute autonomie sans dépendre aucunement des autres blocs logiques. En conséquence, le volume de traitement effectué peut croitre sans que les performances d'une partie de l'application n'en soient affectées pour autant.

Performances Comme ils tirent parti du parallélisme matériel, les FPGA offrent une puissance de calcul supérieure à celle des processeurs de signaux numériques (DSP), car ils s'affranchissent du modèle d'exécution séquentielle et exécutent plus d'opérations par cycle d'horloge. Contrôler les entrées et sorties (E/S) au niveau matériel permet d'obtenir des temps de réponse plus courts ainsi que des fonctionnalités spécifiques, qui répondent mieux aux besoins de l'application.

Coût Les coûts d'ingénierie non récurrents (NRE) des ASIC sont bien supérieurs à ceux des solutions matérielles basées sur du FPGA. L'important investissement de départ que requièrent les ASIC trouve une justification chez les fabriquants qui peuvent livrer des circuits par milliers. Cependant, la plupart des utilisateurs finaux ont besoin de matériels personnalisés pour quelques dizaines ou quelques centaines de systèmes en développement. Par nature, les circuits programmables n'impliquent ni coût de fabrication, ni longs délais d'assemblage. Les besoins de la plupart des systèmes évoluent

avec le temps ; or la modification progressive d'un FPGA représente un coût négligeable comparé à la dépense considérable qu'exige la reconception d'un ASIC.

Fiabilité Tandis que les outils logiciels fournissent l'environnement de programmation, les circuits FPGA sont une véritable implémentation matérielle de l'exécution logicielle. Les systèmes basés processeur comprennent souvent plusieurs couches d'abstraction, pour aider à la planification des tâches et à la répartition des ressources entre les différents processus. La couche de driver contrôle les ressources matérielles et le système d'exploitation gère la mémoire et la bande passante du processeur. Sur chaque coeur de processeur, une seule instruction peut s'exécuter à la fois ; c'est pourquoi les systèmes basés processeur risquent toujours de voir des tâches prioritaires entrer en conflit. Les FPGA, qui n'utilisent pas de système d'exploitation, minimisent les problèmes de fiabilité car ils assurent une exécution véritablement parallèle et un matériel déterministe dédié à chaque tâche

Maintenance à long terme Comme nous l'avons vu, les circuits FPGA sont évolutifs et épargnent la dépense de temps et d'argent qu'implique la reconception des ASIC. Les spécifications des protocoles de communication numériques, par exemple, évoluent avec le temps. Or les interfaces basées sur ASIC peuvent poser des problèmes de maintenance et de compatibilité. Comme les FPGA sont reconfigurables, ils permettent d'intégrer des améliorations fonctionnelles sans perdre de temps à reconcevoir le matériel ou à modifier l'implantation du circuit.

A.3 Détail de la Technologie FPGA

Architecture Générale Chaque circuit intégré FPGA est constitué d'un nombre limité de ressources prédéfinies avec des interconnexions programmables pour mettre en oeuvre un circuit numérique reconfigurable. Dans les spécifications d'un circuit intégré FPGA, il est spécifié la quantité de blocs logiques configurables, le nombre de blocs logiques de fonctions figées, comme les multiplicateurs et la taille des ressources de la mémoire telles que le bloc de RAM embarquée (embedded block RAM). Un circuit intégré FPGA est composé de plusieurs autres éléments, mais ceux-ci sont typiquement les plus importants lors du choix et de la comparaison des FPGA en vue d'une application spécifique.

Les bloc logiques configurables, tels que les slices (tranches) ou les cellules logiques élémentaires (figure G.3), sont constitués de deux éléments essentiels : des tables de correspondance (LUT ou Look-Up-Table) et des bascules Flip-Flop. Les diverses familles des FPGA se distinguent par la façon dont les bascules et les LUT sont conditionnées ensemble. Les FPGA de la famille Virtex-II, par exemple, sont pourvus de slices avec deux LUT et deux bascules, tandis que ceux de la famille Virtex-5 ont des slices avec quatre LUT et quatre bascules. L'architecture des tables de correspondance elle-même peut être différente (quatre entrées au lieu de six).

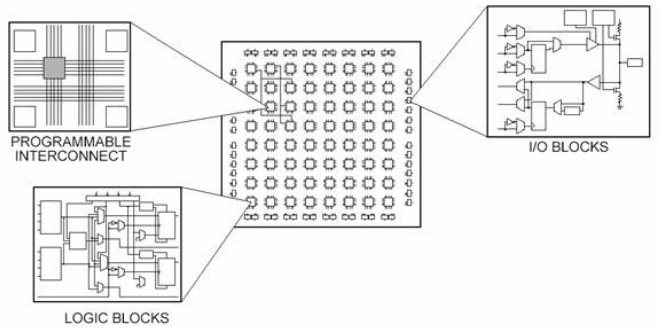

FIGURE A.1 – Architecture d'un FPGA

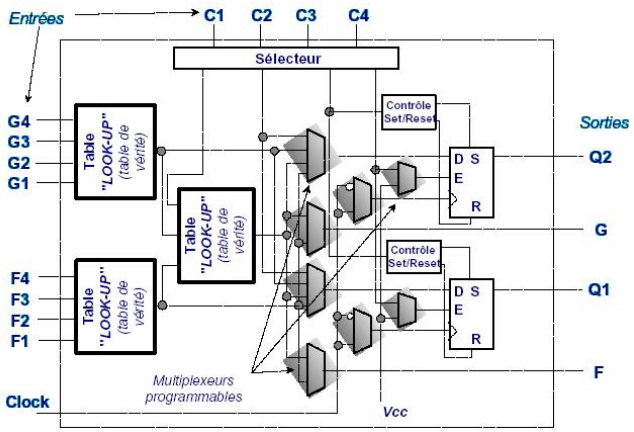

FIGURE A.2 – Detail du Logic Block de la Figure A.1

Les Bascules Les bascules sont des registres à décalage binaires qui servent à synchroniser la logique et à enregistrer les états logiques entre les périodes d'horloges. À chaque front d'horloge, une bascule déclenche la valeur 1 ou 0 (VRAI ou FAUX) sur son entrée et maintient cette valeur stable jusqu'au prochain front d'horloge. Dans des certains systèmes de placement routage propriétaire, une bascule peut être placé entre chaque opération afin d'optimiser le temps de propagation disponible pour l'exécution de chaque opération. L'exception à cette règle se produit lorsque le code est placé à l'intérieur d'une boucle cadencée à une seule période. Dans cette structure de boucle spécifique, les bascules sont ajoutées uniquement au début et à la fin de l'itération de boucle, et c'est au programmeur de comprendre les considérations spécifiques au cadencement.

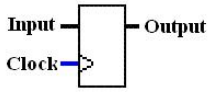

FIGURE A.3 – Schématique d'une Bascule

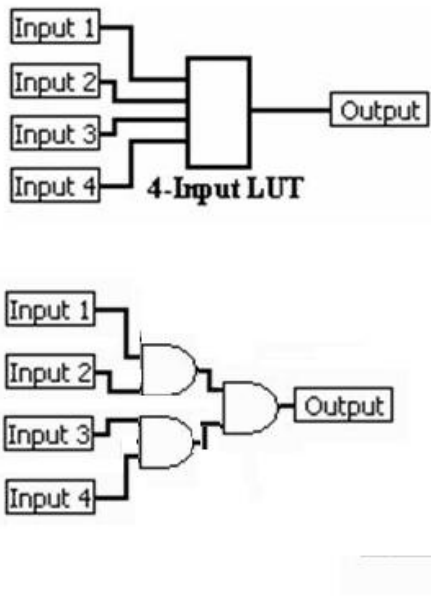

FIGURE A.4 – Schématique d'une Table de Correspondance (LUT)

Les LUTs La logique est mise en oeuvre grâce à de très petites quantités de RAM sous la forme de LUT. Il est facile de supposer que le nombre de portes du système

dans un FPGA se rapporte au nombre de portes NON ET et NON OU dans un circuit intégré spécifique, mais, en réalité, l'ensemble de la logique combinatoire (ET, OU, NON ET, XOU, etc.) est implémentée en tant que tables de vérité dans la mémoire LUT. Une table de vérité est une liste prédéfinie d'états de sorties pour chaque combinaison d'entrées :

*Input*1	*Input*2	*Input*3	*Input*4	*Output*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Par exemple, les FPGA des familles Virtex-II et Spartan-3 sont pourvus de tables de correspondance à quatre entrées pour mettre en oeuvre des tables de vérité avec jusqu'à 16 combinaisons de quatre signaux d'entrée. La figure A.4 est un exemple de mise en oeuvre de circuit à quatre entrées. Les FPGA de la famille Virtex-5 utilisent des LUT à six entrées pour mettre en oeuvre des tables de vérité avec jusqu'à 64 combinaisons de six signaux d'entrée différents.

Les Multiplicateurs L'opération de multiplication de deux nombres peut s'avérer extrêmement consommatrice de ressources et difficile à mettre en oeuvre dans un circuit numérique. Pour fournir un certain cadre de référence, le Figure A.5 représente le schéma d'une des options d'implémentation d'un multiplicateur 4 bits par 4 bits utilisant la logique combinatoire. La multiplication de deux nombres de 32 bits engendrerait au final plus de 2000 opérations pour une seule multiplication. De ce fait, les FPGA récents sont dotés d'un circuit de multiplicateurs préconstruit pour économiser l'utilisation des LUT et des bascules dans les applications de traitement du signal et de mathématiques. Les FPGA des familles Virtex-II et Spartan-3 ont des multiplicateurs 18 bits x 18 bits, si bien que multiplier deux nombres de 32 bits requiert trois multiplicateurs pour une seule opération. De nombreux algorithmes de traitement du signal nécessitent de conserver le cumul des nombres multipliés, et, en conséquence, les FPGA aux performances plus

élevées comme ceux de la famille Virtex-5 ont un circuit préconstruit multiplicateur-accumulateur (MAC). Ces blocs de traitement préconstruits, également appelés DSP48 slices, intègrent un multiplicateur 25 bits x 18 bits avec un circuit additionneur.

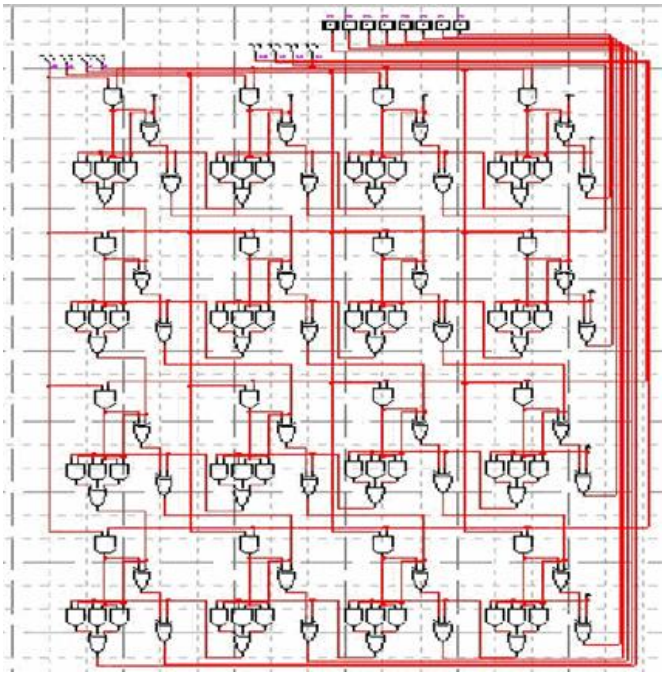

FIGURE A.5 – Architecture d'un Multiplieur

La RAM Les ressources de mémoire sont une autre spécification clé à prendre en compte lors du choix des FPGA. Une mémoire RAM définie par l'utilisateur, embarquée sur le circuit intégré FPGA, est utile pour le stockage d'ensembles de données ou faire passer des valeurs entre des boucles parallèles. Selon la famille de FPGA, il convient de configurer la RAM embarquée en blocs de 16 ou 36 kbits. Il y a toujours la possibilité d'implémenter les ensembles de données sous forme de tableau utilisant des bascules. Toutefois, les tableaux volumineux deviennent rapidement fort consommateurs de ressources de logique FPGA. Par exemple, un tableau comportant 100 éléments de nombres de 32 bits pourrait consommer plus de 30% des bascules dans un FPGA de la famille Virtex-II 1000 ou occuper moins de un pour cents du bloc de RAM embarquée. Les algorithmes de traitement de signaux numériques ont souvent besoin de conserver la trace d'un bloc entier de données, ou des coefficients d'une équation complexe, et sans mémoire embarquée, de nombreuses fonctions de traitement ne pourraient pas s'intégrer dans la logique configurable d'un circuit intégré FPGA.

Il est également possible d'utiliser des blocs de mémoire afin de contenir des données de formes d'ondes périodiques pour la génération de signaux embarquée en stockant une période complète sous la forme d'une table de valeurs et en indexant la table de manière

séquentielle. La fréquence du signal de sortie est déterminée par la vitesse à laquelle les valeurs sont adressées. Cette méthode peut être utilisée afin de modifier de manière dynamique la fréquence de sortie sans introduire de transition prononcée dans la forme d'onde. L'exécution parallèle propre aux FPGA permet aux portions indépendantes de la logique matérielle d'être commandées par différentes horloges. Transmettre les données entre des logiques fonctionnant à des fréquences différentes peut s'avérer difficile, et la mémoire embarquée est fréquemment utilisée dans le but de lisser le transfert à l'aide de buffers FIFO (first-in-first-out ou premier entré, premier sorti). Il est possible de configurer les buffers FIFO, pour qu'ils aient des tailles différentes et garantir que les données ne soient pas perdues entre des parties asynchrones du circuit intégré FPGA.

Annexe B

Approche Graphique adaptée au Contexte d'Etude.

Sommaire

B.1 Décomposition Graphique à considérer

En [MPRG06] une équation du terminal SDR est proposée sans ternir compte du cadencement des OC_k et de la possibilité de dupliquer des instances physiques et d'allouer les fonctions sur les instances virtuelles dans le temps. En chapitre 2, nous présentons ces possibilités et la gestion que les OC_k nécessitent. En Annexe C, nous définissons une première méthodologie d'allocation des instances virtuelles sur les instances physiques qui définissent un nombre d'instances physiques à implémenter N_{imp}. Ce nombre N_{imp} n'est valable que lorsqu'un OC_k est déjà affecté à des fonctions spécifiques. Comme une fonction peut être réalisée par différentes combinaisons d'OC_k, pour un même standard j, plusieurs "versions" de ce standard seront possibles selon quels OC_k sont attribués pour une fonction donnée. Ainsi, pour un standard j, dans une de ces configurations c ($S_{j,c}$), nous pourrons déterminer un nombre d'instances $N_{imp,j,c,k}$. Ce dernier nombre $N_{imp,j,c,k}$ permet ainsi d'introduire le cadencement des OC_k. Il est à noter que dans le cas d'un BOC, le calcul de $N_{imp,j,c,k}$ est immédiat est égal à $\sum_i N_{b,i,j,c,k}$, où pour un OC_k de $S_{j,c}$, nous retrouvons $N_{b,i} = E(\frac{N_{T,i}.T_e}{T_{f,i}}).N_{c,i}$, tel que défini en chapitre 2 pour une f_i utilisant OC_k.

Pour bien comprendre, comment déterminer le coût du terminal SDR, nous revenons sur le graphe défini par rodriguez en [MPRG06]. Nous devons revenir sur le graphe dans son ensemble en se rappelant le postulat de départ d'un seul standard exécuté à la fois. Si nous prenons un exemple simple de la figure B.1. Nous sommes en présence

de deux fonctions F1 et F2 qui peuvent chacune être implémentée par les Opérateurs Communs A et B. Dans notre exemple, A et B ont un cout monétaire identique de 100 et respectivement un coût computationnel de 20 et 10. Les différentes liaisons ont des valeurs de (F1,A) : 5, (F1,B) : 2, (F2,A) : 5 et (F2, B) :11.

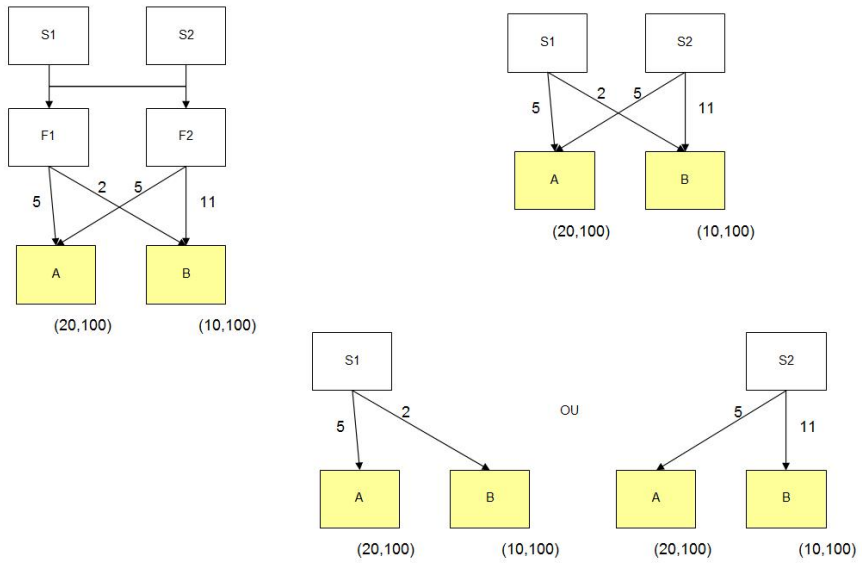

FIGURE B.1 – Adaptation de la méthode au contexte d'étude

Ainsi, pour exécuter F1 et F2, nous avons trois possibilités l'opérateur A, B ou (A et B). En utilisant l'hypothèse qu'une seule instance d'un même opérateur peut exécuter les différentes opérations requises alors parmi les quatre possibilités, le choix de réaliser F1 et F2 par l'opérateur B est la meilleur solution :

Implémentation Possible	Coût Global
Cas 1 : A	(5+5) x 20+100 = 300
Cas 2 : B	(2+11) x 10+100 = 230
Cas 3-a : A et B	11 x 10 + 100 + 5 x 20 + 100 = 410
Cas 3-b : A et B	5 x 20 +100 + 2 x 10 +100 = 320

TABLE B.1 – Exemple de choix avec des Fonctions

Si maintenant, nous considérons que F1 et F2 ne sont plus des fonctions mais des standards S1 et S2, le même algorithme donne exactement les mêmes résultats. Or ce résultat est pertinent si les deux standards S1 et S2 sont requis en "même temps". Comme nous faisons l'hypothèse de départ de ne travailler que sur un seul standard à la fois, nous devons modifier le principe de l'algorithme. En effet, au lieu de considérer que S1 et S2 sont exécutés en même temps, alors nous devons considérer que le terminal exécute l'un ou l'autre sous la condition qu'il puisse exécuter les deux alternativement.

Ainsi, nous n'avons plus besoin de nous focaliser sur la somme des couts NoC provenant de S1 et S2 pour un même opérateur mais seulement sur le nombre maximum qui permet de réaliser les deux. Les standards sont représentés et reliés par les mêmes liaisons que les fonctions et les opérateurs. Or ces liaisons caractérisent un besoin constructif des opérateurs les uns par rapport aux autres. C'est-à-dire que cette représentation graphique donne les meilleurs opérateurs pour un ensemble de standards qui requièrent en même temps les opérateurs. Au niveau des standards la schématique utilisée ne permet pas d'introduire l'alternance dans l'utilisation des standards. Notre cas d'étude dans nos recherches se focalise justement sur un terminal qui exécute en alternance les différents standards. Si maintenant, nous reprenons l'exemple précédent, pour être exécuté en même temps, S1 et S2 requièrent ; 2 appels de B pour S1 et 11 pour S2, ce qui représente 13 appels pour les deux. En considérant l'exécution de l'un ou de l'autre, il ne sera nécessaire de ne considérer que soit 2 appels soit 11 appels de B. Donc le coût computationnel de B pour le terminal ne sera plus de de 2 + 11 = 13 mais de max(2,11) = 11, car avec 11 appels de B pour S2, nous pourrons réaliser les 2 appels de B pour S1. Ainsi, nous n'avons plus qu'à considérer deux cas de figures, et dans cette configuration, le choix de l'opérateur à implémenter n'est plus B mais A :

Implémentation Possible	Coût Global
Cas 1' : A	max(5+5) x 20+100 = 200
Cas 2' : B	max(2+11) x 10+100 = 210

TABLE B.2 – Exemple de choix avec des Standards

B.2 Mise en Equation

Nous pouvons donc proposer une procédure de détermination du meilleur set d'opérateurs. Ce set d'opérateurs doit être capable d'exécuter chaque standard individuellement. Concrètement, pour deux sets set 1 et set 2 valables respectivement pour le Standard S1 et le standard S2. Nous ne pouvons directement sélectionner un de deux sets que s'il est capable d'exécuter S1 et S2. Dans le cas contraire, nous devons construire un set hybride S_H qui sera constitué du maximum des besoins des deux sets opérateurs par opérateurs. Ce point correspond à l'exemple donnée précédemment entre S1 et S2.

Pour un set d'opérateurs donné valable pour un standard, nous devrons évaluer des sets intermédiaires (hybrides) à partir de tous les sets possibles pour les autres standards rendant le problème complexe. Ainsi, nous proposons la procédure suivante :

1. Evaluer tous les sets possibles d'opérateurs pour un standard.
2. Construire tous les sets hybrides en fonction des sets précédents.
3. Déterminer le "coût" de chaque set.
4. Sélectionner le meilleur set.

Si nous considérons le standard $S_{j,c}$, standard S_j, réalisée par une combinaison c d'OC, alors nous pouvons écrire :

$$S_{j,c} = \sum_{k=1}^{N} N_{imp,j,c,k}.OC_k \tag{B.1}$$

Où N est le nombre d'OC distincts. Pour que le terminal puisse réaliser chaque standard en alternance, alors nous devons étudier l'ensemble des combinaisons possibles entre les différents $S_{j,c}$, c'est à dire l'ensemble des sets hybrides $\{S_{H,\chi}\}$:

$$S_{H,\chi} = \sum_{k=1}^{N} max_{j,\chi} N_{imp,j,c,k}.OC_k \tag{B.2}$$

Ici χ représente l'ensemble des sets hybrides possibles, c'est à dire un set obtenu à partir d'un ensemble de standards (j,c). Ainsi, le set minimal $S_{H,\chi}$ sera celui qui minimise la fonction de coût du terminal :

$$C_{SDR} = min_{\chi}(\sum_{k=1}^{N} max_{j,\chi}(N_{imp,j,c,k}.C(OC_k))) \tag{B.3}$$

En utilisant cette équation du terminal SDR, nous pouvons déterminer le meilleur set de d'un terminal multistandard. De plus, en considérant tous les cas possibles d'affectation des fonctions sur les OC, nous obtiendrons l'ensemble des possibilités d'ordonnancement représentées par un $N_{imp,j,c,k}$ pour chaque $S_{j,c}$. La résolution d'une telle équation qui intègre les possibilités de cadencement est un problème complexe qui n'est pas traité dans cette thèse mais qui est un des éléments clés de la technique des OC.

Annexe C

Algorithme d'Allocation des Instances Virtuelles

Sommaire

C.1 Méthode

En maximisant l'utilisation des allocations sur les instances virtuelles, il apparait possible de minimiser le nombre d'instances physiques N_{imp}. Ce nombre d'instances sera ici, déterminé à partir des différents temps $T_{f,i}$ auquel l'opérateur est contraint et des valeurs de $N_{c,i}$ et $N_{T,i}$ affiliées. Afin de minimiser le nombre N_{imp} nous proposons une procédure d'allocation. Celle-ci se base sur la figure 2.19 et aboutit à une première optimisation en figure C.1. Il est à noter que sur la figure C.1, seule une colonne d'opérateurs sera implémentée (instances physiques) et pour chaque exécution supposée synchrone des opérateurs, seule l'allocation des opérations sur ces opérateurs sera modifiée sur les instances virtuelles. L'idée de l'algorithme que nous proposons est de prendre les fonctions f_i selon leur ordre d'arrivée et "d'entasser autant que faire se peut", les blocs d'instances $N_{c,i}$ lors des temps T_e alloué.

Ainsi, nous proposons de construire un vecteur de valeurs Table[j]. Chaque colonne j de Table[j] "représente" le nombre d'instances virtuelles encore disponibles durant le temps $j.T_e$ (Pour rappel T_e est le temps d'exécution de l'OC). L'idée du processus est d'allouer pour chaque nouvelle fonction considérée ces instances virtuelles en diminuant les valeurs de Table[j]. Si le nombre d'instances disponibles n'est pas suffisant, il faudra alors augmenter Table[j] sur les temps T_e considéré ce qui affectera le nombre d'instances physiques N_{imp}. Ainsi, en définissant un vecteur Table[j], chaque valeur "j" représentera le nombre d'instances virtuelles du même opérateur non allouées. Pour

chaque opération f_i prise dans l'ordre chronologique de réalisation, nous placerons sur les colonnes correspondantes, tant que possible les $N_{T,i}$ groupes de $N_{c,i}$ opérateurs. Si chaque valeur de Table[j] est inférieure à $N_{c,i}$ et qu'il reste un nombre $N_{Tres,i}$ d'appels à allouer alors nous devrons augmenter la taille de la colonne d'opérateurs.

Concrètement pour une fonction donnée f_i et pour un état donné j du vecteur d'allocation, chaque valeur de Table[j], sera potentiellement différente. Si nous considérons la fonction f_i et que nous définissons k_{di} et k_i tel que $T_{d,i} = k_{d,i}.T_e$ + a et $T_{f,i} = k_i.T_e$ + b (a et b $< T_e$) alors f_i s'étend sur les colonnes de k_{di} à $k_{di} + k_i$, alors chaque colonne pourra accueillir $E[\frac{table[j]}{N_{c,i}}]$ soit un total de $N_{Tacceuilli} = \sum_{j=k_{di}}^{k_{di}+k} E[\frac{table[j]}{N_{c,i}}]$.

Deux cas de figures sont possibles, (cas 1) soit le nombre $N_{Tacceuilli}$ est supérieur à $N_{T,i}$ soit (cas 2) inférieur. Dans le premier cas de figure, il est possible de mapper f_i sur la colonne d'opérateurs sans en modifier la taille. Dans le second cas, il restera à allouer $N_{Trest,i}$ groupes de $N_{c,i}$ opérateurs, où :

$$N_{Trest,i} = N_{T,i} - \sum_{j=k_{di}}^{k_{di}+k} E[\frac{table[j]}{N_{c,i}}] \tag{C.1}$$

Pour procéder à cette allocation, nous devons augmenter la taille de la colonne d'une valeur $N_{impplus,i}$:

$$N_{impplus,i} = E[\frac{N_{T,i} - \sum_{j=k_{di}}^{k_{di}+k} E[\frac{table[j]}{N_{c,i}}]}{k}].N_{c,i} - min_{j,j\in[k_{d,i},k_{d,i}+k_i]}(Table[j] - E[\frac{Table[j]}{N_{c,i}}]) \tag{C.2}$$

En effet, si nous sommes dans le cas 2, c'est à dire que toutes les valeurs de table[j] $< N_{c,i}$, pour chaque table[j], il y aura encore un nombre identique d'instances libres que nous pouvons utiliser de valeur : $min_{j,j\in[k_{d,i},k_{d,i}+k_i]}(Table[j] - E[\frac{Table[j]}{N_{c,i}}])$.

C.2 Application sur l'Approche Simplifiée

Nous pouvons illustrer l'algorithme en se basant sur le passage de la figure 2.20 à la figure C.1. En effet, une possibilité d'implémentation diminuant le nombre d'opérateurs de 23 à 13 est représentée en figure C.1. La figure C.1 illustre parfaitement le concept de l'opérateur commun appliqué à un seul et même standard. En utilisant et ré-utilisant, l'opérateur à la fréquence F_e, tout en respectant les contraintes de temps imposées par chaque f_i, alors nous pouvons proposer une allocation optimisée du besoin de chaque f_i sur les instances physiquement implémentées. Dans cette exemple, seulement 13 instances du même opérateur seront réellement implémentées. Et pour chaque T_e, nous devrons procéder à une réallocation.

Explication de l'allocation :

- Première étape. Nous considérons f_1 qui s'étend sur les trois premiers T_e ($k_{d,1}$ = 1 et k_1 = 3, avec $N_{c,1}$ = 3 et $N_{T,1}$=4. Ainsi, $N_{b,1}$ = 6. Le nombre d'instances minimum à implanter pour f_1 est de 6. Nous initialisons le vecteur d'allocation

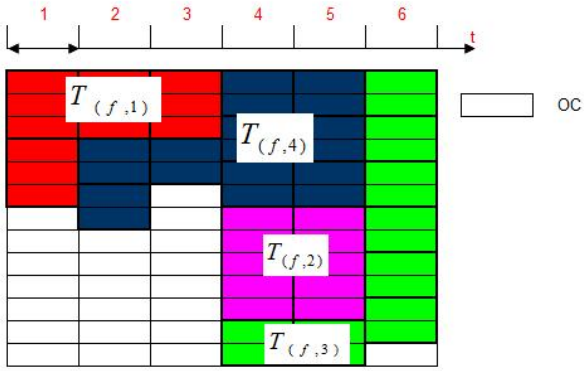

FIGURE C.1 – Introduction de la donnée temporelle - Réallocation des Instances

table à [6,6,6,6,6,6]. En considérant l'allocation des $N_{T,1}$ appels de f_1, la table d'allocation devient [0,3,3,6,6,6].

- Deuxième étape. Nous considérons f_4 qui s'étend sur quatre T_e ($k_{d,4} = 2$ et $k_4 = 4$, avec $N_{c,4} = 2$ et $N_{T,4}= 9$. Ainsi, nous pouvons allouer respectivement 1,1,3,3 $N_{c,4}$ sur table[j] pour j allant de 2 à 5. Nous n'avons alloué que 8 $N_{T,4}$, obtenu l'état de table = [0,1,1,0,0,6] et il reste à allouer 1 $N_{T,4}$ de taille $N_{c,4} = 2$. Comme min(table(2),table(3))=1, alors $N_{implplus,4} = 1$. Ainsi après l'étape 2, table devient [1,0,2,1,1,7].
- Troisième étape. Nous considérons f_2 qui s'étend sur deux T_e ($k_{d,2} = 4$ et $k_2 = 2$, avec $N_{c,2} = 5$ et $N_{T,2}= 2$. Ici, il n'y a pas suffisamment d'instances durant les temps T_e de rang 4 et 5, donc nous devons augmenter la taille de table de 5 - min(1,1) = 4 et faire passer le nombre d'instances de 7 à 11. Ainsi, après allocation table devient [5,4,6,0,0,11].
- Quatrième étape. Nous considérons f_3 qui s'étend sur trois T_e ($k_{d,2} = 4$ et $k_3 = 3$, avec $N_{c,3} = 2$ et $N_{T,3}= 8$. Ainsi, nous pouvons allouer respectivement 0,0,5 $N_{c,3}$ sur table[j] pour j allant de 4 à 5. Ainsi, nous n'avons alloué que 5 $N_{T,3}$ et obtenu l'état de table = [5,4,1,0,0,1]. Il reste à allouer 3 $N_{T,3}$ de taille $N_{c,3} = 2$. Comme min(table(2),table(3))=0, alors $N_{implplus,3} = 2$. Ainsi après l'étape 4, table devient [7,6,8,0,0,1]. Le nombre N_{imp} obtenu est donc de 6 + 1 + 4 + 2 = 13 instances.

Dans cet exemple, nous devons implémenter 13 instances de l'opérateur. Nous remarquons clairement que même si nous économisons des instances de l'opérateurs communs, celles-ci sont principalement toutes utilisées durant les mêmes T_e. De manière empirique, nous pouvons définir le paramètre $R_i = N_{c,i}/k_i$. Ce paramètre R_i permet de qualifier la répartition de la taille des instances sur les temps T_e. En effet, nous remarquons que l'allocation de $N_{c,i}$ de taille importante durant un temps $k_i.T_e$ court pénalise la procédure. Ici, en se basant sur le R_i nous désirons allouer dans un premier temps les blocs de taille $N_{c,i}$ grande sur un k_i petit et ensuite allouer les blocs de taille $N_{c,i}$

petit sur des k_i grand. Si nous classifions non plus les f_i par leurs ordre $k_{d,i}$ d'apparition mais selon leurs R_i du plus grand au plus petit, le résultat nous permet d'obtenir une allocation plus homogène et de diminuer le nombre d'instances à implémenter (Figure C.2).

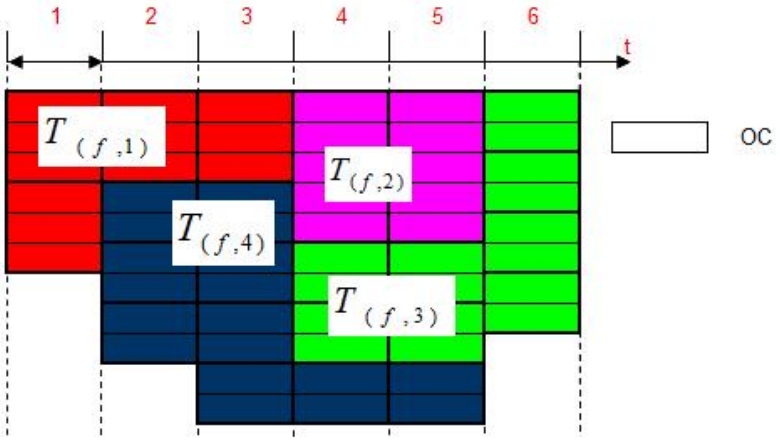

FIGURE C.2 – Amélioration de la réallocation

C.3 Application sur la Gestion du Flux

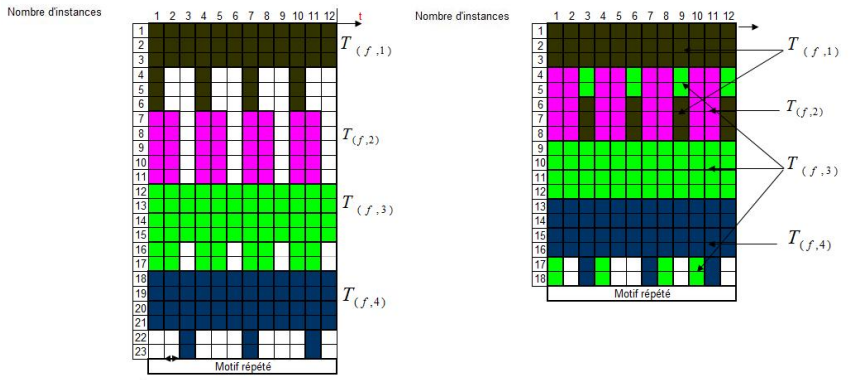

FIGURE C.3 – Enchainement des traitements - B - Motif Repeté et Réallocation

Nous pouvons également appliquer cette procédure sur un enchaînement d'opéra-

tions. Pour y arriver, nous pouvons appliquer la même procédure que précédemment, en considérant la succession des fonctions selon leurs ordres d'apparition et au fur et à mesure des appels de ces fonctions, allouer et progressivement définir le nombre d'instances à implémenter. Or, comme nous travaillons avec systématiquement le même enchainement de fonctions, chacune opérant à une fréquence donnée, c'est à dire avec un temps $T_{f,i}$ défini, nous pouvons simplifier le problème. En effet, nous travaillons avec un ensemble Q de f_i de temps de fonctionnement T_{fi} tel que $T_{fi} = k_i \, . \, T_e$. Si nous considérons chaque succession de fonctions, celle-ci apparaitra de manière périodique dans le treillis de la figure 2.23. Nous pouvons en déduire que nous retrouverons le même agencement des fonctions ("motif") pour un T_F = K.T_e où K est le plus petit commun multiple des différents k_i, K = $PPCM_i(k_i)$. Dans les deux cas de figure considérés (A et B), le PPCM est de 12 (figure 2.23). Il est à noter que ce motif ne se répètera qu'à partir du moment où toutes les opérations ont débutés et tant que toutes les opérations sont exécutées. C'est à dire, qu'en début et fin du treillis, un agencement différent est requis. Néanmoins, ce motif représentera le besoin maximum d'utilisation de l'opérateur commun sur le temps T_F.

Par conséquent, nous pouvons appliquer, la procédure précédente sur la fenêtre de taille 12.T_e et définir le motif répété de l'opérateur alloué. Nous illustrons ce cas de figure sur le cas (B) en figure C.3.

Annexe D

Les Systèmes FIR, IIR et les Différents types de Registres à Décalages

Sommaire

D.1 Les Filtres Numériques

En électronique, un filtre numérique est un élément qui effectue un filtrage à l'aide d'une succession d'opérations mathématiques sur un signal discret. C'est-à-dire qu'il modifie le contenu spectral du signal d'entrée en atténuant ou éliminant certaines composantes spectrales non désirées. Contrairement aux filtres analogiques, qui sont réalisés à l'aide d'un agencement de composantes physiques (résistances, condensateurs, inductances, transistors, etc.), les filtres numériques sont réalisés soit par des circuits intégrés dédiés, des processeurs programmables (FPGA, microprocesseur, DSP, microcontrôleur, etc.), soit par logiciel dans un ordinateur. Un filtre numérique peut être défini par une équation aux différences, c'est-à-dire l'opération mathématique du filtre dans le domaine temporel discret. La forme générale du filtre d'ordre M est la suivante :

$$y_n = \sum_{k=0}^{N} b_k \cdot x_{n-k} - \sum_{k=1}^{M} a_k \cdot y_{n-k} \tag{D.1}$$

La valeur des coefficients a et b fixera le type de filtre (passe-bas, passe-haut, etc) représentée par la fonction de transfert générale d'ordre N :

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b_k \cdot z^{-k}}{\sum_{k=0}^{M} a_k \cdot z^{-k}} \tag{D.2}$$

Il existe deux grandes familles de Filtres : (1) Filtres à réponse impulsionnelle finie (FIR - Finite Impulse Response) et (2) Filtres à réponse impulsionnelle infinie (IIR - Infinite Impulse Response). Le premier type est dit fini, car sa réponse impulsionnelle se stabilisera ultimement à zéro. Un filtre FIR est non récursif, c'est-à-dire que la sortie dépend uniquement de l'entrée du signal, il n'y a pas de contre-réaction. Ce filtre se forme à partir de l'équation précédente avec les coefficients (a_k) égaux à zéro. Une propriété importante des filtres FIR est que les coefficients du filtre (b_k) sont égaux à la réponse impulsionnelle h du filtre. D'autre part, la forme temporelle du filtre est tout simplement la convolution du signal d'entrée x avec les coefficients (ou réponse impulsionnelle) (b_k) (ou h). En opposition, le deuxième type de filtre possède une réponse impulsionnelle qui ne se stabilisera jamais, et ce, même à l'infini. Ce type de filtre est récursif, c'est-à-dire que la sortie du filtre dépend à la fois du signal d'entrée et du signal de sortie, il possède ainsi une boucle de retour (contre-réaction). Les filtres IIR sont principalement la version numérique des filtres analogiques traditionnels : Butterworth, Tchebychev, Bessel, Elliptique. Nous représentons les trois formes les plus utilisées du filtre IIR en figure D.1 (Forme Directe 1), figure D.2 (Forme Directe 2) et figure D.3 (Forme Directe Transposée). Il est à noter que le forme directe 1 en dédoublant le nombre de registre sépare la partie convolutionnelle et la contre réaction. Le schéma N(Z) (figure D.1) correspond donc à celui d'un filtre FIR.

D.2 Registres à Décalages et LFSR

Les registres à décalages sont des architecture simples comprenant une succession de registres de taille fixe dans lesquels les bits présents initialement dans chaque registre sont décalés dans le registre suivant à chaque coup d'horloge (dans le cas d'un système synchrone sur l'horloge) [Koe90].

Un registre à décalage (Figure D.4) est en en général constitué d'un chaînage de bascules synchronisées sur l'horloge, la sortie d'une bascule étant reliée à l'entrée de la suivante. Un registre à décalage peut présenter deux types d'entrées et de sorties ; (1) séries (x(n) et y(n)) et (2) parallèles (Ei(n) et Si(n)) qui spécifient quatre types de registres.

- SIPO (Serial In - Parallel Out)
- SISO (Serial In - Serial Out)
- PISO (Parallel In - Serial Out)
- PIPO (Parallel In - Parallel Out)

Les entrées ou sorties en parallèle permettent d'insérer et de récupérer plusieurs bits en même temps. Dans le cas du type SISO, l'information est présentée à l'entrée de la première bascule. Lors d'une impulsion d'horloge, le bit d'information est introduit dans le registre, et tous les autres bits sont décalés. Dans le cas du type PIPO, en décalant tous les bits d'un nombre binaire vers la droite ou vers la gauche, on divise

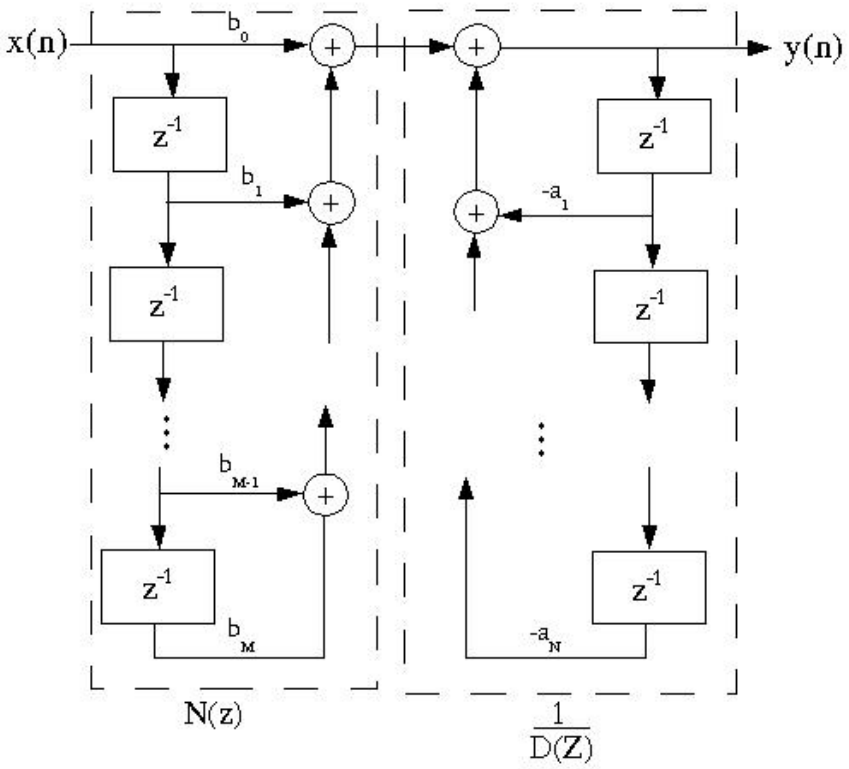

FIGURE D.1 – Première Forme Directe du Filtre IIR

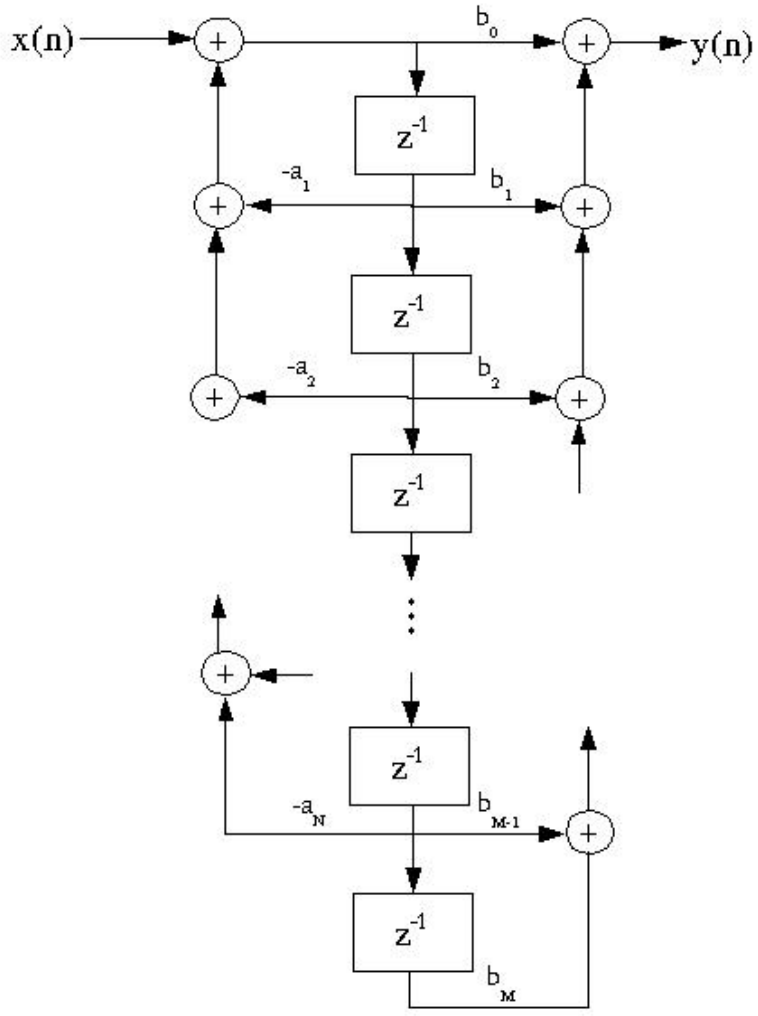

FIGURE D.2 – Deuxième Forme Directe du Filtre IIR

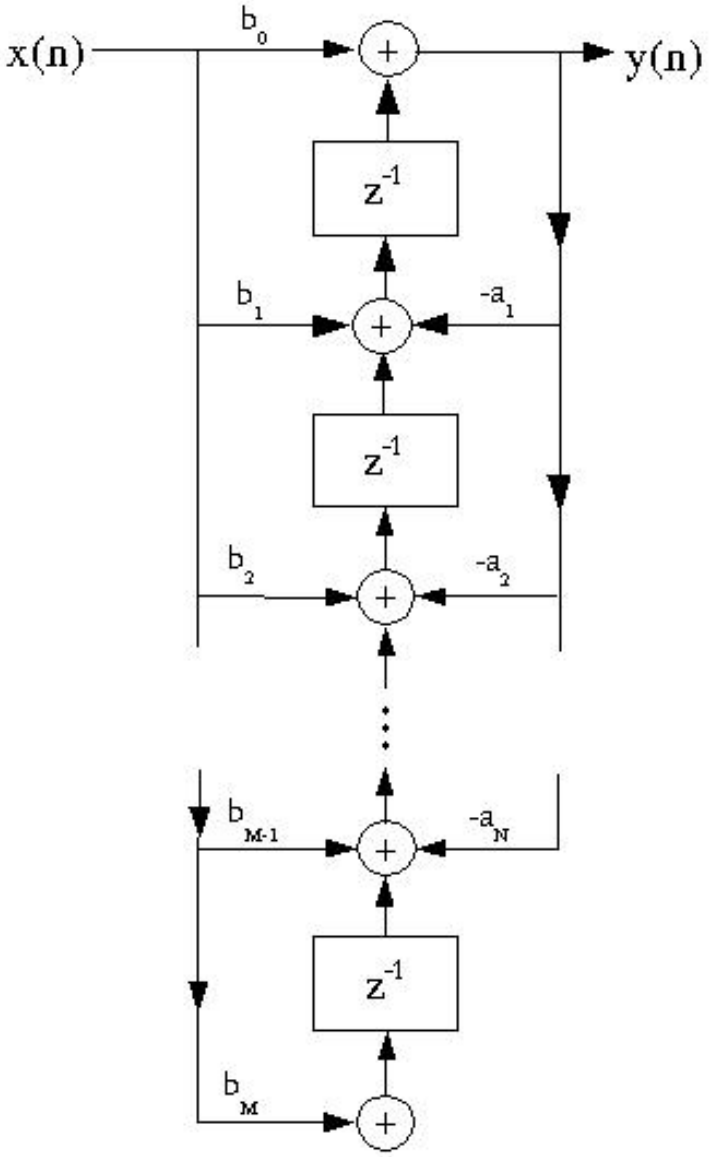

FIGURE D.3 – Forme Transposée du Filtre IIR

ou on multiplie le nombre par 2. Un registre PIPO peut donc être utilisé pour effectuer des calculs (multiplication ou division par une puissance de 2). Les deux autres types (PISO et SIPO) sont utilisés dans les liaisons séries ; ils forment la base des UART et des modems [Sem95].

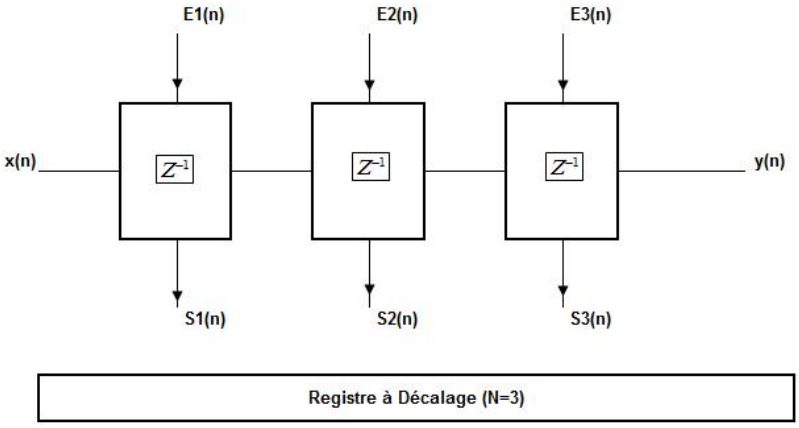

FIGURE D.4 – Architecture Registre Decalage

Parmi différents types de registres à décalages, les registres à rétroactions Linéaires (LFSR) sont une variante avec une unité logique ou arithmétique. Le ou les bit(s) en sortie du registre subissent une série d'opérations et de transformations avant d'être réinsérés dans le registre. Ce type de registre est utilisé en cryptographie pour les implémentations matérielles de certains algorithmes de chiffrement de flot.

D.3 Famille de LFSR

Les opérateurs communs que nous proposons dans ces travaux de thèses sont tous les cinq basés sur des registres à décalages à rétroactions linéaires. Les registres à décalage sont très utilisés dans les normes de télécommunications et se divisent en deux grandes familles : (1) type Galois (Type G) et (2) type de Fibonacci (Type F) [GK02]. Concrètement, les deux types de LFSR se distinguent par rapport à la boucle de rétroaction linéaire et aux unités de calculs qui y sont liées. Ces deux visions spécifiques des LFSRs trouvent leurs explications, dans les fonctions réalisées par chacun des deux types.

D.3.1 Le type de Fibonacci

Les LFSRs-Fibonacci prennent leurs noms car ils se basent sur une récurrence généralisée calquée sur la suite de Fibonacci. En effet, la dénomination de " suite de Fibonacci " est attribuée aux fonctions G définie sur l'ensemble des entiers naturels

vérifiant pour tout n, G(n+2) = G(n+1) + G(n). Ces fonctions sont celles pour lesquelles il existe des nombres a et b, tels que pour tout entier naturel n, G(n) = a.F(n) + b.F(n+1).

Si nous considérons un LFSR de Fibonacci avec S_i le vecteur à l'itération i de l'état de chaque registre $s_{j,i}$ et a_j les coefficients multiplicateurs des boucles de retour la valeur de sortie de l'unité de calcul qui correspond également à la valeur du premier registre est donnée par l'équation D.3 où N est la taille du registre. La figure D.5 illustre cette architecture. Ici, l'unité de calcul est un additionneur modulo 2.

$$\begin{aligned} s_{1,i} &= \sum_{k=1}^{N} a_k \cdot s_{j,i-1} \quad \text{(D.3)} \\ s_{j,i} &= s_{j-1,i-1} \end{aligned}$$

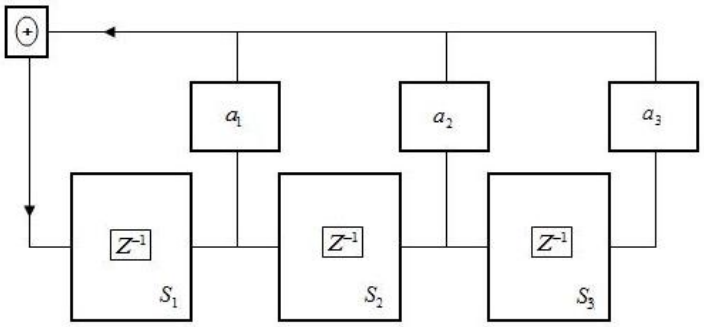

FIGURE D.5 – Structure Générique du LFSR de Fibonacci

D.3.2 Le type de Galois

Le type de Galois : Définition Le LFSR de Galois doit son nom à sa capacité à générer et à travailler sur des corps de Galois. D'un point de vue architectural, le type de Galois se distingue du type Fibonacci par rapport aux valeurs de retour et à leurs utilisations vis à vis des registres. La valeur de retour du LFSR de Galois est directement prise à la sortie du registre à décalage. Elle correspond à la sortie série décrite en D.2. Elle est directement retournée à chaque registre, chacun d'eux possédant une unité de calcul identique permettant de réaliser l'addition modulo 2 de la valeur du registre précédent et de cette valeur de retour pondérée par un coefficient binaire. Ainsi, un type de Galois définit le contenu de chaque cellule par rapport au registre précédent et à la valeur de retour. En utilisant les notations similaires au modèle de Fibonacci, nous donnons l'équation D.4 des $s_{j,i}$ illustrée par la figure D.6.

$$s_{j,i} = s_{j-1,i-1} + a_{j-1}.s_{N,i-1} \tag{D.4}$$

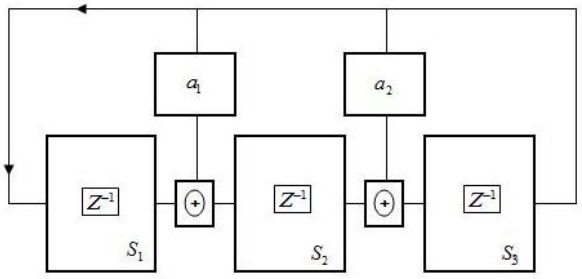

FIGURE D.6 – Structure Générique du LFSR de Galois

Le type de Galois : Utilisation Les " corps de Galois " font partie d'une branche particulière des mathématiques qui modélise les fonctions numériques. Ils sont très utilisés dans la cryptographie ainsi que pour la reconstruction des données. Il y a deux types de corps ; (1) les corps finis et (2) les corps infinis. Les " corps de Galois " finis sont des ensembles d'éléments fermés sur eux-mêmes. L'addition et la multiplication de deux éléments du champ donnent toujours un élément du champ fini. Un " corps de Galois " consiste en un ensemble de nombres, constitués à l'aide de l'élément de base α comme suit : (0,1,$\alpha, \alpha^2, ..., \alpha^{(2N-1)}$). En prenant $2N = m$, on forme un ensemble de 2^m éléments. Le champ est alors noté GF (2^m). GF (2^m) est formé à partir du corps de base GF (2) et contiendra des multiples des éléments simples de GF (2). En additionnant les puissances de α, chaque élément du corps peut être représenté par une expression polynomiale du type : $\alpha^{(m-1)}.x^{(m-1)} + ... + \alpha.x + \alpha^0$, avec : $\alpha^{m-1}, ..., \alpha^0$, éléments de bases de GF (2). Sur les " corps de Galois ", nous pouvons effectuer toutes les opérations de base. L'addition dans un corps fini GF (2) correspond à faire une addition modulo 2, donc l'addition de tous les éléments d'un " corps de Galois " dérivés du champ de base sera une addition modulo 2 (XOR). La soustraction effectuera la même opération qu'une addition, c'est-à dire, la fonction logique " XOR ".La multiplication et la division seront des opérations modulo " grandeur du champ ", donc mod(2^{m-1}). Le LFSR de type Galois permet de générer tous les éléments du corps de GF(2^8). Il suffit pour cela de charger, au début, son registre avec (1,0,...0) et de faire correspondre les boucles de retour du LFSR aux polynôme générateur du corps en question.
Lors de la génération d'une impulsion sur l'entrée horloge du registre, deux cas de figure peuvent se produire :

1. Soit la valeur du registre S_0 vaut 0. Dans ce cas, la prochaine valeur du corps de

galois correspondra simplement au décalage vers la droite de l'élément représenté dans les registres.

2. Soit la valeur du registre S_0 vaut 1. Dans ce cas, la prochaine valeur du corps de galois "dépassera" la valeur maximale de l'élément de base α^{m-1}. La valeur α^m sera donc répercuté sur les valeurs [1,...,α^{m-1}]. Cette répercussion est traduite par la boucle de retour et l'addition de α^m aux valeurs du registre à décalage.

.

Ainsi, a la ième impulsion, le registre contiendra la représentation binaire de l'élément α^{i+1}. Nous pouvons prendre comme exemple la génération de GF(8). Le polynôme primitif générateur du GF(2^8) est $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. Comme α est racine de p(x), alors on peut écrire $\alpha^8 = \alpha^4 + \alpha^3 + \alpha^2 + 1$. Le circuit apte à réaliser cette opération est décrit en figure D.7. Nous donnons en figure D.8, les premières itérations de cette structure.

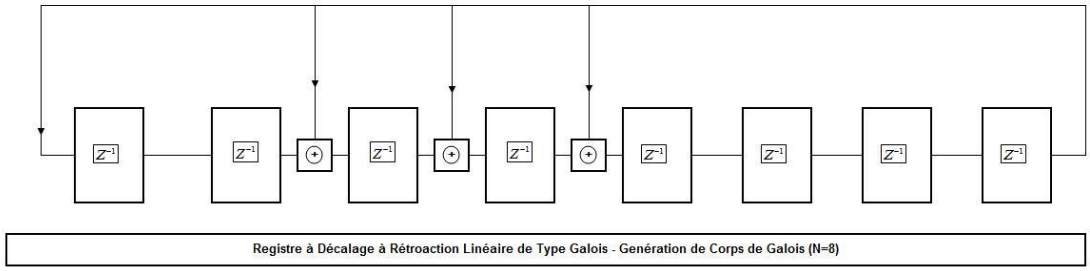

FIGURE D.7 – Structure LFSR de Galois adapté à générer les éléments de $GF(2^8)$

α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^1
1	0	0	0	0	0	0	0	α^0
...	...	...	...	...	...	...	...	...
0	0	0	0	0	0	0	1	α^7
1	0	1	1	1	0	0	0	$\alpha^8 = \alpha^0 + \alpha^2 + \alpha^3 + \alpha^4$
0	1	0	1	1	1	1	0	$\alpha^9 = \alpha^1 + \alpha^3 + \alpha^4 + \alpha^5$
0	0	1	0	1	1	1	0	$\alpha^{10} = \alpha^2 + \alpha^4 + \alpha^5 + \alpha^6$
0	0	0	1	0	1	1	1	$\alpha^{11} = \alpha^3 + \alpha^5 + \alpha^6 + \alpha^7$
1	0	1	1	0	0	1	1	$\alpha^{12} = \alpha^0 + \alpha^2 + \alpha^3 + \alpha^6 + \alpha^7$
1	1	1	0	0	0	0	1	$\alpha^{13} = \alpha^0 + \alpha^1 + \alpha^2 + \alpha^7$

FIGURE D.8 – Génération des éléments de $GF(2^8)$

Annexe E

Réalisation des Opérations par les différents niveaux de granularité des OC LFSR

Sommaire

E.1 Calcul relatif au RF-LFSR : Puissance de G_F

Dans la suite de cette section, nous utiliserons des puissances de G_F pour expliquer le passage d'une architecture à l'autre, nous explicitons donc ici la forme que prend G_F^p.

$$G_F = \begin{pmatrix} g_{F(1,1)} & \cdots & g_{F(1,k)} & \cdots & g_{F(1,N)} \\ 1 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & 0 \\ 0 & 0 & ... & ... & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} g_{F(1,1)} & \cdots & g_{F(1,k)} & \cdots & g_{F(1,N)} \\ | & - & - & - & - \\ | & & T = Matrice & & | \\ | & & deToeplitz & & | \\ | & - & - & - & | \end{pmatrix}$$

G_F est la forme matricielle qui associe la valeur du registres S_i à S_{i-1}, ainsi nous pouvons définir S_i par récurrence par rapport à S_0 en fonction de puissances de G_F. G_F présente une forme particulière, elle est constitué d'une première matrice ligne de taille 1

x N, caractéristique du polynôme générateur de la suite de Fibonacci g_{F1} et d'une sous-matrice T de Toeplitz de taille (N-1) x N. Cette matrice en question est caractéristique d'un registre de décalage unitaire. En multipliant une matrice A quelconque par une matrice T de Toeplitz de décalage de rang 1 de même dimension, on obtiendra la matrice des valeurs décalées. Nous pouvons prendre un exemple simple de dimension 3 x 3. Si nous posons,

$$A_1 = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}, A_2 = \begin{pmatrix} g & h & i \end{pmatrix}, et\ T = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Alors,

$$T.A = T.\begin{pmatrix} A_1 \\ \hline A_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ a & b & c \\ d & e & f \end{pmatrix} = \begin{pmatrix} 0 \\ \hline A_1 \end{pmatrix}$$

En généralisant

$$G_F.G_F = \begin{pmatrix} g_{F1} \\ \hline T \end{pmatrix} \cdot \begin{pmatrix} & g_{F1} & \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} & (g_{F1})^2 & \\ & g_{F1} & \\ 1 & 0 & 0 \end{pmatrix}, avec (g_{F1})^2 = (g_{F1}).G_F$$

$$G_F.(G_F)^2 = \begin{pmatrix} g_{F1} \\ \hline T \end{pmatrix} \cdot \begin{pmatrix} & (g_{F1})^2 & \\ & g_{F1} & \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} (g_{F1})^3 \\ (g_{F1})^2 \\ \hline g_{F1} \end{pmatrix}$$

Ainsi, comme G_F possède une sous matrice de Toeplitz de décalage de rang 1 alors la multiplication de G_F par lui même engendrera une matrice de dimension N x N, où la sous matrice de G_F^2 de dimension (N-1) x N définie à partir de de la ligne j=2, sera équivalente à la sous matrice de G_F de même dimension mais commençant à j=1. La sous matrice de G_F^2 est la décalée de G_F.
Ainsi, pour chaque G_F^p, nous pouvons construire la matrice à partir des (N-1) lignes décalées (de j = 1 à N-1) de la matrice G_F^{p-1} et nous limiter à calculer seulement g_{F1}^p tel que :

$$g_{F1}^p = g_{F1}^{(p-1)}.G_F. \tag{E.1}$$

Nous avons donc :

$$G_F^p = \begin{pmatrix} g_{F1}^p \\ \dots \\ g_{F1}^{p-j} \\ \dots \\ g_{F1}^{p-N+1} \end{pmatrix}$$

E.2 Calcul relatif au RG-LFSR : Réalisation du RF-LFSR par le RG-LFSR

E.2.1 Relation entre les équations du RG-LFSR et du RF-LFSR.

Nous différencions par a_{Fk} et a_{Gk} les coefficients du RF-LFSR et du RG-LFSR, alors pour une même entrée x, la même sortie y sera obtenue si et seulement si $a_{Fk} = a_{G(N-k)}$. Ce résultat se démontre trivialement par récurrence :

$$y_{F(i)} = \sum_{k=1}^{N} a_{F(k)} \cdot y_{F(i-k)} + x_i \tag{E.2}$$

$$y_{G(i)} = \sum_{k=1}^{N} a_{G(N-k)} \cdot y_{G(i-k)} + x_i = \sum_{k=1}^{N} a_{F(N-k)} \cdot y_{G(i-k)} + x_i \tag{E.3}$$

Pour i=0, en considérant les valeurs initiales des registres nulles, alors $y_{F(0)} = x_0 = y_{G(0)}$, par récurrence, nous montrons trivialement que :

$$y_{F(j)} = y_{G(j)} \tag{E.4}$$

Ainsi, par "inversion" des coefficients entre RF-LFSR et RG-LFSR, pour une même entrée x, nous obtiendrons une même sortie y quelque soit le LFSR utilisé. Ce passage d'une structure à l'autre nous permet d'avancer le RG-LFSR comme capable de se substituer à un générateur de séquences auto-synchronisé tel que défini en 3.2.1, ces dernieres ne dépendant que du message recu x et non de la séquence initiale.

E.2.2 Séquences initiales du RG-LFSR et du RF-LFSR.

L'égalité précédente entre les sorties du RF-LFSR et du RG-LFSR n'est valable que lorsque nous considérons les séquences initiales nulles. Nous devons prouver l'équivalence entre RF-LFSR et RG-LFSR à partir d'une séquence initiale spécifique et déterminer une méthode de passage de l'un à l'autre.

Pour les deux opérateurs, pour tout x_i alors nous avons $y_{F(i)} = y_{G(i)}$. Or, une séquence de N entrées x_i produira non seulement N sorties $y_{F(i)}$ et $y_{G(i)}$ équivalentes mais aussi N valeurs des registres de Galois $s_{G(j,i)}$ et de Fibonacci $s_{F(j,i)}$. Ainsi, réciproquement, pour chaque $s_{G(j,i)}$ et $s_{F(j,i)}$, il existe une séquence d'entrée x_i telle que $y_{F(i)} = y_{G(i)}$. Donc, à partir de $S_{F(0)}$ =[0], pour une séquence $S_{F(i)}$, il existe à la Nième itération une séquence de N entrées x_i tel que $y_{F(i)} = y_{G(i)}$. Donc à la Nième itération, pour la même séquence de N x_i, il existera $S_{G(i)}$ tel que $y_{F(j)} = y_{G(j)}$. Ainsi, pour un $S_{F(i)}$ donné, il existe $S_{G(i)}$, tel que $y_{F(i)} = y_{G(i)}$. En considérant la séquence initiale de chaque registre comme la fonction de N entrées x_i, nous prouvons qu'à partir d'une séquence donnée de Fibonacci, il existe une séquence de Galois qui produit les mêmes résultats.

A partir d'une séquence initiale de Fibonacci, nous devons donc déterminer les N valeurs initiales de la séquence de Galois. Comme nous avons N inconnues, nous devons

établir un système à N équations, c'est à dire les N premières égalités entre $y_{F(i)} = y_{G(i)}$. Comme, nous ne travaillons qu'à partir des séquences initiales, les différents X_i sont pris égaux à zero. Poser l'égalité des N premiers $y_{F(i)}$ et $y_{G(i)}$ revient à poser l'égalité des N premiers $s_{F(1,i)}$ et $s_{G(N,i-1)}$. Comme nous avons les relations :

$$S_i = G_F^i.S_O$$
$$S_i = G_G^i.S_O$$

Alors :

$$s_{F(1,i)} = g_{F1}^i.S_{F0}$$
$$s_{G(N,i-1)} = g_{GN}^i.S_{G0}$$

Ce qui nous donne un système à N équations et N inconnues :

$$\begin{pmatrix} g_{F1} \\ ... \\ g_{F1}^j \\ ... \\ g_{F1}^N \end{pmatrix}.S_{F0} = \begin{pmatrix} 1 \\ ... \\ g_{GN}^j \\ ... \\ g_{GN}^{N-1} \end{pmatrix}.S_{G0}$$

Avec,

$$G_{F1} = \begin{pmatrix} g_{F1} \\ ... \\ g_{F1}^j \\ ... \\ g_{F1}^N \end{pmatrix}, G_{G(N-1} = \begin{pmatrix} 1 \\ ... \\ g_{GN}^j \\ ... \\ g_{GN}^{N-1} \end{pmatrix}$$

Nous avons donc l'égalisté des N premières sorties des deux architectures :

$$G_{F1} \cdot S_{F0} = G_{G(N-1)} \cdot S_{G0} \tag{E.5}$$

Dans la mesure, où nous connaissons les séquences de Fibonacci Initiales, les séquences de Galois équivalentes seront données par :

$$S_{G0} = G_{G(N-1)}^{-1} \cdot G_{F1} \cdot S_{F0} \tag{E.6}$$

Nous illustrons notre propos (Figure E.1) en considérant le RF-LFSR de polynôme générateur $x^3 + x^2 + 1$ et de séquence initiale $[1, 0, 1]^T$. Le paramétrage équivalent du RG-LFSR sera d'après les explications précédentes, généré par le polynôme $x^3 + x + 1$. En considérant la séquence initiale du RG-LFSR ($[A, B, C]^T$), nous avons besoins d'un système à 3 inconnues. Nous définissons les trois premières puissances de G_F afin de construire G_{F1}. Dans le même temps, nous calculons les deux premières puissances de G_G et définissons $G_{G(N-1)}$ et $(G_{G(N-1)})^{N-1}$ pour finalement déterminer la séquence A,B,C. Nous implémentons une telle séquence initiale et vérifions sur Matlab l'égalité des résultats, que nous retranscrivons en Figure E.2. En prenant, N=3 comme ordre du polynôme générateur, nous n'avons qu'à vérifier l'égalité des $2^N - 1 = 7$ premiers résultats, première période de la séquence générée.

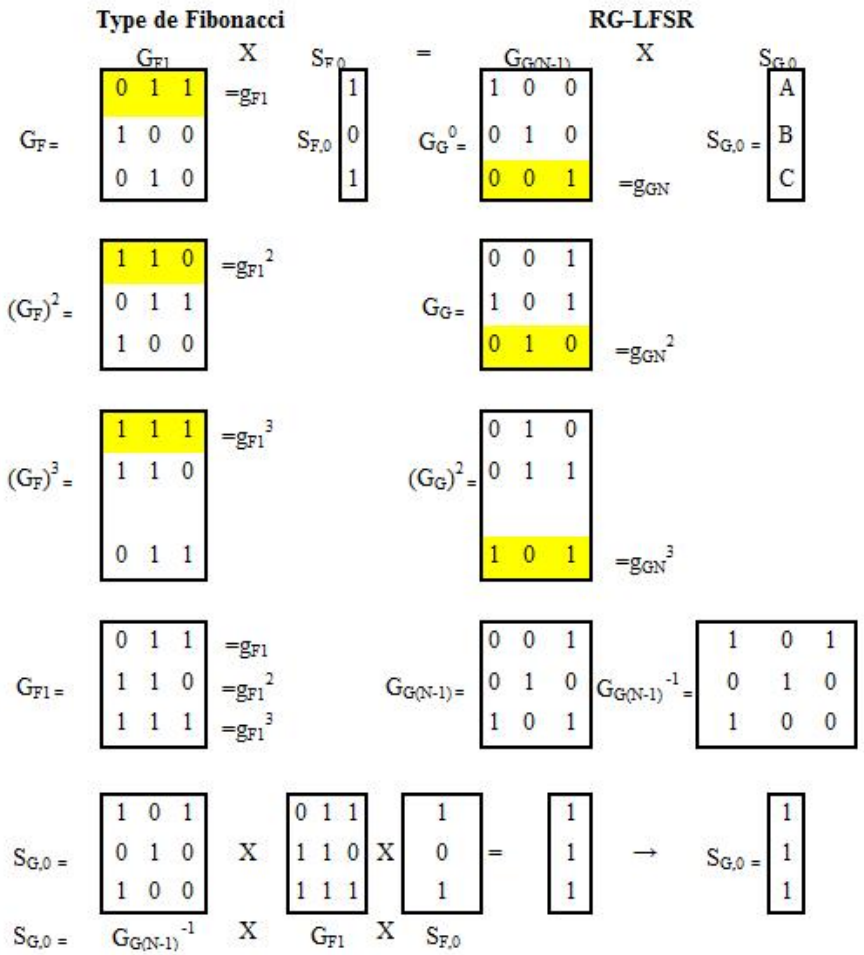

FIGURE E.1 – Exemple de la génération de la même séquence par le RF-LFSR et le RG-LFSR

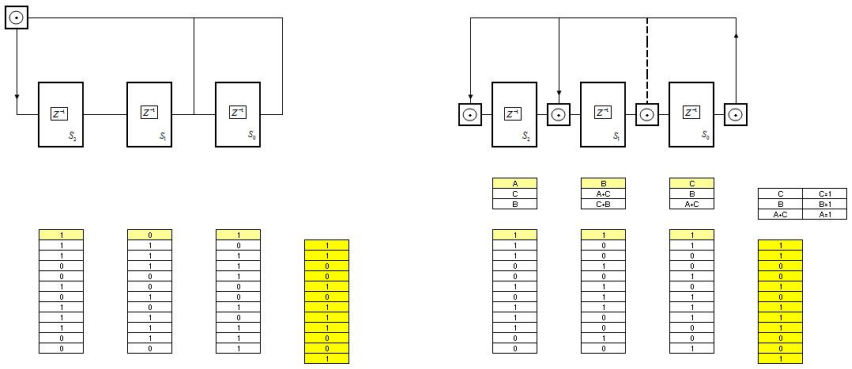

FIGURE E.2 – Exemple de la génération de la même séquence par le RF-LFSR et le RG-LFSR : Architecture et Premières Itérations

En conclusion, le RG-LFSR peut être paramétré pour le calcul des divisions nécessaires au CRC et pour la génération de séquences aléatoire équivalentes à celles d'un RG-LFSR, c'est à dire à celles d'un LFSR ou d'un LFG.

E.3 Calcul relatif au R-LFSR : Séquences initiales du R-LFSR et du IIR

Dans le chapitre E.2.2, nous définissions une méthode de détermination de la séquence initiale du RG-LFSR à partir de celle du RF-LFSR afin d'obtenir la même séquence pseudo aléatoire. Nous présentons en 3.2.1, les différentes méthode pour obtenir ces séquences aléatoires, codeurs congruentiels, LFG ou LFSR. Néanmoins, certains standards définissent des générateurs de séquences aléatoires plus complexes et qui ne peuvent être réalisées par ces architectures. Dans notre cas d'étude, le générateur qui arbore un architecture particulière est celui du 3GPP LTE, présent avec des polynômes générateurs différents pour la voix montante et descendante. Celui-ci (figure 3.5) génère en simultané deux séquences de Gold. Les séquences de Gold sont par définition construites à partir de deux m-séquences, chacune de ces m-séquences pouvant être définies par un LFSR.

Dans le cas présent, nous avons bien deux registres de forme LFSR qui génèrent deux m-séquences, qui sont combinées pour former la première séquence de Gold. Ce schéma classique de génération de m-séquences peut être aisément produit par un LFSR de Fibonacci sous condition d'additionner (modulo 2) de manière synchrone ces séquences pour former la séquence de Gold.
Cependant, la problématique intervient quant à la génération de la deuxième m-séquence. Celle-ci est construite à partir d'une combinaison linéaire des premières m-séquences. Concrètement, une fois générée par le LFSR de Fibonacci, chacune des valeurs des m-séquences est réintroduite dans le registre à décalage. C'est à partir des valeurs de ces

m-séquences présentes dans le registre à décalage que la deuxième séquence de Gold est obtenue.

D'un point de vue architectural, chacune de cette deuxième série de m séquences et une combinaison linéaire des valeurs du registre d'un LFSR de Fibonacci, c'est à dire équivalentes aux sorties d'un filtre IIR. Nous avons montré dans ce rapport que le R-LFSR fournit la même sortie y qu'un filtre IIR pour un même entrée x. Dans le cas des générateurs de séquences aléatoires, nous n'avons pas de valeur d'entrée x, celle-ci est nulle et la séquence aléatoire ne se base que sur les valeurs initiales. Nous devons par conséquent, déterminer la possibilité de créer la même séquence à partir d'un R-LFSR que d'un IIR et le cas échéant nous devons proposer l'équivalence entre séquences initiales d'un filtre IIR et du R-LFSR.
En suivant le raisonnement équivalent proposé en 4.3.2 quand à l'équivalence entre RF-LFSR et RG-LFSR relatif à l'égalité des séquences produites, nous pouvons avancer la possibilité de créer la même séquence aléatoire entre filtre IIR et R-LFSR. Nous devons déterminer une méthode de passage de l'un à l'autre.

Lors d'une génération de séquences aléatoires dans le cas d'un R-LFSR basé uniquement sur la séquence initiale, alors $x_i = 0$. Par conséquent, la génération de séquences aléatoires devient équivalente à celle d'un RG-LFSR. Cependant, si le R-LFSR (Filtre IIR en forme transposée) équivaut au RG-LFSR (LFSR de Galois), le filtre IIR générateur de séquences aléatoires, n'est pas assimilable au LFSR de Fibonacci. En effet, en gardant les mêmes notations que dans la démonstration de 4.3.2, la valeur de sortie du LFSR de Fibonacci est égale à $s_{F(1,i)}$ alors que la valeur de sortie y_i du filtre IIR est donnée par :

$$y_i = s_{F(1,i)} + R \cdot S_{F(i-1)} \tag{E.7}$$

R représente le vecteur associé au polynôme de la boucle de sortie de y_i. Ainsi, nous ne pouvons pas nous ramener au cas de 4.3.2. En suivant néanmoins, le même raisonnement :

$$y_i = g_{F1} \cdot S_{F(i-1)} + R \cdot S_{F(i-1)} = [g_{F1} + R] \cdot S_{F(i-1)} \tag{E.8}$$

Ainsi, par récurrence :

$$y_i = [g_{F1} + R] \cdot (G_F)^{i-1} \cdot S_{F(0)} \tag{E.9}$$

A partir d'une séquence initiale de Fibonacci, nous devons donc déterminer les N valeurs initiales de la séquence de Galois. Comme nous avons N inconnues, nous devons établir un système à N équations, c'est à dire les N premières égalités entre $y_i = y_{G(i)}$. Poser l'égalité des N premiers y_i et $y_{G(i)}$ revient à poser l'égalité des N premiers y_i et $s_{G(N,i-1)}$. Nous avons les relations :

$$y_i = [g_{F1} + R] \cdot (G_F)^i \cdot S_{F(0)}$$
$$y_{G(i)} = s_{G(N,i-1)} = g_{GN}^{i-1} . S_{G0}$$

Ce qui nous donne un système à N équations et N inconnues :

$$\begin{pmatrix} (g_{F1} + R) \cdot (G_F)^0 \\ ... \\ (g_{F1} + R) \cdot (G_F)^j \\ ... \\ (g_{F1} + R) \cdot (G_F)^{N-1} \end{pmatrix} . S_{F0} = \begin{pmatrix} 1 \\ ... \\ g_{GN}^j \\ ... \\ g_{GN}^{N-1} \end{pmatrix} \cdot S_{G0}$$

Nous avons donc l'égalisté des N premières sorties des deux architectures :

$$G_{IIR} \cdot S_{F0} = G_{G(N-1)} \cdot S_{G0} \tag{E.10}$$

$$G_{IIR} = \begin{pmatrix} (g_{F1} + R) \cdot (G_F)^0 \\ ... \\ (g_{F1} + R) \cdot (G_F)^j \\ ... \\ (g_{F1} + R) \cdot (G_F)^{N-1} \end{pmatrix}$$

Dans la mesure, où nous connaissons les séquences de Fibonacci Initiales, les séquences de Galois équivalentes seront données par :

$$S_{G0} = G_{G(N-1)}^1 \cdot G_{IIR} \cdot S_{F0} \tag{E.11}$$

Nous illustrons notre propos (Figure E.3) en considérant le système IIR de polynôme de rétroaction $x^3 + x^2 + 1$ et de polynôme de diffusion $x^3 + x + 1$ et de séquence initiale $[1, 0, 1]^T$. Le paramétrage équivalent du R-LFSR sera d'après les explications précédentes, généré respectivement par les polynômes $x^3 + x + 1$ et $x^3 + x^2 + 1$.
En considérant la séquence initiale du R-LFSR ($[A, B, C]^T$), nous avons besoins d'un système à 3 inconnues. Nous définissons les deux premières puissances de G_F afin de construire G_{IIR}. Dans le même temps, nous calculons les deux premières puissances de G_G et définissons $G_{G(N-1)}$ et $(G_{G(N-1)})^{-1}$ pour finalement déterminer la séquence A,B,C. Nous implémentons une telle séquence initiale et vérifions sur Matlab l'égalité des résultats, que nous retranscrivons en Figure E.4. En prenant, N=3 comme ordre du polynôme générateur, nous n'avons qu'à vérifier l'égalité des $2^N - 1 = 7$ premiers résultats, première période de la séquence générée.

E.4 Calcul relatif à l'ER-LFSR : Comparaison entre ER-LFSR et m-CRSC du Turbo Codage

Même si le ER-LFSR(m,n) proposé comme architecture commune peut être présenté comme une extension du R-LFSR, l'architecture ER-LFSR(m,n) est en premier lieu définie pour correspondre à la définition de Berrou du turbo code utilisant les m-CRSC

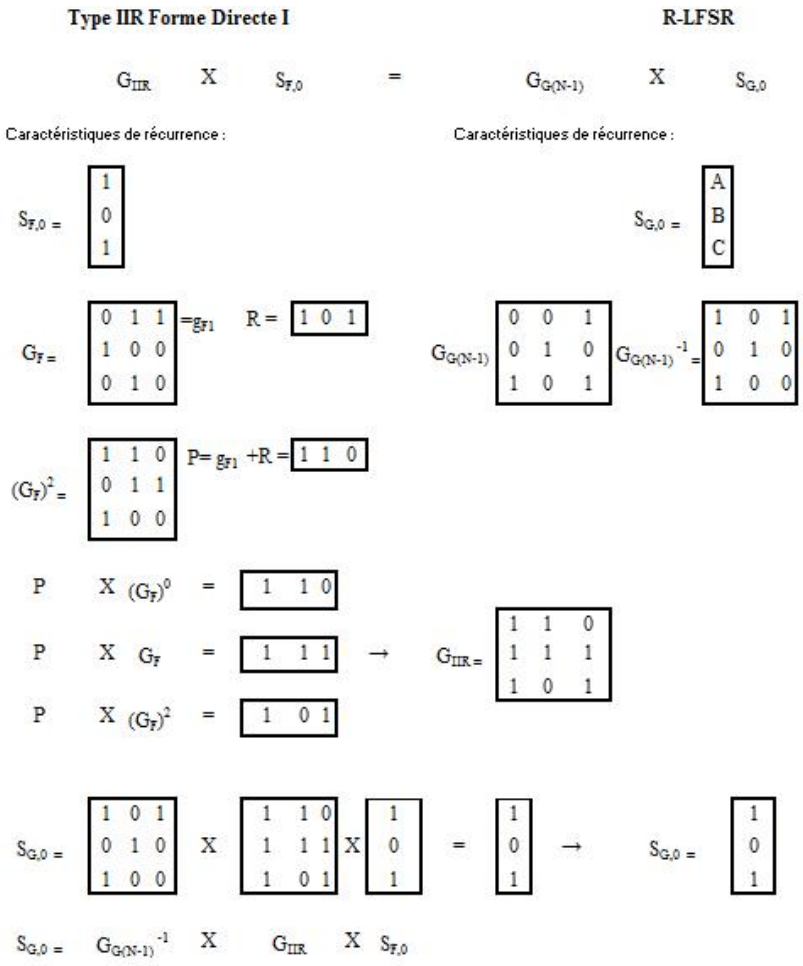

FIGURE E.3 – Exemple de la génération de la même séquence par un scrambler de forme IIR et le R-LFSR

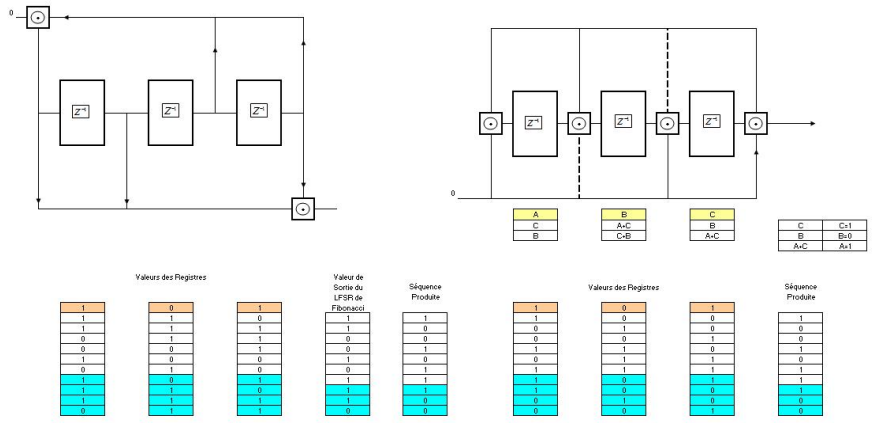

FIGURE E.4 – Exemple de la génération de la même séquence par un scrambler de forme IIR et le R-LFSR : Architecture et Premières Itérations

[BC99]. Ainsi, afin de pouvoir étendre la gamme de fonctions réalisables par le m-CRSC aux opérations exécutables par le R-LFSR (4.3.3), nous devons notamment être capable d'effectuer les opérations de divisions polynomiales dans GF(2) et la génération du corps de Galois. Nous avons expliqué en D.3 que parmi les LFSR de Galois et Fibonacci, seul le type de Galois pouvait mener à bien ces deux opérations. Ainsi, l'architecture générique en ER-LFSR(m,n) se distingue architecturalement du m-CRSC par :

- L'ajout d'une boucle de rétroaction de Type de Galois.
- L'ajout d'une N+1 entrée $x_{(N+1),i}$.

Ainsi, par rapport à la définition de [BC99], les équation du ER-LFSR(m,n) deviennent :

$$S_i = G \cdot S_{i-1} + T'_i + [s_{(N,i-1)} + x_{(N+1),i}] \cdot g_{GN} \tag{E.12}$$

Dans ce cas précis, les notations matricielles S_i, T'_i et G reprennent les notations de [BC99]. Ils représentent respectivement, les valeurs des registres $s_{(j,i)}$ à l'itération i, les N valeurs $t'_{j,i}$ des combinaisons des m entrées à l'itération i et le polynôme reliant les valeurs de S_i à S_{i-1}. Par rapport à ces notations initiales, nous définissons :

- La matrice g_{GN} de taille 1 x N, caractéristique des coefficients du polynôme générateurs de la boucle de rétroaction de type de Galois.
- L'entrée $x_{(N+1),i}$ est la N+1 ème entrée permettant de faire correspondre le ER-LFSR(m,n) au R-LFSR et plus précisemment au RG-LFSR, afin de répondre au besoins en calcul de CRC.

Ainsi, la définition du ER-LFSR(m,n) nécessite la définition d'une nouvelle matrice C' faisant passer la matrice C notée C_{Berrou} et initialement définie en [BC99] de taille N x m à (N+1) x m afin de tenir compte du N+1 ème $x_{j,i}$. Pour rappel, la matrice C est la matrice des combinaisons linéaires reliant chaque $x_{j,i}$ aux m entrées $d_{j,i}$.

L'entrée $x_{N+1,i}$ ne sera pas injecté directement dans le calcul des registres à l'itération i mais sera "additionnée" à la valeur du registre $s_{N,i-1}$ pour former la valeur de

retour du LFSR de Galois. Comme l'addition garde sa propriété d'associativité dans GF(2), alors chaque $x_{j,i}$ sera additionné à $g_{G(j,N)} \cdot x_{N+1,i}$:

$$t'_{j,i} = x_{j,i} + x_{N+1,i} \cdot g_{G(j,N)} \tag{E.13}$$

Comme $x_{j,i}$ et $x_{N+1,i}$ sont tous deux des combinaisons linéaires des $d_{j,i}$ alors nous pouvons définir une nouvelle matrice C' et obtenir la nouvelle égalité entre T_i et d_i :

$$T'_i = C' \cdot d_i \tag{E.14}$$

La nouvelle matrice C' ($[c'_{(j,p)}]$) est définie par rapport à la matrice C ($[c_{(j,p)}]$) de la grille de connections tel que :

$$c'_{(j,p)} = c_{(j,p)} + g_{G(j,N)} \cdot c_{(N+1,p)} \tag{E.15}$$

$$C' = \begin{pmatrix} \dots & \dots & \dots \\ \dots & c_{j,k} + g_{G(j,N)} \cdot c_{(N+1),k} & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

Ainsi, en utilisant ces paramètres, nous pouvons définir le ER-LFSR(m,n) par l'équation précédente :

$$S_i = G_{ER}.S_{i-1} + T_i \tag{E.16}$$

La linéarité de S_i par rapport à S_{i-1} permet d'intégrer nos travaux dans l'état de l'art défini par Berrou et en utilisant les travaux de [BC99], nous pouvons déterminer un état S_c du registre à décalage tel que S_c=S_0 :

$$S_c = \langle I + G_{ER} \rangle^{-1} . \sum_{p=1}^{k} G^{k-p}.X_p \tag{E.17}$$

Ainsi, le ER-LFSR(m,n) peut se définir comme un m-CRSC sous condition d'inversibilité de la matrice G_{ER}.

Annexe F

Les Standards Etudiés

Sommaire

F.1 Standard IEEE 802.11

La norme IEEE 802.11 (ISO/IEC 8802-11) est un standard international décrivant les caractéristiques d'un réseau local sans fil (WLAN). Le nom Wi-Fi correspond initialement au nom donnée à la certification délivrée par la Wi-Fi Alliance (autrefois WECA pour Wireless Ethernet Compatibility Alliance) qui est chargé de maintenir l'interopérabilité entre les matériels répondant à la norme 802.11. Par vulgarisation du terme, le nom de la norme est également associé aujourd'hui avec le nom de la certification. La norme 802.11 permet ainsi de créer des réseaux locaux sans fils à haut débit à condition que le terminal à connecter ne soit pas trop distant par rapport au point d'accès. En pratique, le WiFi permet de relier des ordinateurs portables, des ordinateurs de bureau, des assistants personnels (PDA) ou tout type de périphériques à une liaison haut débit (11 Mbps ou supérieur) sur un rayon de plusieurs dizaines de mètres en intérieur à plusieurs centaines de mètres en environnement ouvert.

La norme 802.11 définit les couches basses du modèle OSI pour une liaison sans fil utilisant des ondes électromagnétiques, c'est-à-dire :

- la couche PHY, proposant trois types de codages de l'information.
- la couche liaison de données, constituée de deux sous-couches : le contrôle de la liaison logique (Logical Link Control, ou LLC) et le contrôle d'accès au support (Media Access Control, ou MAC)

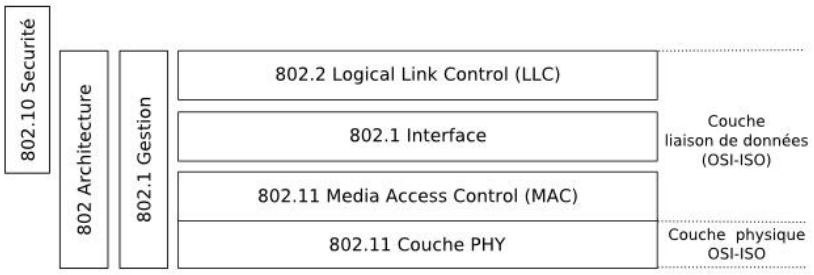

FIGURE F.1 – Modele OSI du WIFI

La couche physique définit la modulation des ondes radio-électriques et les caractéristiques de la signalisation pour la transmission de données, tandis que la couche liaison de données définit l'interface entre le bus de la machine et la couche physique, notamment une méthode d'accès proche de celle utilisée dans le standard ethernet et les règles de communication entre les différentes stations.

La norme IEEE 802.11 est en réalité la norme initiale offrant des débits de 1 ou 2 Mbps. Des révisions ont été apportées à la norme originale afin d'optimiser le débit (c'est le cas des normes 802.11a, 802.11b et 802.11g, appelées normes 802.11 physiques) ou bien préciser des éléments afin d'assurer une meilleure sécurité ou une meilleure interopérabilité.

Standard	Bandes de fréquences	Débit	Portée
WiFi a (802.11a)	5 GHz	54 $Mbit/s$	10 m
WiFi b (802.11b)	2.4 GHz	11 $Mbit/s$	100 m
WiFi g (802.11b)	2.4 GHz	54 $Mbit/s$	100 m
WiFi n (802.11n)	2.4 GHz *ou* 5 GHz	540 $Mbit/s$	200 m

TABLE F.1 – Différents Modes du WIFI

Standard IEEE 802.11a La norme 802.11a (Wi-Fi 5) permet d'obtenir un débit théorique de 54 Mbps, soit cinq fois plus que le 802.11b, pour une portée d'environ une trentaine de mètres seulement. La norme 802.11a s'appuie sur un codage du type Orthogonal Frequency Division Multiplexing (OFDM) sur la bande de fréquence 5 GHz et utilisent 8 canaux qui ne se recouvrent pas. La norme 802.11a spécifie 52 canaux de sous-porteuses radio dans la bande de fréquences des 5 GHz, huit combinaisons, non superposées sont utilisables pour le canal principal.

Ainsi, les équipements 802.11a ne sont donc pas compatibles avec les équipements 802.11b. Il existe toutefois des matériels intégrant des puces 802.11a et 802.11b, on parle alors de matériels "dual band".

Débit théorique (en intérieur)	Portée
54 $Mbits/s$	10 m
48 $Mbits/s$	17 m
36 $Mbits/s$	25 m
24 $Mbits/s$	30 m
12 $Mbits/s$	50 m
6 $Mbits/s$	70 m

TABLE F.2 – Spécificité du 802.11a

Standard IEEE 802.11b La norme 802.11b est la norme la plus répandue actuellement. Elle propose un débit théorique de 11 Mbit/s (6 Mbit/s réels) avec une portée pouvant aller jusqu'à 300 mètres (en théorie) dans un environnement dégagé. La plage de fréquences utilisée est la bande des 2,4 GHz avec, en France, 13 canaux radio disponibles dont 4 au maximum non superposés (1 - 5 - 9 - 13).

Débit théorique	Portée (en intérieur)	Portée (a l'extérieur)
11 $Mbits/s$	50 m	200 m
5,5 $Mbits/s$	75 m	300 m
2 $Mbits/s$	100 m	400 m
1 $Mbit/s$	150 m	500 m

TABLE F.3 – Spécificité du 802.11b

Standard IEEE 802.11g La norme 802.11g permet d'obtenir un débit théorique de 54 Mbps pour des portées équivalentes à celles de la norme 802.11b. D'autre part, dans la mesure où la norme 802.11g utilise la bande de fréquence 2,4GHZ avec un codage OFDM, cette norme est compatible avec les matériels 802.11b, à l'exception de certains anciens matériels. La norme 802.11g a une compatibilité ascendante avec la norme 802.11b, ce qui signifie que des matériels conformes à la norme 802.11g peuvent fonctionner en 802.11b. Cette aptitude permet aux nouveaux équipements de proposer le 802.11g tout en restant compatibles avec les réseaux existants qui sont souvent encore en 802.11b. Il est possible d'utiliser, au maximum, 4 canaux non superposés (1 - 5 - 9 - 13).

Standard IEEE 802.11n La norme 802.11n est disponible depuis le 11 septembre 2009. Le débit théorique atteint les 300 Mbit/s (débit réel de 100 Mbit/s dans un rayon de 100 mètres) grâce aux technologies MIMO (Multiple-Input Multiple-Output) et OFDM (Orthogonal Frequency Division Multiplexing).

Le 802.11n a été conçu pour pouvoir utiliser les fréquences 2,4 GHz ou 5 GHz. Les premiers adaptateurs 802.11n actuellement disponibles sont généralement simple-bande à 2,4 GHz, mais des adaptateurs double-bande (2,4 GHz ou 5 GHz, au choix) ou même double-radio (2,4 GHz et 5 GHz simultanément) sont également disponibles. Le

Débit théorique	Portée (en intérieur)	Portée (à l'extérieur)
54 $Mbits/s$	27 m	75 m
48 $Mbits/s$	29 m	100 m
36 $Mbits/s$	30 m	120 m
24 $Mbit/s$	42 m	140 m
18 $Mbit/s$	55 m	180 m
12 $Mbit/s$	64 m	250 m
9 $Mbit/s$	75 m	350 m
6 $Mbit/s$	90 m	400 m

TABLE F.4 – Spécificité du 802.11g

802.11n pourra combiner jusqu'à 8 canaux non superposés, ce qui permettra en théorie d'atteindre une capacité totale effective de presque un gigabit par seconde.

La couche physique de la norme 802.11 définit ainsi initialement plusieurs techniques de transmission permettant de limiter les problèmes dûs aux interférences.

- La technique de l'étalement de spectre à saut de fréquence
- La technique de l'étalement de spectre à séquence directe
- La technologie infrarouge

Ainsi, la norme IEEE 802.11 propose deux techniques de modulation de fréquence pour la transmission de données issues des technologies militaires. Ces techniques, appelées étalement de spectre (spread spectrum - SS) consistent à utiliser une bande de fréquence large pour transmettre des données à faible puissance.

Etalement de spectre à saut de fréquence La technique FHSS (Frequency Hopping Spread Spectrum - étalement de spectre par saut de fréquence) consiste à découper la large bande de fréquence en un minimum de 75 canaux d'une largeur de 1MHz, puis de transmettre en utilisant une combinaison de canaux connue de toutes les stations de la cellule. Dans la norme 802.11, la bande de fréquence 2.4 - 2.4835 GHz permet de créer 79 canaux de 1 MHz. La transmission se fait ainsi en émettant successivement sur un canal puis sur un autre pendant une courte période de temps (d'environ 400 ms), ce qui permet à un instant donné de transmettre un signal plus facilement reconnaissable sur une fréquence donnée.

L'étalement de spectre par saut de fréquence a originalement été conçu dans un but militaire afin d'empêcher l'écoute des transmissions radio. En effet, une station ne connaissant pas la combinaison de fréquences à utiliser ne pouvait pas écouter la communication car il lui était impossible dans le temps imparti de localiser la fréquence sur laquelle le signal était émis puis de chercher la nouvelle fréquence.

Aujourd'hui les réseaux locaux utilisant cette technologie sont standards ce qui signifie que la séquence de fréquences utilisées est connue de tous, l'étalement de spectre par saut de fréquence n'assure donc plus cette fonction de sécurisation des échanges.

En contrepartie, la FHSS est désormais utilisé dans le standard 802.11 de telle manière à réduire les interférences entre les transmissions des diverses stations d'une cellule.

Etalement de spectre à séquence directe La technique DSSS (Direct Sequence Spread Spectrum - étalement de spectre à séquence directe) consiste à transmettre pour chaque bit une séquence Barker (parfois appelée bruit pseudo-aléatoire) de bits. Ainsi chaque bit valant 1 est remplacé par une séquence de bits et chaque bit valant 0 par son complément. La couche physique de la norme 802.11 définit une séquence de 11 bits (10110111000) pour représenter un 1 et son complément (01001000111) pour coder un 0. On appelle chip chaque bit encodé à l'aide de la séquence. Cette technique module chaque bit avec la séquence barker. Grâce au chipping, de l'information redondante est transmise, ce qui permet d'effectuer des contrôles d'erreurs sur les transmissions, voire de la correction d'erreurs.

Dans le standard 802.11b, la bande de fréquence 2.400-2.4835 GHz (d'une largeur de 83.5 MHz) a été découpée en 14 canaux séparés de 5MHz, dont seuls les 11 premiers sont utilisables aux Etats-Unis. Seuls les canaux 10 à 13 sont utilisables en France. Toutefois, pour une transmission de 11 Mbps correcte il est nécessaire de transmettre sur une bande de 22 MHz car, d'après le théorème de Shannon, la fréquence d'échantillonnage doit être au minimum égale au double du signal à numériser. Ainsi certains canaux recouvrent partiellement les canaux adjacents, c'est la raison pour laquelle des canaux isolés (les canaux 1, 6 et 11) distants les uns des autres de 25MHz sont généralement utilisés. Le standard 802.11a utilise la bande de fréquence 5.15GHz à 5.35GHz et la bande 5.725 GHz à 5.825 GHz, ce qui permet de définir 8 canaux distincts d'une largeur de 20Mhz chacun, c'est-à-dire une bande suffisamment large pour ne pas avoir de parasitage entre canaux.

Infrarouge Le standard IEEE 802.11 prévoit également une alternative à l'utilisation des ondes radio : la lumière infrarouge. La technologie infrarouge a pour caractéristique principale d'utiliser une onde lumineuse pour la transmission de données. Ainsi les transmissions se font de façon uni-directionnelle, soit en "vue directe" soit par réflexion. Le caractère non dissipatif des ondes lumineuses offre un niveau de sécurité plus élevé. Il est possible grâce à la technologie infrarouge d'obtenir des débits allant de 1 à 2 Mbit/s en utilisant une modulation appelé PPM (pulse position modulation).

La modulation PPM consiste à transmettre des impulsions à amplitude constante, et à coder l'information suivant la position de l'impulsion. Le débit de 1 Mbps est obtenu avec une modulation de 16-PPM, tandis que le débit de 2 Mbps est obtenu avec une modulation 4-PPM permettant de coder deux bits de données avec 4 positions possibles.

F.2 Standard IEEE 802.16

F.2.1 Le Concept du IEEE 802.16

Les transmissions sans-fils sont devenues une solution populaire avec le WiFI présenté précédemment. Ce standard a été utilisé à la base pour connecter un LAN. En effet, la portée et le débit du protocole ont été conçu en ce sens. Le WiMAX pour "Worldwide Interoperability for Microwave Access" répond au besoin d'un standard gérant une forte sécurité, qualité de service et débit sur de longues distances, et unifiant l'architecture réseau globale. Tout l'intérêt de ce standard réside dans ses possibilités d'adaptation. Il est ainsi possible de choisir des interfaces physiques sans-fils spécifiques en fonction de la topologie d'un réseau, utiliser des sous-couches particulières ou encore un schéma de modulation adaptatif. Il peut être implémenté sur tous les continents en utilisant des fréquences assujetties à des licences ou encore libres :

- Entre 10 et 66 GHz (Bandes de fréquences sous licences) : L'utilisation de cette bande fréquence requière une ligne de vue en raison de la petite longueur de l'onde mis en oeuvre. Des canaux de 25 et 28Mhz sont typiquement utilisés, le débit maximum est de 120 Mbits/s. Le WiMAX offre les meilleurs performances dans ce cas. Il est à noter qu'un signal transmit sans aucune obstruction entre deux stations est appelé un Line-of-Sight signal (ligne de vue). Quand certains objets comme des arbres viennent interférer, le signal est appelé Non-line-of-sight (non ligne de vue).
- En dessous de 11 GHz (Bandes de fréquences sous licences) : En dessous de cette fréquence, une ligne-de-vue n'est pas requise en raison de la grandeur de l'onde utilisée. Dans ce cas, des paramètres supplémentaires liés à la MAC PHY sont requis comme les techniques de management de puissance ou de multiples antennes, une topologie en mesh ou encore des requêtes automatiques de renvoi de paquets. Ces techniques sont utilisés pour manager les interférences en encore simplement augmenter le débit.
- Bande de fréquences libres : Dans ce cas, le choix d'une fréquence doit s'effectuer en dessous de 11GHz, typiquement à 5-6 GHz. L'utilisation de cette bande de fréquences augmente les interférences en raison de la coexistence possible d'autres réseaux comme le WiFi. Des mécanismes avancés comme la "Dynamic Frequency Selection" sont utilisés pour solutionner ce problème.

F.2.2 La Structure du IEEE 802.16

Ce standard inclus le choix d'une couche physique particulière et de trois sous-couches MAC :

- Le Service-Specific Convergence Sublayer (CS) transcrit des unités externes de données en unités de service MAC.
- La MAC
- La sous-couche sécurité

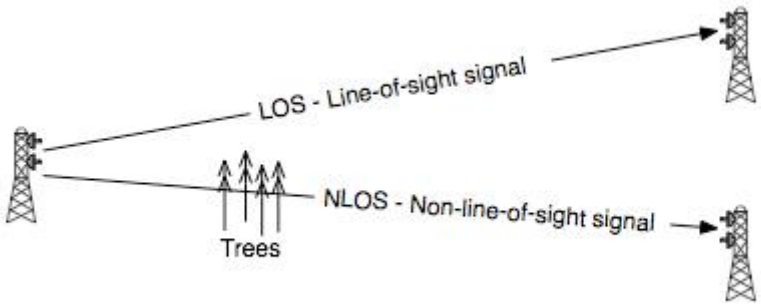

FIGURE F.2 – Definition de la ligne de vue

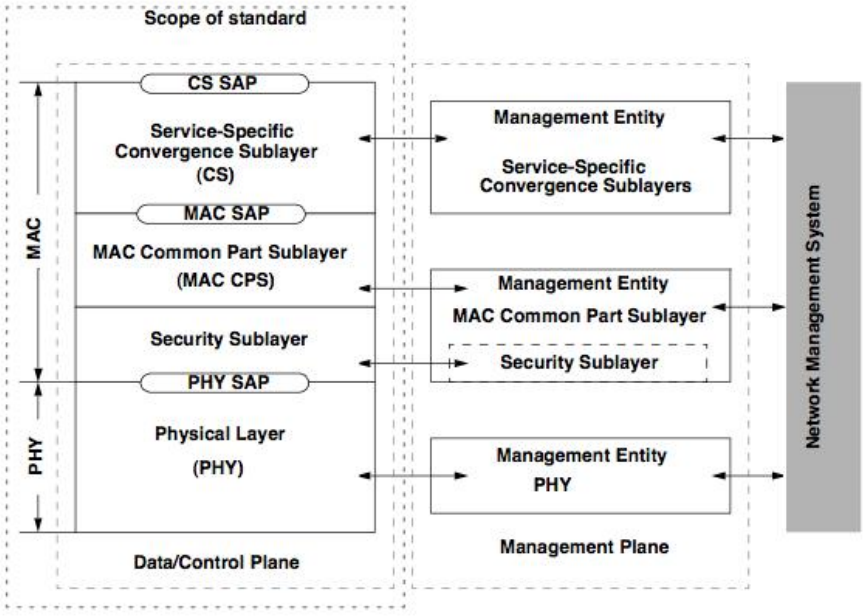

FIGURE F.3 – Modele OSI du WIMAX

Couche PHY La couche physique d'accès du WiMAX supporte deux types de duplexing : Frequency Division Duplex (FDD) et Time Division Duplex (TDD). Dans le mode TDD, les données sont transmises sur le même canal de fréquence à travers différentes périodes de temps. Dans le mode FDD, les données sont transmises en utilisant deux fréquences séparées généralement de 50 à 100 MHz correspondant aux liens montants et descendants .

Ces deux modes peuvent utiliser le processus Single Carrier ou encore OFDM. OFDM est basé sur un processus mathématique appelé Fast Fourrier Transform, qui permet à 52 canaux de se superposer sans perdre leurs caractéristiques individuelles. OFDM est également utilisé avec le standard 802.11g en raison de sa résistance aux interférences et de sa faible dégradation avec l'environnement. La révision finale du standard définit 5 variantes de couches physiques :

Nom	Freq (Ghz)	LoS/NLoS	Options	Duplexage
MAN-SC	$10-66$	LoS	$-$	TDD, FDD
MAN-SCa	$2,5-11$	$NLoS$	AAS, ARQ, STC	TDD, FDD
MAN-OFDM	$2,5-11$	$NLoS$	$AAS, ARQ, STC, Mesh$	TDD, FDD
MAN-OFDMA	$2,5-11$	$NLoS$	AAS, ARQ, STC	TDD, FDD
HUMAN	$2,5-11$	$NLoS$	$ARQ, STC, Mesh$	TDD

Chaque variante est optimisée pour une utilisation particulière et peut supporter des antennes adaptatives (AAS), Schéma de diversité (STC), Automatic Retransmission Request (ARQ), Topologie en mesh.

Utilisation Single Carrier La WirelessMAN-SC permet une utilisation flexible de la bande fréquence à travers la fréquence de 10-66 GHz et l'utilisation des techniques de multiplexage TDD et FDD. Une transmission en mode Full-duplex et half-duplex est supportée, ceci permettant quelques facilités de déploiement. Cette interface nécessite d'opérer en ligne-de-vue et convient bien aux réseaux de fournisseurs d'accès. La seconde variante du processus single carrier est la WirelessMAN-SCa pour Single Carrier Access et fonctionne en dessous de 11 GHz. Cette nomenclature fonctionne en non-ligne-de-vue à travers l'utilisation de composants supplémentaires.

Des antennes adaptatives (AAs) peuvent être utilisées, ainsi que le Space Time Coding (STC). Le schéma de codage utilisant FEC (Forward Error Correction) augmente le débit et la résistance à l'environnement. Ces variantes sont destinées à des communications Point à point.

Utilisation OFDM Toutes les variantes OFDM sont destinés à être utilisées en application résidentielles en NLOS. La variante WirelessMAN-OFDM utilise OFDM avec 256-points de transformation. Il supporte aussi les modes FDD et TDD. La "sous-channelisation" est supportée avec le lien montant à travers 16 "sous-canaux". Il n'y a pas de tel support pour le lien descendant. La WirelessMAN-OFDMA utilise 2048

points de transformation pour supporter l'accés de multiple récepteurs. FDD et TDD mode, STC, AAs et Multiple Input Multiple Output (MIMO) sont supportés. La "sous-channelisation" est supportée avec le lien montant aussi bien qu'avec le lien descendant.

Couche MAC La couche MAC du WiMAX supporte les opérations Point-to-multipoint (PMP) et mesh. En mode PMP, le lien descendant est généralement broadcasté ; dans une fréquence donnée et un secteur particulier, toutes les stations reçoivent la même transmission. La station réceptrice regarde l'identificateur de connexions (CID) dans les unités de données (PDUs) et ne retient que celles qui lui sont adressées. La station réceptrice partage le lien montant sur base d'un mécanisme de planification. La MAC supporte les opérations multicast et broadcast. Elle est orientée connexion, toutes les communications sont dans un contexte d'une connexion, chaque connexion est reliée à un flux de service. Ceci détermine le moyen de requête de bande passante. En mode Mesh, le trafic ne s'effectue pas uniquement de la station émettrice à la station réceptrice, le trafic est routé à travers tous les "voisins" à cette station ou encore directement entre les stations. Le trafic peut être géré selon un algorithme centralisé ou distribué.

F.3 Standard 3GPP LTE

F.3.1 Le Concept du 3GPP LTE

LTE est un acronyme pour Long Term Evolution, technologie appartenant à la branche GSM et qui repose initialement sur la base du 3G WCDMA. Cette technologie représente une voie de plus en plus sûre vers l'après 3.5G, au-delà du HSPA (High Speed Packet Access). Les évolutions des systèmes de communication sans fil se sont accélérées ces dernières années plus rapidement que les entités internationales de normalisation l'avaient envisagées. Les deux entités internationales de normalisation qui travaillent à l'amélioration des performances des systèmes de communication radioélectriques, sont le 3GPP pour le monde occidental et le 3GPP2 pour l'Asie et l'Amérique du Nord.

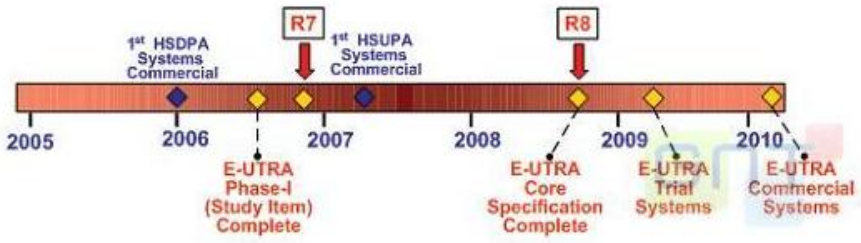

FIGURE F.4 – Evolution du Concept de LTE

Le "LTE" apparait comme une étape normative qui se veut définitive pour le moment. Ainsi en l'état actuel des avancées, le LTE associe les différentes normes des réseaux

d'accès radioélectriques numériques (RAN) utilisant la technique de paquets (avec le protocole IP) de sorte que les différents RAN puissent se marier avec les réseaux filaires en fibre optique, en câble ou en DSL (mêmes interfaces communs, mêmes capacités d'interfonctionnement, même système de signalisation). Ceci permettant d'ouvrir la voie à de nombreuses applications numériques de mobilité avec la complicité de la Toile. Détail important, le LTE porte le nom d'un de ses quatre constituants.

Concrètement l'édition 8 des normes du 3GPP, décrit l'architecture de réseau permettant de respecter les objectifs de qualité nécessaires dans l'EPS (Evolved Packet System), règles qui se partagent en deux volets : le SAE (Service Architecture Evolution) et le LTE (Long Term Evolution). En effet, pour unifier les réalisations antérieures et améliorer la qualité de service et les fonctions, il faut modifier légèrement les règles propres aux paquets d'information et améliorer l'architecture des réseaux de connexions vers l'accès. Le réseau EPS consiste en les entités suivantes :

- eNodeB
- Mobility Management Entity (MME)
- Serving Gateway
- Packet Data Network Gateway (PDN GW)
- Home Subscriber Server (HSS)
- Policy and Charging Rules Function (PCRF)

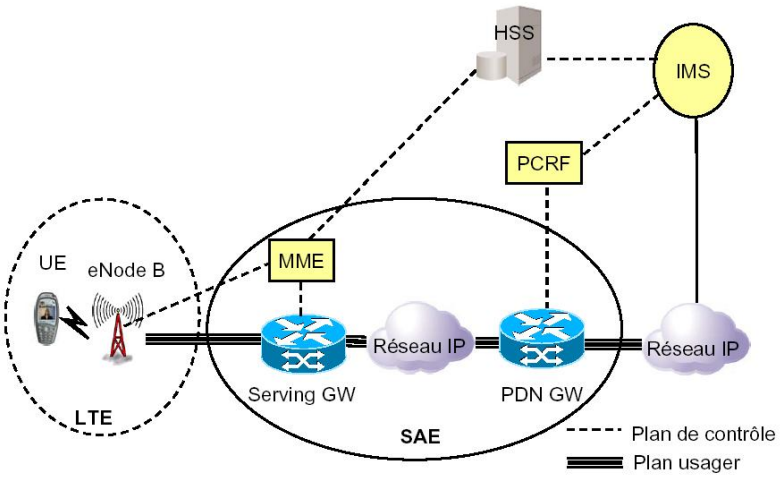

FIGURE F.5 – Principe de l'EPS

Aux deux volets précédents, le 3GPP a donc ajouté deux nouvelles spécifications majeures, l'EPC (Evolved Packet Core) pour le coeur de réseau et l'E-UTRAN (Evolved Universal Terrestrial Radio Access Network), pour les réseaux d'accès. Toute la révolution des applications tient dans le contenu de ces quatre documents techniques qui forment ensemble la nouvelle édition des normes relatives à la téléphonie mobile. L'EPC

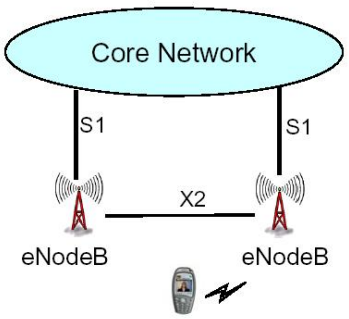

FIGURE F.6 – Principe de l'E-Utran

(Evolved Packet Core), définit un coeur de réseau basé sur le protocole IP qui permet aux exploitants de raccorder des réseaux d'accès de type 3GPP (LTE, 3G et 2G), des réseaux qui ne se réfèrent pas au 3GPP (par exemple, HRPD, WLAN, WiMAX) et des réseaux fixes en Ethernet, en fibre optique, en DSL ou en réseaux câblés. L'EPC garantit la mobilité, la gestion, la facturation et la sécurité des communications des terminaux en itinérance (déplacement d'un abonné d'un service mobile dans une zone différente de sa zone habituelle).

F.3.2 La Structure du 3GPP LTE

L'accès radio en LTE repose sur les techniques OFDM et l'accès aux canaux de fréquences met en jeu les ressources FDD et TDD, c'est-à-dire que l'échange entre les correspondants est de type duplex dans des canaux de fréquence utilisés en division de fréquences et par division temporelle, ce qui optimise les ressources offertes par le réseau. Ce double procédé permet l'accès à un plus grand nombre d'utilisateurs. Grâce à l'emploi de la technologie d'antennes multiples MIMO, les débits de crête peuvent atteindre 75 Mbit/s dans le sens montant et 300 Mbit/s dans le sens descendant. L'objectif du LTE est un système de radiocommunication mobile performant, optimisé paquets, pouvant atteindre des débits de 100 Mbit/s dans le Downlink et de 50 Mbit/s dans le Uplink. LTE peut être opéré dans les mêmes bandes de fréquences que l'UMTS. Bien qu'entraînant de nombreuses innovations techniques, la technologie LTE fait partie des évolutions UMTS.

Le 3GPP opte pour de nouveaux procédés de transmission et de modèles d'architecture pour le LTE, lequel ne sera plus basé sur l'accès WCDMA utilisé en UMTS. Le LTE Downlink utilisera la procédure d'accès OFDMA déjà définie dans le cadre du WiMAX. Les systèmes OFDMA offrent une transmission de données robuste avec une bonne efficacité spectrale. Le procédé de transmission utilisé dans le LTE Uplink est le SC-FDMA (Single Carrier Frequency Division Multiple Access), retenu notamment en raison de ses caractéristiques de signal favorables.En effet, les signaux SC-FDMA

disposent de facteurs de crête typiquement plus faibles que les signaux OFDMA et facilitent par conséquent la conception des amplificateurs de puissance pour les terminaux. Le SC-FDMA peut être vue comme un procédé OFDMA linéaire précodé (figure F.7).

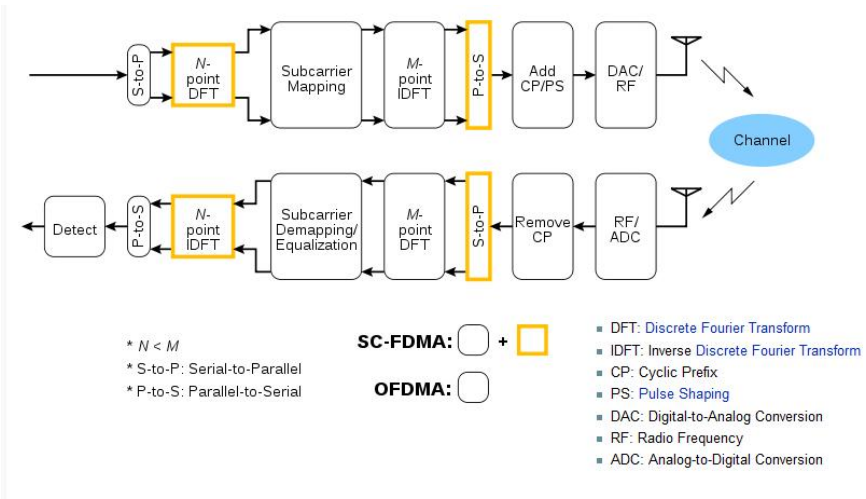

FIGURE F.7 – Principe du SC-FDMA

L'utilisation de deux antennes émettrices (côté station de base) et de deux antennes réceptrices (côté terminal) permettent de transmettre simultanément des trains de données indépendants sur la même ressource radio. Ces trains de données peuvent être destinés à un ou plusieurs usagers. Les systèmes MIMO offrent une capacité nettement supérieure et constituent de ce fait un élément essentiel de LTE. Outre ces procédés physiques de transmission, le LTE renonce à l'utilisation de canaux dédiés qui assignent une ressource déterminée à un usager pour la durée totale d'une liaison , la station de base LTE informe en effet les usagers de la ressource disponible pour une transmission de données. Ce principe appelé "Shared Channel", déjà connu de HSDPA, est optimal pour des services de transmission orientés paquets.

Annexe G

Le Banc d'Opérateurs Communs Optimisé

Sommaire

G.1 Détermination de T_r

G.1.1 Cas du BOC

Nous présentons en 5.2.2.1 la structure que nous développons de manière à diminuer le nombre de registres à implémenter et à définir un opérateur commun LFSR de taille la plus adaptée possible aux standards considérés. Nous créons ainsi, un banc d'opérateurs communs mais où les opérateurs communs sont interconnectés entre eux. Maintenant, nous devons déterminer la taille T_r du LFSR à considérer. Nous nous plaçons dans le cas de l'ER-LFSR qui remplace 100% des opérations en LFSR requises. Nous devons donc considérer toutes les structures énumérées dans les tableaux des figures 3.3, 3.10 et 3.19. Dans un premier temps nous minimisons l'expression, Taille(T_r) = $Max_j(\sum_i(N_{ch,ij}.E(\frac{N_{r,ij}}{T_r}).T_r))$:

1. Pour chaque standard/mode possible, la quantité $\sum_i(N_{ch,ij}.E(\frac{N_{r,ij}}{T_r}).T_r)$ est évaluée pour une gamme de T_r comprise entre 1 et 32. En effet, comme la taille maximale requise est 32, au delà de celle-ci, le nombre de registres ne peut qu'augmenter linéairement.
2. Ensuite, pour chacun des T_r, nous évaluons sur tous les standards, le maximum

de la quantité Taille(T_r) qui définit la taille requise de la structure pour chaque T_r.

3. Finalement, nous sélectionnons la valeur de Taille(T_r) la plus faible.

Celle-ci correspondra à une taille T_r du LFSR et à son nombre d'instances associées N_{T_r}. Pour plus de simplicité, nous donnons les résultats de ce calcul sous forme de graphe. En abscisses, sont représentées les valeurs de T_r et en ordonnées le nombre de registres requis. D'après les calculs, il apparait que la taille minimale est atteinte pour $T_r = 1$, où la taille de la structure est égale à la taille NTS. De plus, nous pouvons remarquer que la taille de la structure évolue linéairement à l'exception des valeurs qui sont les plus proches des PPCM des différentes tailles requises $(8, 12, 24)$. Cette approche supposerait d'implémenter une structure de 156 registres, tous interconnectés par des liaisons paramétrables. Si en termes de registres, cette implémentation pourrait être la plus économe, nous ne devons pas oublier que chaque interconnexion paramétrable aura un coût à prendre en compte.

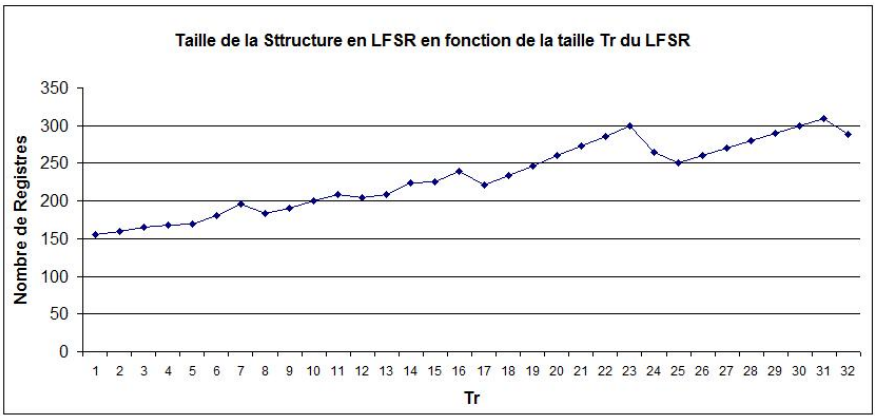

FIGURE G.1 – Taille de la Structure en LFSR

Afin de tenir compte des interconnexions, nous évaluons un coût approximatif des structures vis à vis des transistors requis. Il est vrai que notre cible première étant un FPGA, la logique combinatoire ne se répercutera pas sous la forme de transistors mais de tables de vérité (LUT). Néanmoins, comme nous travaillons à un niveau binaire, avec des structures qui peuvent être définies facilement par des portes logiques de base, nous utilisons cette approche pour appréhender le surcoût des liaisons. En chapitre III, nous définissons l'architecture de l'ER-LFSR comme constituée d'une structure dite en H qui se répète N fois pour une taille N de registres. Afin d'évaluer le coût en transistors de chaque structure, nous décomposons un LFSR de taille T_r en trois parties :

- T_r cellules en H.
- 1 cellule de Feedback qui permet d'assurer la redistribution des données sur les T_r cellules en H.

– 1 cellule de liaison qui permet d'assurer les interconnexions entre les LFSR de taille T_r.

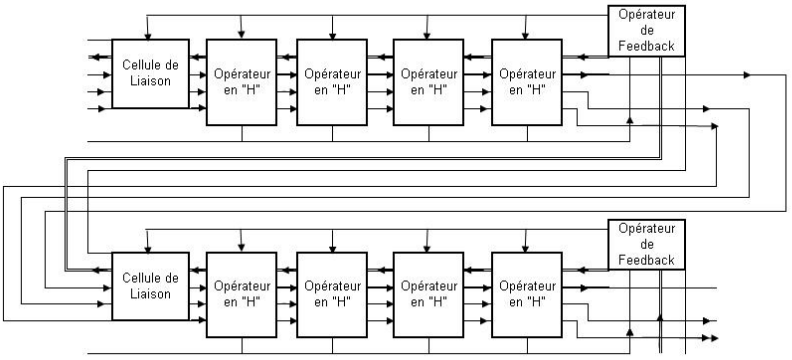

FIGURE G.2 – Décomposition des LFSR

Pour évaluer, le coût en transistors nous nous basons sur la description en logique combinatoire des cellules en H du chapitre III. Nous considérons la décomposition de chaque porte logique en portes logiques de base (figure G.4) que sont les portes NAND, NOR et NOT implémentées via des transistors CMOS (figure G.3). Ainsi pour l'ER-LFSR, nous arrivons au résultat :

Element :	*Nombre de Transitors*
Cellule H	110
Cellule Feedback	50
Cellule Liaison	52

TABLE G.1 – Coût en Transistors des Cellules du ER-LFSR

En appliquant le même raisonnement que précédemment sur les T_r mais en se focalisant sur le coût en transistors, nous obtenons les résultats de la figure G.5. Ainsi, le résultat le plus faible en termes de transistors est obtenu pour une taille T_r égale à 8 et un nombre d'instances égal à 23. Cette taille correspond concrètement à la taille de la structure requise la plus utilisée et constitue un diviseur de la majorité des autres tailles à considérer.

G.1.2 Cas d'un OC cadencé

Même si le réseau de LFSR est développé en priorité pour diminuer le nombre de registres dans un système de Banc, il peut être utilisé par n'importe laquelle des structures qui utilise des LFSR. Ainsi, nous pouvons appliquer ce réseau d'interconnexions aux deux premiers OC LFSR, OC_{LFSR_1} et OC_{LFSR_2}. Dans ce cas, nous ne cherchons

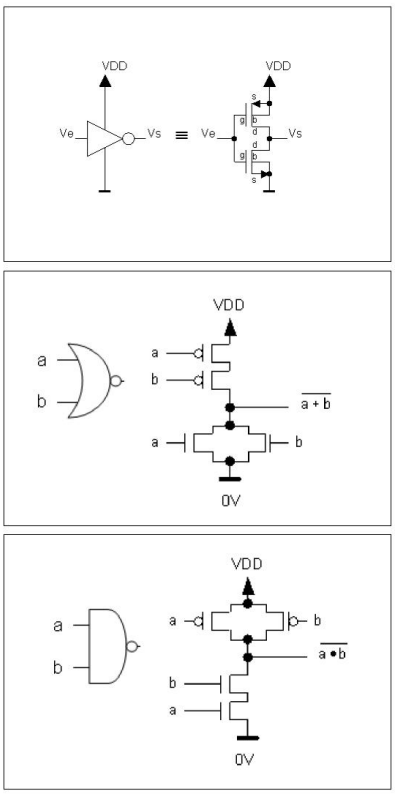

FIGURE G.3 – Décomposition des Portes Logiques de Base en Transistors

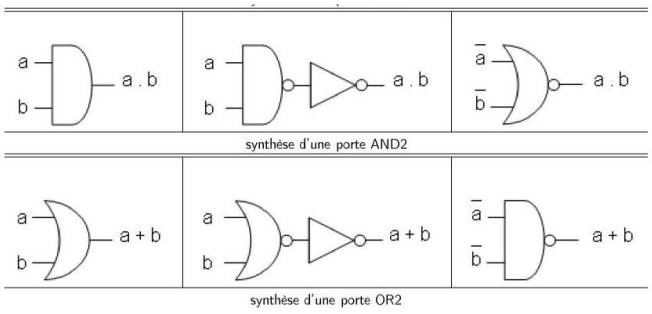

FIGURE G.4 – Décomposition des Portes Logiques en Portes Logiques de Base

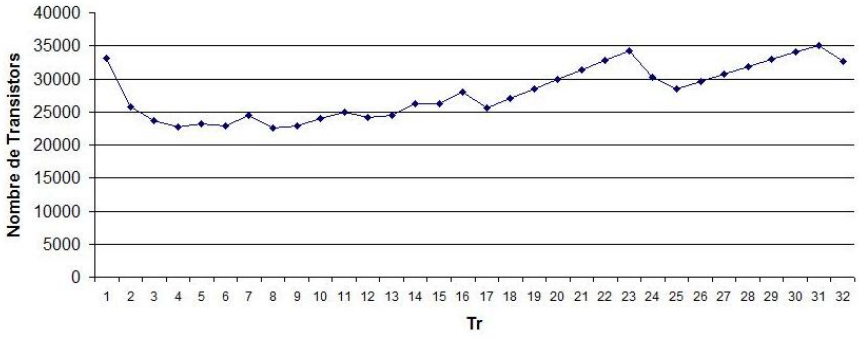

FIGURE G.5 – Coût de la Structure en LFSR pour un BOC tri-Standard

plus à tenir compte de l'exécution de différentes fonctions sur le Banc, étant donné que ces deux OC ne sont dédiés qu'à l'exécution d'une seule opération à la fois. Pour évaluer T_r, nous devons évaluer chaque fonction indépendamment et trouver T_r qui minimise :

$$Max_{ij}((N_{ch,ij}.E(\frac{N_{r,ij}}{T_r}).T_r)) = N_{T_r}.T_r \tag{G.1}$$

Ici, le calcul est plus rapide que dans le G.1.1, car il suffit d'évaluer pour une gamme de T_r, le maximum sur toutes les structures de la quantité $(N_{ch}.E(\frac{N_r}{T_r}).T_r)$. Comme précédemment, nous intégrons le coût des liaisons en transistors et donnons les résultats en figure G.6. Ici, aussi le T_r à privilégier sera égal à 8. Cette minimisation obtenue pour une taille de 8 résultant du générateur de séquences de Gold, nécessitera 8 instances pour l'ER-LSR et 16 pour le R-LFSR.

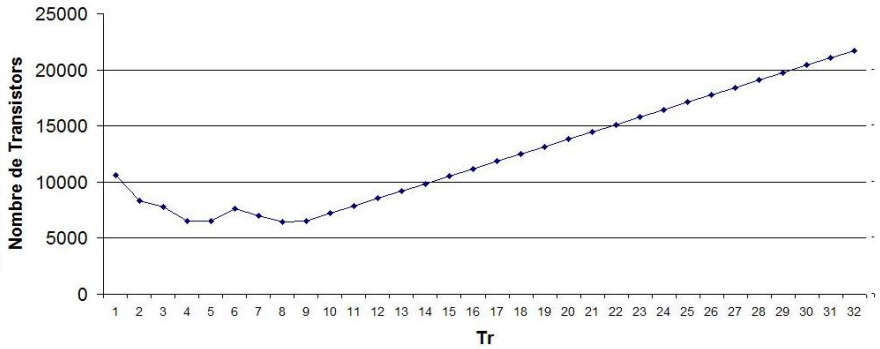

FIGURE G.6 – Coût de la Structure en LFSR pour un OC cadencé

Il est à noter que les mêmes calculs ont été mené pour le R-LFSR et des résultats similaires ont été obtenus pour les deux meilleures valeurs de T_r possibles c'est à dire 8 puis 4. Ainsi, tous les résultats donnés dans ce rapport avec T_r égal à 8 ont été également réalisés pour T_r égal à 4. Dans tous les cas, les résultats pour T_r égal à 8 sont meilleurs que T_r égal à 4. Pour ne pas surcharger le rapport, ceux sont uniquement ces résultats que nous présentons.

G.2 Allocation des Standards sur le réseau de LFSR

Maintenant que nous venons de définir la taille T_r pour un système en Banc, nous expliquons ici concrètement comment nous implémentons le Banc d'Opérateurs Communs qui utilise la structure spécifique en LFSR. Comme nous le précisons en chapitre II, la considération d'OC ne nous dédouane pas de considérer l'aspect fonctionnel. Ainsi, nous devons considérer dans le pire des cas, 7 fonctions différentes. Deux de Codage et décodage CRC, une de codage convolutif et deux fonctions différentes de scrambler/déscrambler, une obligatoire dans chacun des modes et une autre optionnelle pour les différents cas du WIMAX. Comme nous ne travaillons que sur une partie du terminal, nous pouvons définir un niveau d'abstraction de plus haut niveau qui considère les fonctions de manière à obtenir un point de comparaison. Ainsi, nous devons répartir ces dites 7 fonctions sur l'enchainement des 23 ER-LFSR (figure G.7). Ce mapping sera dans notre cas prédéfini pour les trois standards que nous utilisons. Il s'agira concrètement de faire correspondre les entrées et les sorties du BOC aux entrées des fonctions. Celà sera fait par un réseau d'interconnexions sous forme de switchs. Ici, contrairement à la technique des fonctions communes le réseau de switchs change les affections des entrées mais utilise la même structure.

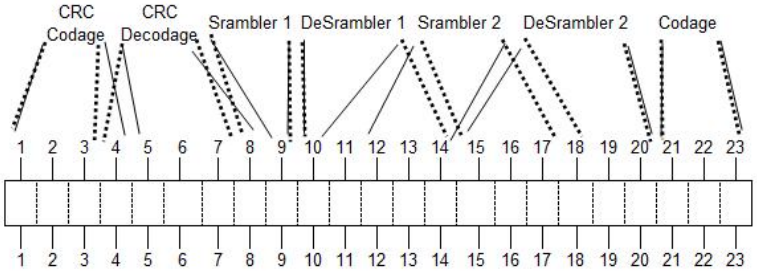

FIGURE G.7 – Repartition des Fonctions sur le BOC

Dans notre cas de figure spécifique, nous pouvons faciliter l'implémentation du design. En effet, nous pouvons identifier un paramétrage minimum de la structure qui permet de définir au préalable une partie des liaisons, c'est à dire qu'une partie des interconnexions sera systématiquement la même quelque soit le standard implémenté.

Le figure G.8 donne une représentation de cette structure prédéfinie. Ainsi, nous pouvons définir 9 sous structures indépendantes qui permettront de mapper les 7 fonctions à considérer. De plus, nous sommes en mesure de répartir ces sous structures pour les mêmes fonctionnalités dans la majorité des standards, diminuant de la sorte, le réseau d'allocation des fonctions. Prenons l'exemple du 3GPP LTE, pour les différents modes possibles chacune des 5 fonctions est physiquement allouée aux même opérateurs communs. Par exemple, pour le bloc dédié au CRC, c'est par la paramétrisation spécifique des sous structures liées aux différents polynômes générateurs que les différents CRC seront obtenus. Pour les deux autres standards, les mêmes propriétés sont possibles mais dans une moindre mesure. Pour le 802.11, les fonctions de CRC sont reliées aux LFSR de 1 à 4 et de 7 à 8 et les fonctions de scramblers aux LFSR de 9 à 11 et de 12 à 14. Dans ces cas, la différenciation des fonctions se fera également par paramétrisation du polynôme générateur adéquate. En ce qui concerne la fonction de codage, elle sera reliée soit aux LFSR 21 à 23 pour le codeur NSC 1/2 soit aux deux structures de 15 à 17 et de 18 à 20 pour le mode ERP NSC 2/3. Il est à noter que les deux options de codages sont implémentées sur des structures dédiées à cet effet et qui ne seront pas utilisées autrement par le standard. Nous pouvons des lors paramétrer une seule fois les deux structures pour n'importe lequel des modes 802.11. En ce qui concerne le 802.16, il suit le même principe que le 802.11 pour les deux fonctions de Scrambling (obligatoire) et de CRC. Par contre, les différentes possibilités entre modulation SC, SCa, OFDM et OFDMA induisent des besoins hétérogènes et ne permettent pas une séparation des allocations par fonction. Le turbo codeur duo binaire nécessite deux fois deux instances de l'ER-LFSR et sera alloué en deux fois des LFSRs 18 à 23. Hormis, cette structure, les opérations de codage sont implémentées du LFSR 21 au 23 et les fonctions de scrambling du 15 au 20. Maintenant, si on regarde vis à vis des trois standards, l'allocation des fonctions sur les opérateurs reste limitée. Les CRC alterneront entre deux possibilités, les scramblers (obligatoire) auront une allocation fixe du LFSR 9 au 14 et une optionnelle du 1 au 8 pour le 3GPP. La fonction de codage, occupera les trois LFSRs indépendants 21 à 23, plus du 15 au 20 pour le turbo code du WIMAX. Ainsi, le réseau d'interconnexions des fonctions sur le Banc d'Opérateurs Commun restera limité en complexité.

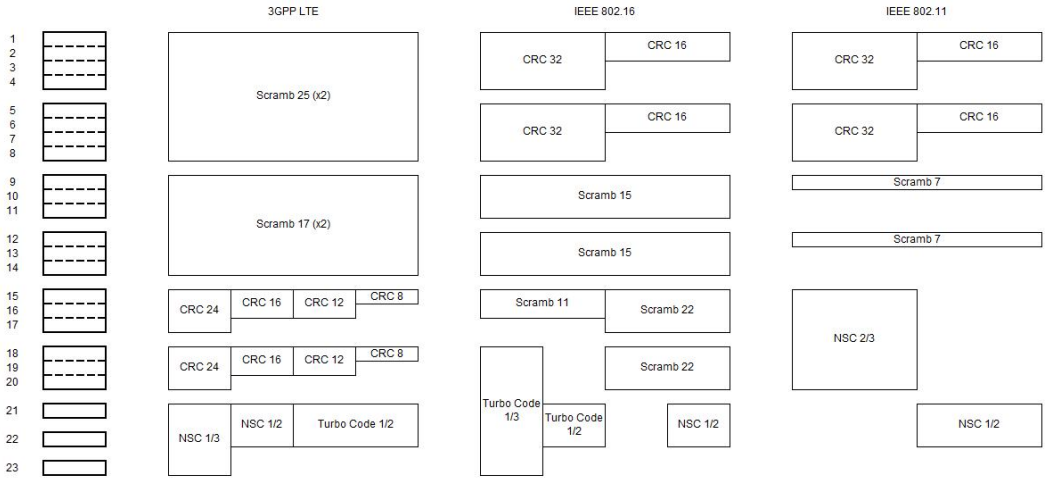

FIGURE G.8 – Repartition des Fonctions sur le BOC - Details

Annexe H

Implémentation de l'algorithme CORDIC avec LFSR

Sommaire

H.1 Le principe de l'algorithme CORDIC.

L'objectif de l'algorithme de CORDIC est de permettre d'obtenir une rotation d'angle α par une succession de plus petites rotations d'angles α_i. Pour cela, nous nous plaçons dans un espace vectoriel réel de dimension 2 où est définie une forme bilinéaire :

$$[(x,y),(x',y')] \rightarrow (x,y)Q_m.\begin{pmatrix} x' \\ y' \end{pmatrix} \tag{H.1}$$

Tel que $Q_m = \begin{bmatrix} 1 & 0 \\ 0 & m \end{bmatrix}$ et la semi-norme associée $\|(x,y)\| = \sqrt{x^2 + m.y^2}$

Pour m =0, +1, -1, la matrice de rotation généralisée $\theta_m(x)$ possède la propriété d'orthogonalité par rapport à Q_m avec $= \theta_m.Q_m.\theta_m = Q_m$.

$$\theta_m(\alpha) = \begin{pmatrix} cos(\sqrt{m}.\alpha) & -\sqrt{m}.sin(\sqrt{m}.\alpha) \\ \frac{1}{\sqrt{m}}.sin(\sqrt{m}.\alpha) & cos(\sqrt{m}.\alpha) \end{pmatrix}$$

L'algorithme CORDIC peut être utilisé dans trois cas d'utilisation ; (1) Euclidien pour m=1, (2) Hyperbolique m=-1 et (3) Linéaire pour m=0.

Les matrices $\theta_m(x)$ associée à chacun des cas sont :

$$\theta_1(\alpha) = \begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix}$$

$$\theta_{-1}(\alpha) = \begin{pmatrix} ch(\alpha) & sh(\alpha) \\ sh(\alpha) & ch(\alpha) \end{pmatrix}$$

$$\theta_0(\alpha) = \begin{pmatrix} 1 & 0 \\ \alpha & 1 \end{pmatrix}$$

Pour assurer la rotation du vecteur (x,y) par un angles α, nous décomposons α en une succession d'angle α_i dont nous faisons subir des rotations successives au vecteur (x,y) tel que :

$$\alpha = \sum \mu_i.\alpha_i, avec\ \mu_i = \pm 1. \tag{H.2}$$

A chaque itération, l'opération effectuée est l'opération élémentaire $\theta_m(\mu_i.\alpha_i)$.

$$\theta_m(\mu_i.\alpha_i) = \begin{pmatrix} cos(\sqrt{m}.\alpha_i) & -\mu_i.\sqrt{m}.sin(\sqrt{m}.\alpha_i) \\ \mu_i.\frac{1}{\sqrt{m}}.sin(\sqrt{m}.\alpha_i) & cos(\sqrt{m}.\alpha_i) \end{pmatrix}$$

Ainsi, le calcul de $\theta_m(\alpha)$ devient :

$$\theta_m(\alpha) = \theta_m(\sum_{i=1}^{n} \mu_i \cdot \alpha_i) = \prod_{i=1}^{n} \theta_m(\alpha_i.\mu_i)$$

$$\theta_m(\alpha) = \prod_{i=1}^{n} \frac{1}{\sqrt{1 + tg^2.(\sqrt{m}.\alpha_i)}} \times \prod_{i=1}^{n} \begin{pmatrix} 1 & -\mu_i\sqrt{m}.tg(\sqrt{m}.\alpha_i) \\ \mu_i.\frac{1}{\sqrt{m}}.tg(\sqrt{m}.\alpha_i) & 1 \end{pmatrix}$$

Ainsi, à partir de cette écriture, le CORDIC peut s'écrire sous forme itérative entre les différents vecteurs (x_i, y_i) qui représente un état d'avancement dans la succession des rotations successives :

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \theta_m(\alpha_i) \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

.

Il apparaît que le facteur de normalisation α, qui permet d'assurer la conservation de la norme entre l'entrée et la sortie de l'algorithme, peut donc être calculé une fois pour toutes [KAI85b], la correction intervenant alors seulement en fin de calcul. Ainsi,

l'algorithme CORDIC est une mise en application simple d'un registre à décalages pour réaliser une division par deux. La représentation binaire quantifiée sur N bits est insérée dans le registre à décalage et chaque décalage correspond à une division par deux. En faisant provisoirement abstraction du facteur de normalisation (terme en cos), on peut écrire :

$$x_{i+1} = x_i - \mu_i.\sqrt{m}.tg(\alpha_i.\sqrt{m}).y_i$$
$$y_{i+1} = y_i - \mu_i.\frac{1}{\sqrt{m}}.tg(\alpha_i.\sqrt{m}).x_i$$

.

Le principe de base de l'algorithme CORDIC est de prendre $(\frac{1}{\sqrt{m}}.tg(\alpha_i.\sqrt{m}))$ égal à une puissance de deux, ou plus généralement à la base de calcul de l'arithmétique utilisée, de telle sorte que les itérations ne comportent que des additions et des décalages :

$$x_{i+1} = x_i - m.\mu_i.2^{(m,i)}.y_i$$
$$y_{i+1} = y_i + \mu_i.2^{(m,i)}.x_i$$

.

H.2 Le Reconfigurable LFSR CORDIC - R-LFSR CORDIC

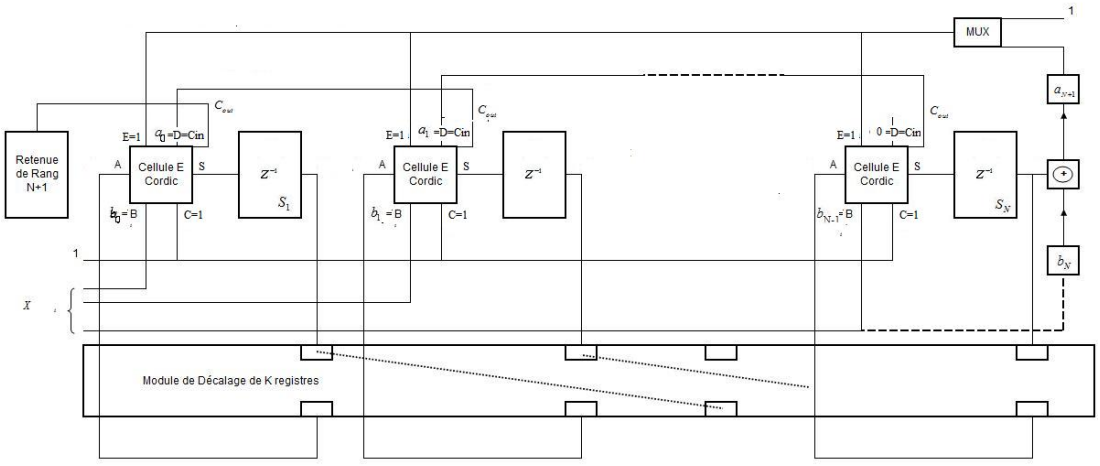

FIGURE H.1 – Schématique du R-LFSR Cordic

Dans la recherche d'un Opérateur Commun, nous voulons définir un opérateur qui puisse être utilisé par le maximum de fonctions afin de limiter le nombre d'opérateurs à considérer et de définir un design reconfigurable et capable de s'adapter à des opérations non initialement prévues. L'opérateur CORDIC a été défini dans le premier chapitre comme pouvant réaliser un grand de fonctionnalités différentes au sein d'un terminal de

télécommunication. Nous venons de démontrer dans la première partie des opérateurs à base de registres à décalages que nous définissons des opérateurs qui sont capables de réaliser la majorité des opérations de FEC. L'intérêt d'un opérateur commun et de ne pas se focaliser sur un type de fonctionnalités et d'exécuter le maximum d'opérations possibles. Ainsi, nous proposons ici, une évolution de l'opérateur R-LFSR afin de le rendre capable d'être implémenté dans un module de l'algorithme CORDIC. En se fiant à l'équation :

$$x_{i+1} = x_i - m.\mu_i.2^{(m,i)}.y_i$$
$$y_{i+1} = y_i + \mu_i.2^{(m,i)}.x_i$$

.

Il apparait que le calcul à chaque itération du CORDIC nécessite une division par une puissance de 2. Cette opération peut être facilement réalisée par n'importe lequel des LFSR présentés dans nos travaux étant donné qu'une division par 2 consiste à un seul décalage du registre et qu'une division par 2^k consiste à k décalages. Cette utilisation d'un de nos LFSR est triviale et nous désirons réellement définir nos architectures en LFSRs comme parties prenantes du calcul des CORDIC et ne pas nous limiter au décalage classique. Le R-LFSR (et aussi l'ER-LFSR), possède en plus du système de N registres à décalages, N cellules en E (en H pour le ER-LFSR) qui permettent d'opérer des additions dans GF(2). Pour rappel dans GF(2), la cellule en E pour les valeurs d'entrée A,B,C,D et E donnent la sortie S tel que :

$$S = A + B.C + D.E \; ou \; S = A \; XOR \; (B \; AND \; C) \; XOR \; (D \; AND \; E)$$

Nous exploitons et "upgradons" le potentiel de calcul de la cellule pour réaliser des opérations sur des réels codés en Binaires. De plus, nous utilisons la structure du R-LFSR qui nous permet d'opérer sur les valeurs "en train d'être décalées", pour effectuer simultanément une division par 2^k et l'addition de la donnée divisée avec une deuxième valeur. Ainsi, nous définissons dans cette partie une architecture intitulée R-LFSR CORDIC qui se défini par l'équation reliant un succession d'entrées X_p et une sortie Y_p tel qu'à la pième itération, la sortie du R-LFSR de CORDIC donne :

$$Y_p = \sum_{j=1}^{p} \frac{X_j}{2^{k(p-j)}} \tag{H.5}$$

Dans cette configuration, les entrées X comme les sorties Y sont des nombres réels codés sur N bits, où N reste la taille du LFSR considéré. Ainsi, contrairement aux applications précédentes de nos LFSRs, nous travaillons avec des registres de type PIPO (Parrallel In Parralel Out). L'entrée X codée sur N bits correspondra aux N premiers coefficients b_i du R-LFSR et la sortie Y au N valeurs des registres. Il est à noter que dans cette paramétrisation, l'entrée série x comme la valeur de rétroaction du LFSR seront arbitrairement fixées à 1. Comme nous l'expliquons dans la suite de ce paragraphe, le

coefficient b_N ne sera pas utilisé et les N coefficients a_i seront utilisés pour la propagation de la retenue binaire.

Si nous nous référerons aux besoins du CORDIC, en ne tenant pas compte des facteurs m et μ_i qui sont égaux à ± 1, les opérations du types : $x_i - m.\mu_i.2^{(m,i)}.y_i$ ou $y_i + \mu_i.2^{(m,i)}.x_i$ reviennent à calculer : une opération du type : $X \pm \frac{Y}{2^i}$.

Grâce à un codage des données en compléments à deux, pour p=2, l'opérateur R-LFSR CORDIC réalisera donc : $\frac{X_1}{2^i} + X_2$ en posant $X_2 = \mathrm{X}$ et $X_1 = \mathrm{Y}$, nous répondons bien au besoin du module de CORDIC.

En étant capable de générer les itérations de l'algorithme CORDIC, nous permettons de ramener l'ensemble des opérations qu'il peut exécuter à l'utilisation d'une fonction basée sur les LFSR. Nous expliquons dans la suite de ce paragraphe les modifications qui nous permettent d'obtenir ce résultat. Celles-ci s'organisent en trois étapes :

- Première étape : Division d'ordre 2^k.
- Deuxième étape : Définition du Calcul : $\sum_{j=1}^{p} \frac{X_j}{2^{k(p-j)}}$
- Troisième étape : Adaptation à la soustraction réelle par utilisation du complément à deux.

H.2.1 Première étape : Division d'ordre 2^k.

La première modification à apporter au R-LFSR est de pouvoir fournir une division paramétrable par une puissance de deux. En considérant une valeur réelle X codée en binaires sur les registres du LFSR, une division par 2^k correspond à k décalages. Afin d'utiliser le R-LFSR pour cette division nous définissons les coefficients b_i pour i $\in [0, N-1]$, comme les "nouvelles" entrées de notre R-LFSR. Ainsi, une fois X entrée au niveau des b_i, sa représentation binaire sera automatiquement insérée dans les registres et nous pourrons procéder aux décalages adéquats. Il est à noter que dans cette configuration, l'entrée ligne x devra être figé à "1".

Au préalable pour chaque nouvelle entrée du LFSR, nous réalisions un seul décalage, désormais nous paramétrons le décalage en fonction d'un paramètre k. Au Niveau du R-LFSR, la cellule de base dite "en E" est constituée de l'unité de Calcul "en E" et du registre. Pour le cas intrinsèque du R-LFSR l'entrée de la pième cellule correspond à la valeur du (p-1) ième registre. Dans le cas présent, cette même entrée de rang p prendra la valeur du (p-k)ième registre. Ceci revient à créer un nouveau mapping qui fait correspondre entrées et sorties des cellules en "E" (figure H.1).

H.2.2 Deuxième étape : Définition du Calcul : $\sum_{j=1}^{p} \frac{X_j}{2^{p-j}}$.

Défini initialement dans GF(2), le R-LFSR doit maintenant supporter des opérations d'additions de nombres réels codés en binaire. L'addition de deux nombres codés en Binaire se réalisent bits à bits mais comme en base 10, une retenue peut apparaître et doit être répercutée à l'addition binaire de puissance de deux supérieures. En décalant de k le registre au moment du décalage l'unité de calcul en E de rang p reçoit les données du (p-k)ième registre et du bit de rang p des coefficients b_i. C'est à dire que bit à bit, nous sommes en présence de l'écriture binaire de $\frac{X_{i-1}}{2^k}$ et de X_i. Nous pouvons donc

réaliser simultanément l'addition et la division par 2^k. Pour pouvoir réaliser l'addition, nous devons tenir compte de la propagation de la retenue et modifier notre cellule en "E". Si, nous considérons les deux entrées Binaires à additionner A et B et la retenue de l'addition de rang inférieur C_{in} alors les valeurs de la sortie S de l'additionneur et de la retenue C_{out} sont données par la table de vérité :

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

.

Ce qui correspond aux équations :

$$S = A\ XOR\ B\ XOR\ C_{in} \tag{H.7}$$

$$C_{out} = ((A\ XOR\ B)\ AND\ C_{in})\ OR\ (A\ AND\ B) \tag{H.8}$$

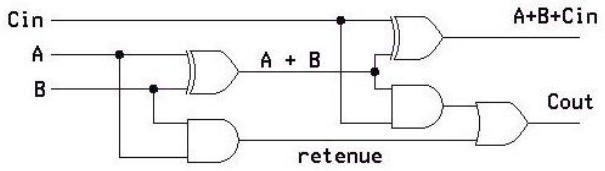

FIGURE H.2 – Schématique du Calcul de la retenue d'une addition binaire

Nous représentons le schéma d'un tel calcul en figure H.2. Ainsi, nous définissons une nouvelle cellule de calcul nommée "E_{Cordic}", capable de reprendre les calculs nécessaires au R-LFSR et à l'addionneur complet. Cette cellule est représentée en figure H.3. Elle reprend la cellule "E" initiale qui permet d'effectuer le Calcul de S et comprend le module de calcul supplémentaire pour la retenue. Le paramétrage est indiquée dans le cas d'une utilisation en mode CORDIC.

Les équations de notre cellule "E_{Cordic}" deviennent donc :

$$S = A\ XOR\ (B\ AND\ C) XOR\ (D\ AND\ E) \tag{H.9}$$

$$C_{out} = ((A\ XOR\ B)\ AND\ (D\ AND\ E)\ OR\ (A\ AND\ B) \tag{H.10}$$

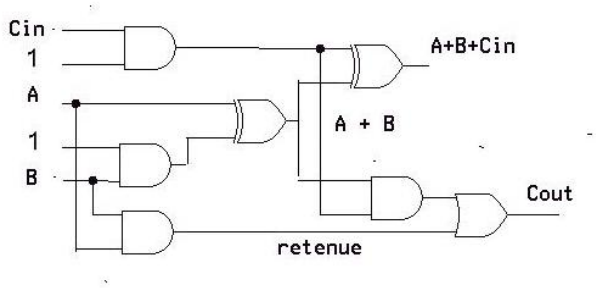

FIGURE H.3 – Schématique de la cellule E_{Cordic}

En posant A=A, B=B, C=1, $D = C_{in}$ et E=1, nous retrouvons bien les équations régissant le modules d'additions complets. Ainsi, dans ce paramétrage, la retenue p est distribuée au coefficient a_{p+k}. En modifiant, l'unité de calcul de chaque cellule du R-LFSR, nous pouvons des lors réaliser une addition binaire et obtenir la séquence en sortie du R-LFSR : $\sum_{j=1}^{p} \frac{X_j}{2^{k(p-j)}}$.

H.2.3 Troisième étape : Adaptation à la soustraction réelle par utilisation du complément à deux.

Le dernier est troisième point permettant de définir notre R-LFSR CORDIC est de répondre au besoin d'addition soustraction demandé par les calculs des différentes itérations CORDIC. Comme nous l'avons mentionné plus haut, l'opération nécessaire pour le CORDIC est $\frac{Y}{2^k} \pm X$ c'est à dire $\frac{X_{j-1}}{2^k} \pm Xj$ dans notre cas. Or notre opérateur tel que défini permet d'obtenir $\sum_{j=0}^{j=1} \frac{X_j}{2^{k.(p-j)}}$. Afin de ne pas modifier notre design et de ne pas complexifier d'avantage le R-LFSR, nous permettons d'effectuer l'opération voulue en nous basant sur des calculs en compléments à deux. Ainsi, nous pourrons utiliser le R-LFSR défini dans les deux points précédents pour effectuer des additions et des soustractions réelles codés en binaires. En effet, comme expliqué en [LS05], le complément à deux permet de transformer la représentation binaire d'un nombre réel en une représentation permettant d'opérer les calculs de soustractions par additions. Simplement, un nombre réel X sera représenté en binaire par $2^n - |X|$, si X est positif alors il gardera sa représentation binaire classique sur N bits alors que s'il est négatif, X sera modifié en son complément à deux. D'un point de vue pratique, le calcul du complément à deux se déroule en deux temps :

- On inverse les bits de l'écriture binaire de sa valeur absolue (opération binaire NOT), on fait ce qu'on appelle le complément à un,
- On ajoute 1 au résultat (les dépassements sont ignorés).

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \Rightarrow & 127 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \Rightarrow & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Rightarrow & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \Rightarrow & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \Rightarrow & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & \Rightarrow & -2 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Rightarrow & -127 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \Rightarrow & -128 \end{bmatrix}$$

Ainsi, en utilisant la complémentation à deux vis à vis des entrées du R-LFSR, nous pouvons opérer addition et soustraction réelle. Le complément à deux est le système de codage binaire utilisé dans les langages de programmation comme le VHDL pour marquer des entiers signés. En reprenant, ce système binaire précis, nous restons dans l'idée première de la technique des opérateurs communs en spécifiant un nouvel IP capable de s'intégrer directement dans les langages de programmation actuel.

Concrètement, le complément à deux réalise une translation de valeurs de 2^n pour ne plus réaliser des opérations par rapport à la valeur nulle mais par rapport à 2^n pour des nombres codés sur n bits. Ainsi, un nombre positif X_1 gardera son écriture traditionnelle en binaire mais en pratique représentera $2^n + X_1$, alors que si X_2 est négatif, il sera représenté par 2^n - $|X_2|$. Les calculs s'opère vis à vis d'une nouvelle valeur de référence qui n'est plus zéro mais 2^n. Néanmoins, si le complément à deux est la solution pour la soustraction, une modification mineure du R-LFSR doit être faite afin de pouvoir procéder à la division d'un nombre négatif complémenté. En effet, comme la division est associative par rapport à l'addition alors la division peut s'appliquer dans le cas d'un complément. Cependant, en ce qui concerne la représentation binaire d'un nombre négatif, la complémentation à deux se définit par $2^n - |X|$. Cela se traduit en binaire par la présence d'une série de 1 représentant la différence entre 2^n et $|X|$. Or lors du décalage des registres, cette série de 1 est elle aussi décalée, et "techniquement" remplacée par des "zéros". Pour conserver l'écriture en complément à deux associée à X, nous devons par conséquent, remplacer la série de bits laissés à zéro par des 1 caractéristiques de $2^n - |X|$ où plus précisément de $2^n - |\frac{X}{2^k}|$. Ceci uniquement dans le cas d'un nombre négatif. Nous illustrons ce cas de figure avec la division de -20 par 4 en figure H.4.

H.3 Implémentation des LFSRs - Cas Spécifique d'une application CORDIC

H.3.1 Fonctionnement du LFSR CORDIC

Il existe deux modes distincts d'utilisation du CORDIC, un mode vecteur et un mode rotation. Le mode rotation résout, simultanément à partir d'un angle initial, le sinus et le cosinus de ce dernier. Le mode vecteur calcule le module et la phase du vecteur à partir des coordonnées x et y.

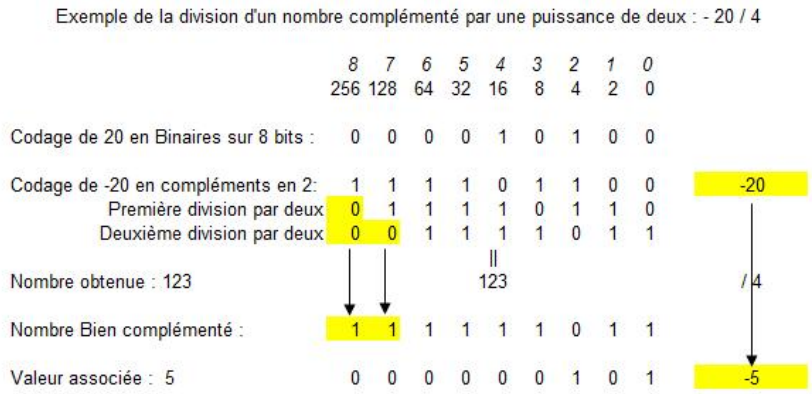

FIGURE H.4 – Exemple de Compléments à deux associé à une division

Mode Vecteur Pour calculer la phase et le module à partir des coordonnées x et y du vecteur, nous cherchons à annuler la partie imaginaire par des rotations successives. Les variables x_0, y_0 et θ_0 sont initialisées respectivement à x, y et 0 et nous exécutons les rotations jusqu'à l'annulation de y. Nous illustrons ce principe en figure H.5 avec un vecteur initial d'angle θ_0 égal à 100 degrès. Ensuite, nous appliquons une succession de rotation d'angle 90, 45, 26, 14 et 7 et ainsi de suite jusqu'à ce que le vecteur se confonde avec l'axe des abscisses.

.

Mode Rotation Dans ce mode, l'algorithme fait tendre la variable θ_0, initialisée à l'argument de la rotation voulue θ vers zéro. Le signe de y est alors défini par celui de θ, ce qui correspond à une commande par rétroaction asservissant les itérations de manière à faire tendre θ vers 0. L'angle de rotation est tout d'abord déterminé par le mode vecteur puis exécuté par le mode rotation. Cette mise à jour est effectuée en parallèle avec les itérations de calcul des rotations, sous la forme :

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} cos & -sin \\ sin & cos \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Prenons un exemple d'un vecteur de module 1 et d'angle initial de 36 degrés et un angle de rotation de 54 degrés. Cette angle de rotation sera obtenu par la succession de rotations : +90, -45, +26, -14, -7, +3 et +1.

.

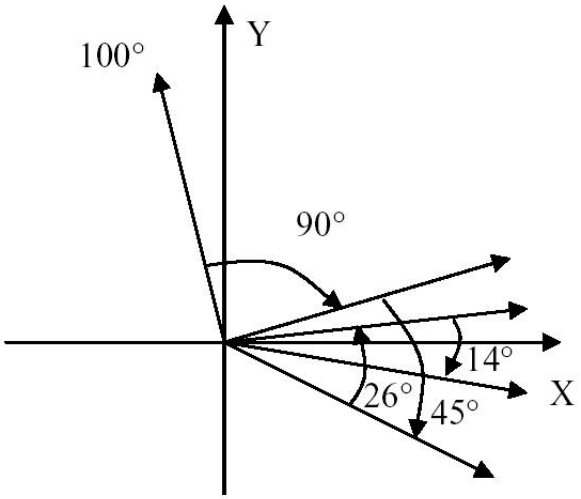

FIGURE H.5 – Exemple du CORDIC en mode Vecteur

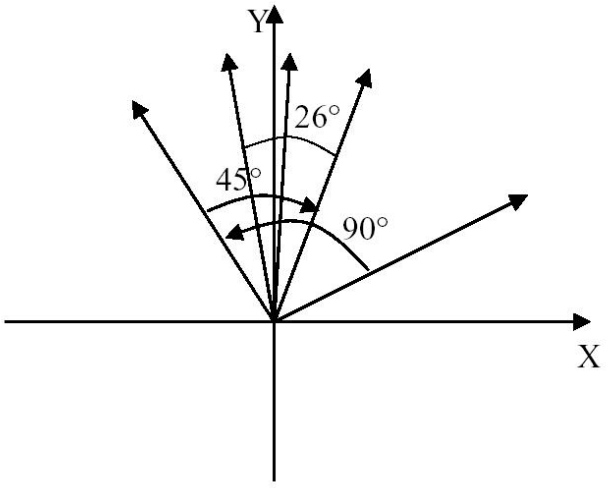

FIGURE H.6 – Exemple du CORDIC en mode Rotation

H.3.2 Architecture du CORDIC

Dans l'implémentation de l'algorithme CORDIC, trois architectures se distinguent selon le parallélisme des opérations à chaque itération et les profondeurs de pipeline des additionneurs/soustracteurs. Le choix est le fait d'un compromis entre la précision souhaitée, la vitesse de calcul nécessaire et le niveau de complexité demandé par l'application.

Architecture à opérations série et itérations série Cette architecture ne se compose que d'un seul additionneur/soustracteur. Elle exécute les opérations séries et des itérations séries. Les coordonnées x, y, et z sont calculées séquentiellement. Elle est l'architecture la moins complexe en termes de ressources matérielles mais également la moins performante et nécessite d'utiliser un séquenceur plus complexe par rapport aux autres architectures présentées par la suite. Ce dernier séquencement nécessite un nombre important de multiplexeurs.

Architecture à opérations parallèles et itérations série Cette architecture met en parallèle trois additionneurs/soustracteurs, chacun dédié à l'exécution d'une des trois équations de l'algorithme, en conservant néanmoins un calcul séquentiel des itérations. Cette architecture requière la sauvegarde des valeurs angulaires utilisées dans le calcul sur la variable auxiliaire z (ROM ou registre).

Architecture à opérations parallèles et itérations parallèles Cette architecture est la plus complexe sur le plan matériel, en autorisant une mise en oeuvre pipeline et en limitant ainsi la longueur des chemins critiques de façon à accroître la vitesse de fonctionnement. Dans cette structure, les décalages sont réalisés à l'aide de registres à chaque étage. L'architecture à opérations parallèles et itérations parallèles présentée ici est très intéressante, car elle permet d'atteindre des débits de calcul beaucoup plus élevés.

H.3.3 Mise en Oeuvre du CORDIC LFSR

Opérateur CORDIC Le R-LFSR CORDIC est défini comme un élément constitutif d'une architecture CORDIC qui opère l'opération de décalage/addition/soustraction, nécessaire à chaque étage. L'opérateur CORDIC peut être lui même construit selon une des architecture précédentes. C'est pour cette raison que nous définissons notre propre IP (Intellect Property) CORDIC qui est flexible et modulaire.

Nous définissons une architecture de type pipeline afin d'optimiser la fréquence de fonctionnement du circuit. Ce type d'architecture nécessite une succession d'étages identiques de manière à exécuter à chaque front d'horloge, l'intégralité des itérations nécessaires. A chaque étage, l'exécution des trois différentes équations est obtenue par l'utilisation de trois additionneurs/soustracteurs, trois multiplexeurs, deux registres à décalage réalisant une division par 2 pour x et y plus un registre pour stocker les valeurs des angles de rotation. Le choix entre addition et soustraction à chaque étage dépendra

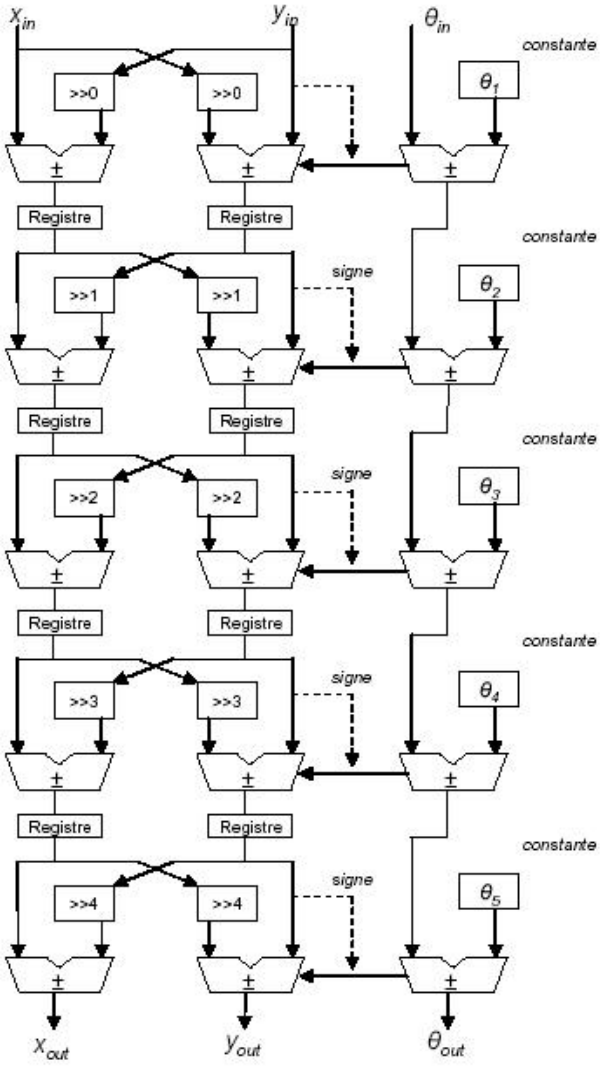

FIGURE H.7 – Schématique d'un module CORDIC à 5 étages

du signe de y ou θ en fonction du mode Vecteur ou Rotation. Le procédé d'une telle architecture est le suivant :

- Les valeurs initiales sont insérées dans les registres via des multiplexeurs.
- A chaque coup d'horloge, les valeurs des registres sont transmises aux additionneurs/soustracteurs et aux registres à décalage.
- Les résultats sont ensuite stockés dans les registres et le numéro de registre stockant les angles sera incrémenté à chaque coup d'horloge afin d'envoyer les valeurs des angles appropriées à l'additionneur/soustrateur.

Après N itérations, les résultats seront obtenus directement en sorties des additionneurs/soustracteurs. La figure H.7 donne une schématique de principe de l'architecture que nous développons avec 5 étages indépendants dans le cas d'un CORDIC en mode rotation. Les registres utilisés entre les étages ont pour fonction de construire la structure pipeline, ainsi que resynchroniser les signaux après les opérations effectuées par la logique combinatoire. Le signe de y défini le sens de rotation. Les valeurs (x_i, y_i), (x_{i+1}, y_{i+1}),..., (x_{i+n}, y_{i+n}) sont modifiées par rotations élémentaires d'étage en étage. D'après les explications précédentes, nous pouvons distinguer que le R-LFSR CORDIC ne sera qu'une partie de cette architecture, celui-ci n'opérant que les opérations successives de divisions et d'additions. Nous représentons en figure H.8, les étages de CORDIC que nous considérons, le premier implémenté directement par le biais d'un registre et d'un additionneur/soustracteur, le deuxième implémenté avec le R-LFSR CORDIC. La structure du R-LFSR CORDIC s'intègre directement dans le module CORDIC. La seule modification à prendre en compte est le signe de l'addition/soustraction qui doit être codée avant l'application du R-LFSR CORDIC.

Dans le cadre du projet ANR INFOP, nous réalisons l'implémentation, de la version sans et avec le R-LFSR CORDIC de taille 16, d'un CORDIC à 12 étages traitant des entrées de 16 bits, sur une cible Virtex-4 4vlx160 de Xilinx.

Type de Cordic :	Classique	R-LFSR / CORDIC
Nombre de Slices :	583	1636
Nombre de Registres :	1077	1361
Fréquence Maximale utilisable:	445 MHZ	317 MHz

TABLE H.1 – Résultats d'Implémentations d'un module CORDIC 12 étages (Avec ou sans le R-LFSR CORDIC de taille 16)

Intégration dans la FFT Nous présentons dans le chapitre II, différentes possibilités d'implémentations de l'opérateur CORDIC. En développant, le R-LFSR CORDIC, nous définissons un opérateur commun à partir d'un autre opérateur commun. Nous pouvons des lors étudier l'impact d'une combinaison d'opérateurs par rapport aux fonctions qu'ils peuvent réaliser. Dans cette optique, nous pouvons implémenter une FFT réalisée à partir d'un opérateur CORDIC, lui même construit à partir d'un opérateur R-LFSR CORDIC. En effet, par rapport aux opérateurs FFT/Viterbi et AS^3 que nous proposons, une autre alternative est possible en considérant l'opérateur CORDIC. Considérons

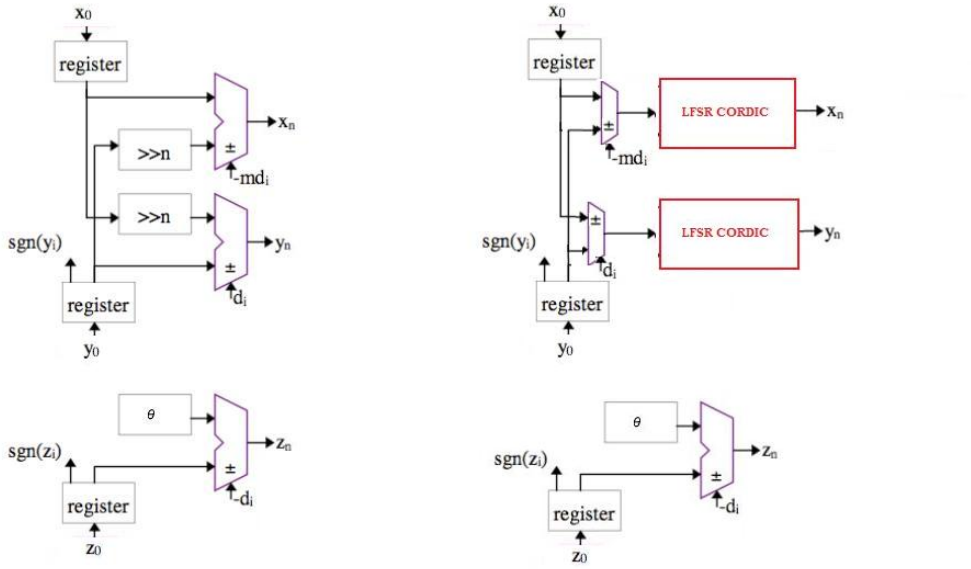

FIGURE H.8 – Schématique d'un étage du module CORDIC avec et sans l'utilisation du R-LFSR CORDIC

l'algorithme de Cooley-Tukey :

$$X(2p) = \sum_{n=0}^{N/2-1} [x(n) + x(n+N/2)].W_{N/2}^{pk} \tag{H.11}$$

$$X(2p+1) = \sum_{n=0}^{N/2-1} ([x(n) - x(n+N/2)].W_{N/2}^{pk}) \cdot W_N^n \tag{H.12}$$

Dans le cadre du projet Européen INFOP, nous développons un module de FFT basé sur l'utilisation d'un module CORDIC. Pour ce faire, nous considérons, les équations précédentes mais ramenées à une division en radix 4 de notre transformée de Fourier rapide (figure H.9). Dans cette configuration, chaque papillon en radix 4 du treillis se décomposera en une succession d'additions/soustractions des quatre entrées suivie par une multiplication de facteur de type $e^{\frac{-2.i.\pi.k}{N}}$.

$$\begin{aligned} X(4k) &= \sum_{n=0}^{N/4-1} [x(n) + x(n+\frac{N}{4}) + x(n+N/2) \\ &\quad + x(n+\frac{3N}{4})]W_N^0 W_{N/4}^{kn} \end{aligned}$$

$$\begin{aligned} X(4k+1) &= \sum_{n=0}^{N/4-1} [x(n) - x(n+\frac{N}{4}) + x(n+N/2) \\ &\quad + jx(n+\frac{3N}{4})]W_N^n W_{N/4}^{kn} \end{aligned}$$

$$\begin{aligned} X(4k+2) &= \sum_{n=0}^{N/4-1} [x(n) - x(n+\frac{N}{4}) + x(n+N/2) \\ &\quad - x(n+\frac{3N}{4})]W_N^{2n} W_{N/4}^{kn} \end{aligned}$$

$$\begin{aligned} X(4k+3) &= \sum_{n=0}^{N/4-1} [x(n) - jx(n+\frac{N}{4}) - x(n+N/2) \\ &\quad - jx(n+\frac{3N}{4})]W_N^{3n} W_{N/4}^{kn} \end{aligned}$$

Avec k = 0, 1, ... , N/4-1.

Habituellement effectué en deux temps par le calcul de $e^{\frac{-2.i.\pi.k}{N}}$, suivi d'une multiplication, en utilisant les explications précédentes sur l'algorithme CORDIC, nous pouvons

opérer directement ce calcul par une rotation de phase, mettant en oeuvre un CORDIC en mode rotation d'angle $\frac{-2.\pi.k}{N}$. Ainsi, pour chacune des quatre sorties d'un papillon en radix 4, de la FFT, nous aurons besoin de trois additionneurs/soustracteurs et d'un module de CORDIC. En reprenant, le système de treillis présenté en chapitre III, pour une taille de FFT de N échantillons (N=2^m), alors le treillis présentera m-3 étages et chaque étage nécessitera $\frac{N}{4}.log_4(N)$ module de CORDIC.

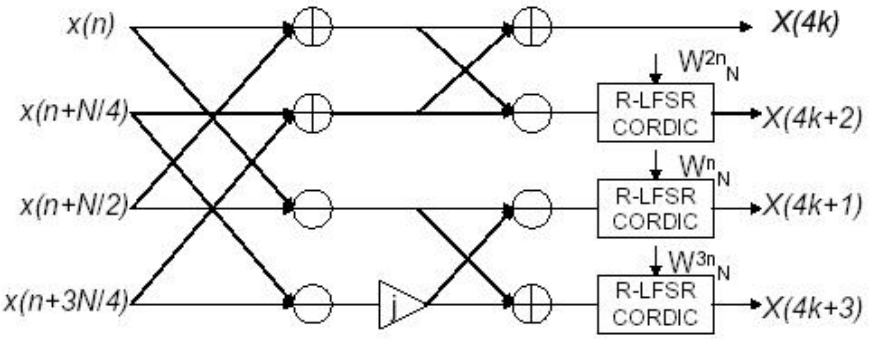

FIGURE H.9 – FFT réalisée par un papillon en CORDIC en RADIX 4

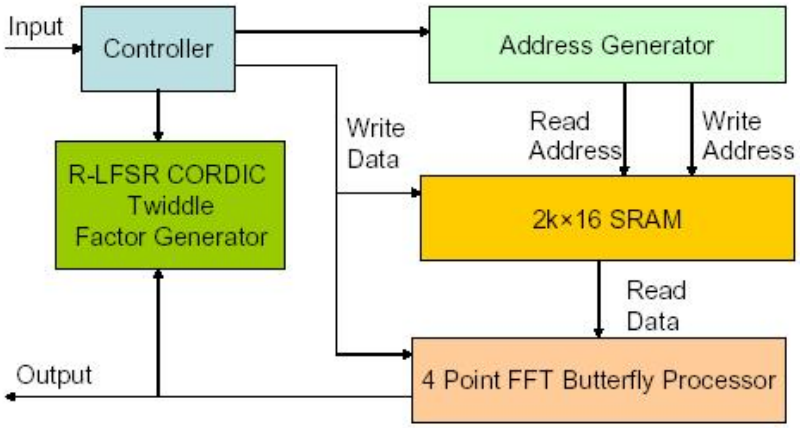

FIGURE H.10 – Architecture d'un processeur FFT 1024

Dans le cadre du projet ANR INFOP, nous réalisons l'implémentation sur une cible Virtex-4 4vlx160 de Xilinx, d'un module FFT 1024, utilisant un SEUL module CORDIC physique, à 12 étages présenté précédemment, dans deux cas, avec ou sans utilisation du R-LFSR CORDIC (figure H.10).

Type de CORDIC :	Classique	R-LFSR - CORDIC
Nombre de Slices :	795	1979
Nombre de Registres :	1287	1874
Fréquence Maximale utilisable :	345 MHZ	289 MHz

TABLE H.2 – Résultats d'Implémentations d'un module FFT 1024 utilisant un module CORDIC 12 étages (Avec ou sans le R-LFSR CORDIC de taille 16)

Glossaire

2G	Mobile de 2nd Génération
3G	Mobile de 3e Génération
3G LTE	3rd Generation Partership Project Long Term Evolution
4G	Mobile de 4e Génération
AM	Amplitude Modulation
AMPS	Advanced Mobile Phone System
ANFR	Agence Nationale des Fréquences
API	Application Programming Interface
ARCEP	Autorité de Régulation des Communications Électroniques et des Postes
ARM	Advanced RISC Machine
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction set Processor
ASSP	Application-Specific Standard Product
ATSC	Advanced Television Systems Committee
BCH	Broadcas Channel
BOC	Banc d'Opérateurs Communs
BPSK	Binary Phase Shift Keying
BTS	Base Transceiver Station
CAN	Convertisseur Analogique Numérique
CCITT	Comité Consultatif International Téléphonique et Télégraphique
CCM	Corba Component Model
CDMA	Code Division Multiple Access
CEPT	Conférence Européenne des Postes et Télécommunications
CIM	Computation Independent Model
CLB	Configurable Logic Bloc
CMA	Constant Module Algorithm
CNA	Convertisseur Numérique Analogique
CORBA	Common Object Request Broker Architecture
CORDIC	Coordinate Rotation Digital Computer
CPCH	Common Packet Channel
CR	Cognitive Radio
CRC	Cyclic Redundancy Check
CRM	Cognitive Radio Management
CRMU	Cognitive Radio Management Unit

CSA	Conseil Supérieur de l'Audiovisuel
DARPA	Defense Advanced Research Projects Agency
DCH	Dedicated Channel
DECT	Digital Enhanced Cordless Telephone
DFE	Dicision Feedback Equalizer
DFT	Discrete Fourier Transform
DSCH	Dedicated Shared Channel
WIFI	Wireless Fidelity ou IEEE802.11b
DMA	Direct Memory Access
DSA	Dynamic Spectrum Access
DSP	Digital Signal Processor
DSSS	Direct Spread Spectrum Sequence
DTMB	Digital Terrestrial Multimedia Broadcast
DVB	Digital Video Broadcasting
EDGE	Enhanced Data rates for GSM Evolution
EQM	Erreur Quadratique Moyenne ou en anglais Mean Square Error MSE
ERP	Extended Rate PHY
FACH	Forward Access Channel
FC	Fonction Commune
FCC	Federal Communications Commission
FCS	Frame Check Sequence
FDD	Frequency Division Duplexing
FDMA	Frequency Division Mutliple Acces
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FHSS	Frequency Hopping Spread Spectrum
FI	Fréquence intermédiaire
FIR	Finite Impulse Response
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
GALS	Globally Asynchronous Locally Synchronous
GDFE	Generalised Feedback Decision Equalizer
GIPS	Giga Instruction per second
GPP	General Purpose Processor
GPRS	General Packet Radio Service
GPS	Global Positionning System
GPU	Graphics Processing Unit
GSM	Global System for Mobile communications
HDCRAM	Hierarchical and Distributed Cognitive Radio Management
HDREM	Hierarchical and Distributed Reconfiguration Management
HEC	Header Extension Code
HF	High Frequency
HSDPA	High Speed Downlink Package Access
ICA	Independent Component Analysis

IF	Intermediate Frequency ou Fréquence Intermédiaire
INTERRAP	Integration of Reactive Behavior and Rational Planning
IOB	Input Output Bloc
IP	Intellect Property
IS136	Interim Standard 136, Version de la norme TDMA
ISDB	Integrated Services Digital Broadcasting
ISI	Intersymbol Interference
JTRS	Joint Tactical Radio System
LUT	Look Up Table
MAQ	Modulation d'Amplitude en Quadrature
MC-CDMA	Multi-Carriers Code Division Multiple Access
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MDP	Modulation par Déplacement de Phase
MEMS	Micro Electro Mechanical Systems
MENHIR	Multi-standard ENHanced Interoperable Radio
MIMO	Multiple-Input Multiple-Output
MIPS	Million Instruction per Seconde
MISO	Multiple Input Single Output
ML	Maximum d'Likelehood
MMSE	Minimum Mean Square Error
MUK	MUltiuser Kurtosis maximization
NOC	Network on Chip
PPU	Physics Processing Unit
OC	Opérateur Commun
OCL	Object Constraint Language
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiplexing Access
OMG	Object Management Group
OS	Operating System
OSIC	Ordered Successif Interference Cancellation
O-STBC	Orthogonal Space-Time Block Coding
P-HAL	Physical-Hardware Abstraction Layer
PCH	Paging Channel
PCI	Peripheral Component Interconnect
PIM	Platform Independant Model
PLCP	Physical Layer Convergence Procedure
PMCS	Programmable Modular Communications System
PMR	Private Mobile Radiocommunications
PSM	Platform Specific Model
QAM	Quadrature Amplitude Modulation
QOS	Quality of Service
QPSK	Quaternary Phase Shift Keying
RACH	Random Access Channel

RC	Radio Cognitive
REM	Reconfiguration Management
REMU	Reconfiguration Management Unit
RF	Radiofréquence
RFI	Request For Information
RKRL	Radio Knowledge Representation Language
RL	Radio Logicielle
RLR	Radio Logicielle Restreinte
RR	Règlement des Radiocommunications
RS	Reed-Solomon
RSB	Rapport Signal à Bruit
RTOS	Real Time Operating System
SC	Single Carrier
SCa	Single Carrier Access
SCA	Software Communication Architecture
SDR	Software Defined Radio
SIC	Successif Interference Cancellation
SIMO	Single Input Multiple Output
SISO	Single Input Single Output
SNR	Signal Noise Ratio
SoPC	System on Programmable Chip
SSB	Single Sideband Modulation
SSC	Shared Spectrum Company
STBC	Space-Time Block Coding
STC	Space-Time Coding
STTC	Space-Time Trellis Coding
SVD	Singular Value Decomposition
SWR	SoftWare Radio
TDD	Time Division Duplexing
TDMA	Time Division Mutliple Access
TEB	Taux d'Erreur Binaire
TNS	Traitement Numerique du Signal
UIT	Union internationale des télécommunications
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications
UTRA	Universal Terrestrial Radio Access
V-BLAST	Vertical Bell Labs Layered Space-Time)
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VHF	Very High Frequency
WEP	Wired Equivalent Privacy
WIFI	Wireless Fidelity ou IEEE802.11b
WIMAX	Worldwide Interoperability For Microwave Access
WLAN	Wireless Local Area Networks
ZF	Zero Forcing

Bibliographie

[ADM82] H Ahmed, J.M Delosme, and M Morf. Highly concurrent computing structures for matrix arithmetic and signal processing. In *IEEE Computer Magazine*, 1982.

[AGLP06] Ali Al Ghouwayel, Yves Louet, and Jacques Palicot. A reconfigurable architecture for the fft operator in a software radio context. In *IEEE International Symposium on Circuits and Systems*, 2006.

[Ahm85] H Ahmed. *Alternative Arithmetic Unit Architectures for VLSI Digital Signal Processors in VLSI and moder signal processing.* Prentice Hall, 1985.

[ALMA81] H.M Ahmed, D.T Lee, M Morf, and P.H Ang. A vlsi speech analysis chip set based on square root normalized ladder forms. In *ICCASSP'81*, 1981.

[ALR+09] L Alaus, Y Louet, C Rolland, J Palicot, and D Noguet. The common operator technique : A promising method for sdr terminal. In *VLSI*, 2009.

[ANJ+04] D Andrews, D Niehaus, R Jidin, M Finley, W Peck, M Frisbie, J Ortiz, K ED, and P Ashenden. Programming models for hybrid fpga-cpu computanional components : a missing link. In *IEEE MICRO*, 2004.

[AQD09] A-R Al-Qawasmi and O Daoud. An ofdm free access technique using gold sequence. In *European Journal of Scientific Research*, 2009.

[Arm24] E.H. Armstrong. The super-heterodyne its origin, development, and some recent improvements. In *In Proceddings of the IRE*, 1924.

[BC99] Claude Berrou and Douillard Catherine. Multi-parrallel concatenation of circular recursive systematic convolutional codes. In *Annales des Télécommunications 54, num 3-4*, 1999.

[BEM+03] V Baumgarte, G Ehlers, F May, G Nuckel, M Vorbach, and M Weinhardt. Pact xpp - a self-reconfigurable data processing architecture. In *The Journal of Supercomputing*, 2003.

[BMB07] I Belaid, F Muller, and M Benjemaa. Off-line placement of hardware tasks on fpga. In *11th International Conference on Compouter Supported Cooperative Work in Design*, 2007.

[BN04] L Biard and D Noguet. An adaptable architecture for the viterbi algorithm. In *International Symposium on Wireless Personal Multimedia Communications*, 2004.

[BP95] K Berberidis and J Palicot. A frequency domain decision feedback equalizer for multipath echo cancellation. In *Globecom'95*, 1995.

[BP96] K Berberidis and J Palicot. A block quasi-newton algorithm implemented in the frequency domain. In *EUSIPCO'96*, 1996.

[BSE04] A.R.S Bahai, B.R Saltzberg, and M Ergen. *Multi Carrier Digital Communications - Theory and Applications of OFDM.* New York Springer, 2004.

[BSL+04] N Bruels, E Sicheneder, M Loew, A Schackow, J Gliese, and C Sauer. A 2.8 gb/s, 32 state, radix-4 viterbi decoder add-compare-select unit. In *Symposium on VLSI Circuits*, 2004.

[CL87] J.R. Cavallaro and F.T Luk. Cordic arithmetic for an svd processor. In *Proceedings of the 8th Symposium on Computer Arithmetic*, 1987.

[Com03] Federal Communication Commission. Notice of rulemaking on cognitive radio. Technical report, Federal Communication Commission, 2003.

[CT65] J.W. Cooley and J.W. Tukey. *An Algorithm for the Machine Calculation of Complex Fourier Series.* American Mathematical Society, 1965.

[DCPS02] R David, D Chillet, S Pillement, and O Sentieys. Dart : a dynamically reconfigurable architecture dealing with future mobile telecommunications constraints. In *Parallel and Distributed Processing Symposium*, 2002.

[DDN85] P Dewilde, E Deprettere, and R Nouta. *Parallel and pipelined VLSI implementation of signal processing algorithms in VLSI and modern signal processing.* Prentice Hall, 1985.

[Dep83] E Deprettere. Synthesis and fixed-point implementaion of pipelined true orthogonal filters. In *ICASSP'83*, 1983.

[Des74] A.M Despain. Fourier transform computers using cordic iterations. In *IEEE Transactions on Computers*, 1974.

[Des79] A.M Despain. Very fast fourier transform algorithms hardware for implementations. In *IEEE Transactions on Computers*, 1979.

[ELM04] D Ercegovac, Thomas Lang, and R Modiri. Implementation of fast radix-4 division with operands scaling. In *Computer Science*, 2004.

[ETS00] ETSI. Dvb, interaction channel for satellite distribution systems. In *ETSI EN 301 790, V1.2.2, pp. 21-24*, 2000.

[FCG95] E.R Ferrara, C.F.N COWAM, and P.M Grant. Frequency domain adaptive filtering. In *Prince-Hall*, 1995.

[GAP+10] S Gul, L Alaus, J Palicot, C Moy, and N Noguet. Linear feedback shift registers as common operators in cognitive radio. In *International Journal of Autonomous and Adaptive Communications*, 2010.

[GK02] Mark Goresky and Andrew M. Klapper. Fibonacci and galois representations of feedback-with-carry shift registers. In *IEEE Transactions on Information Theory, Vol. 48, No. 11*, 2002.

[GMP07] S.T GUL, C Moy, and J Palicot. Two scenarios of flexible multi standard architecture designs using a multi-granularity exploration. In *PIMRC'O7*, 2007.

[Gol82] S.W. Golomb. *Shift Register Sequences*. Aegean Park Press, 1982.

[GSS+02] D Greifendorf, J Stammen, S Sappok, M Ackeren Van, and P Jung. A novel hardware design paradigm for mobile "software defined radio" terminals. In *In IEEE Seventh International Symposium on Spread Spectrum Techniques and Applications*, 2002.

[Had95] J.D. Hadley. Design methodologies for partially reconfigured systems. In *IEEE Symposium on FPGA's for Custom Computing Machines*, 1995.

[HC98] M.E. Hamid and C.I.H. CHEN. A note to low-power linear feedback shift registers. In *IEEE Transactions on Circuits and Systems II : Analog and Digital Signal Processing*, 1998.

[Hen02] T Hentschel. Channelization for software defined base station. In *Annals of Telecom, 57, vol 5-6*, 2002.

[HW97] JR Hauser and J Wawrzynek. Garp : A mips processor with a reconfigurable coprocessor. In *IEEE Symposium on FPGAs for Custom Computing Machines*, 1997.

[Ins98] Texas Instrument. Tms320c5x user's guide (rev. d). In *Texas Instrument Technical documents, spru056d ed.*, 1998.

[JF07] S.G Johnson and M Frigo. A modified split-radix fft with fewer arithmetic operations. In *IEEE Transaction Signal Processing 55*, 2007.

[JHF86] I-Chang Jou, YU-Hen Hu, and W.S Feeg. A novel implementation of pipelined toeplitz system solver. In *Processing of the IEEE*, 1986.

[Jon02] F Jondral. *Book Title : Software Defined Radio Enabling Technologie (by Walter Tuttlebee)*, chapter Parametrization - A technique for SDR Implementation. Wiley, 2002.

[Kai85a] T Kailath. *Signal Processing in the VLSI Era in VLSI and moder signal processing*. Prentice Hall, 1985.

[KAI85b] T KAILATHWALTHER. Signal processing in the vlsi era. In *VLSI and Modern Signal Processing*, 1985.

[Knu74] Donald Knuth. Structured programming with go to statements. In *Cumputing Surveys, Vol 6, num 4*, 1974.

[Koe90] John Koeter. *What's a LFSR ?* Texas Instruments Incorporated, 1990.

[Lav96] Pierre Lavoie. A high-speed cmos implementation of the winograd fourier transform algorithm. In *IEEE Transactions on Signals Processings, Voll 44, num 8*, 1996.

[LDP05] P Leray, J.P. Delaye, and J Palicot. A hierarchical modeling approach in software defined radio system design. In *SIPS*, 2005.

[LNP+02] P Loumeau, J.F. Naviner, H Petit, L Naviner, and P Desgreys. Analog to digital conversion : technical aspects. In *In Annales des télécommunications, volume 57*, 2002.

[Low96] M Lowy. Parallel implementation of lfsr for low power application. In *IEEE Trans. on Circuit and System*, 1996.

[LS05] D.J Lilja and S.S Sapatnekar. *Designing Digital Computer Systems with Verilog*. Cambridge University Press, 2005.

[LTC+03] A Lodi, M Toma, F Campi, A Cappelli, R Canegallo, and R Guerrieri. A vliw processor with reconfigurable instruction set for embedded applications. In *IEEE Journal of Solid-State Circuits*, 2003.

[LTE] *IEEE Std 802.11g-2003*.

[LZWP07] Liang Liang, Xue-Gong Zho, Ying Wang, and Cheng-Lian Peng. Online hybrid task scheduling in reconfigurable systems. In *11th International Conference on Compouter Supported Cooperative Work in Design*, 2007.

[Man01] StarCore 140 DSP Core Reference Manual. Starcore 140 dsp core reference manual. In *Reference Manual, Rev. 3 ed*, 2001.

[MBE+05] C Metelis, P Bougas, G Economakas, P Kalivas, and K Petmestzi. High-speed pipeline implementation of radix-2 dif algorithm. In *Proceedings of World Academy of Science, Engineering and Technology Volume 2*, 2005.

[MG82] D Mansour and A.H Gray. Unconstrained frequency domain adaptive filtering. In *IEEE transaction of ASAP*, 1982.

[Mit00] Joseph Mitola. *Cognitive Radio : An integrated Agent Architecture For Software Defined Radio*. PhD thesis, Royal Institute of Technology of Stockholm, 2000.

[MMAD82] M Morf, C.H Muravhchik, P.H Ang, and J.M Delosme. Fast cholesky algorithms and adaptative feedback filters. In *ICASSP'82*, 1982.

[MPRG06] C Moy, J Palicot, V Rodriguez, and D Giri. Optimal determination of common operators for multi standard software defined radio. In *4th Karlsruhe Workshop on Software Radios*, 2006.

[NCC05] S Nandagopalan, C Cordeiro, and K Challapalli. Spectrum agile radios : Utilization and sensing architectures. In *IEEE Dynamic Spectrum Access Networks*, 2005.

[Nev77] Robert L. Nevin. Application of the rader-brenner fft algorithm to number-theoretic transforms. In *IEEE Transactions on Acoustics, Speech and Signal*, 1977.

[Nog04] D Noguet. A reconfigurable systolic architecture for umts/tdd joint detection real time computation. In *IEEE 8th International Symposium on Spread Sprectrum Techniques and Applications*, 2004.

[OMW+08] H Otrok, N Mohammed, L Wang, M Debbabi, and P Bhattacharya. A game-theoretic intrusion detection model for mobile ad hoc networks. In *IEEE Comput Commun*, 2008.

[Pop98] A Pope. The corba reference guide : Understanding the common object request broker architecture. 1998.

[PR02] J Palicot and C Rolland. A self adaptive universal receiver. In *Annals of Telecom, 57, vol 5-6*, 2002.

[PR03] J Palicot and C Roland. Fft : a basic function for a reconfigurable receiver. In *ICT'2003*, 2003.

[Pro07] John Proakis. *Digital Communications*. McGraw-Hill Higher Education, 2007.

[Ras02] R Rasheed. Reconfigurable viterbi decoder fopr mobile platform. In *Mobile Communications Departments, Eurocom*, 2002.

[RBR63] L.P.A Robichard, M Boisvert, and J Robert. Graphes de fluences, application à l'électrotechnique et à l'électronique, calculateurs analogiques et digitaux. In *Annales des Télécommunications*, 1963.

[Ren87] M Renaudin. Architecture d'un opérateur cordic. In *Rapport de Stage, CNET-Grenoble, CEPHAG*, 1987.

[Rhi02] Arnd-Ragnar Rhiemeier. Benefits and limits of parameterized channel coding for software radio. In *2nd Karlsruhe Workshop on Software Radios*, 2002.

[RK84] S.K Rao and T Kailath. Orthogonal digital filters for vlsi implementaiton. In *IEEE Transactions on Circuits and Systems, CAS-31*, 1984.

[Sem95] National Semiconductor. *PC16550D Universal Asynchronous Receiber/Transmitter with FIFOs*. National Semiconductor, 1995.

[SF84] L.H Sibul and A.L Fogelsaner. Application of coordinate rotation algorithm to singular value decomposition. In *Proceedings of ISCAS'84*, 1984.

[SH86] Tza-Yun Sung and Yu-Hen Hu. Vlsi implementaion of real-time kalman filter. In *ICCASSP'86*, 1986.

[SLL+00] H Singh, M Lee, G Lu, F Kurdahi, N Bagherzadeh, and E Chaves Filho. Morphosys : an integrated reconfigurable system for data-parallel and computationintensive applications. In *Transactions on Computers*, 2000.

[SRD96] G Slavenburg, S Rathnam, and H. Dijkstra. The trimedia tm-1 pci vliw media processor. In *IEEE CS Press,Los Alamitos, Californie*, 1996.

[STGB02] S Swaminathan, R Tessier, D Goeckel, and W Burleson. A dynamically reconfigurable adaptive viterbi decoder. In *Department of Electronical and Computer Engineering, University of Massachusetts*, 2002.

[Tut02] Walter Tuttlebee. *Software Defined Radio Enabling Technologies*. Wiley, 2002.

[Vay02] M Vaya. Viturbo : A reconfigurable architecture for ubiquitous wireless networks. In *M.S. Thesis, Rice University, Houston, Tx*, 2002.

[Vol59] J.E Volde. The cordic trigonometric computing technique. In *Transactions on Electronic Computers, EC-8*, 1959.

[VPGLDM94] J Van Praet, G Goossens, D Lanneer, and H De Man. Instruction set definition and instruction selection for asips. In *Proceedings of the Seventh International Symposium on High-Level Synthesis*, 1994.

[Wal71] J.S. Walther. A unified algorithm for elementary functions. In *Proceedings Joint Spring Computer Conference*, 1971.

[WDLP07] H Wang, J.P. Delaye, P Leray, and J Palicot. Managing dynamic reconfiguration on mimo decoder. In *IPDPS*, 2007.

[WH01] S.Y Wang and C.C Hunag. On the architecture and performance of an fft-based spread spectrum downlink rake receiver. In *IEEE Transaction - Vehicular Technology*, 2001.

[WIF] *IEEE Std 802.11b-1999/Cor 1-2001.*

[WIM] *IEEE Std 802.16 - Part 16.*

[WLP06] H Wang, P Leray, and J Palicot. Reconfigurable architecture for mimo systems based on cordic operators. In *Comptes Rendus Physique, Elsevier, volume 7 septembre 2006, pp 735-738*, 2006.

[ZWZ95] Meng Chu Zhou, Chi-Hsu Wang, and Xiaoyong Zhao. Automating mason's rule and its application to analysis of stochastic petri nets. In *IEEE Transactions on Control Systems Technology*, 1995.

Table des figures

Résumé

Dans un contexte de multiplication des normes de télécommunications aux spécifications distinctes, nos travaux de recherche définissent une architecture reconfigurable pour des terminaux radio multistandard. Nous proposons ainsi une nouvelle méthode de paramétrisation intitulée "la Technique des Opérateurs Communs" afin d'accéder à la combinaison " reconfiguration temps réel/généricité du terminal". Cette technique est une méthodologie de conception et d'exécution qui vise à développer un ensemble limité d'"Opérateurs Communs" (OC), génériques, reconfigurables par simple téléchargement de paramètres, indépendants de la cible technologique d'implémentation et qui constituent les éléments de base de notre terminal multistandard. Afin d'exécuter l'ensemble du terminal multistandard, trois familles d'OC sont proposées (LFSR, FFT/VITERBI, CORDIC). Une méthodologie de gestion des OC est également présentée et une étude de différents cas d'implémentations est menée, permettant d'atteindre des gains en complexité jusqu'à 40% de la complexité matérielle sur une cible FPGA dans le cas d'un terminal tri-standard (IEEE 802.11, IEEE802.16 et 3GPP LTE).

Abstract

In the present day, the profusion of wireless communication standards leads to complex terminals able to manage a wide range of standards, which calls for multistandard terminals. In order to meet the requirement of such terminals, we propose a new Parameterization strategy to design a Reconfigurable Terminal. With this method, - The Common Operator Technique - Parameterization focuses on smaller building blocks that can be reused across many of the functions required by each standard. The Method leads up to higher scalability and reconfigurability at the expense of an extra scheduling to handle with. As a consequence, we propose a new architecture in Common Operator Bank (COB), which limits the scheduling issue though optimizing the hardware complexity. Three families of Common Operators are introduced, (LFSR, Treillis/Butterflies, CORDIC). The first realizations obtained in COB are based on LFSR and CORDIC operators. Centered on a tri-standard terminal (3GPP LTE, IEEE802.11g and 802.16e), the implementation on a FPGA, Altera/Cyclone II results in a Logic Cells complexity decrease of 40%.