THESE

présentée pour obtenir le titre de Docteur de l'Ecole Nationale Supérieure des Télécommunications

Spécialité : Electronique et Communications

Olivier Gay-Bellile

Architecture Programmable pour le Décodage Canal

Soutenue le 28 Avril 1999 devant le jury composé de

Glenn Gulak	Président
Eric Martin Roger Reynaud	Rapporteurs
Emmanuel Boutillon Eric Dujardin Jean Gobert	Examinateurs

Remerciements

Je tiens à exprimer tout ma reconnaissance à Nicolas Darbel, Emmanuel Boutillon et Eric Dujardin pour avoir encadré cette thèse.

Je remercie également Glenn Gulak, Professeur à l'université de Toronto, de m'avoir fait l'honneur de présider ce jury.

J'adresse mes remerciements à Roger Reynaud, Professeur à l'université d'Orsay, ainsi qu'à Eric Martin, Professeur à l'université de Rennes, pour avoir accepté d'être rapporteurs et à Jean Gobert pour s'être intéressé à mon travail.

Je tiens également à remercier Nicolas Demassieux, Responsable du département Électronique de l'ENST et Mohammad Haghiri, Responsable de la division Videocommunications du LEP de m'avoir accueilli dans leur laboratoires.

Un grand merci à tous les correcteurs de ce manuscrit et tout particulièrement mon frère Christophe, Emmanuel Boutillon et Fabrice Franceschi pour leurs nombreuses correction malgrè la distance, ainsi qu'à toutes les personnes qui sont venues me soutenir le jour de ma soutenance et surtout ma mère, Loic et Myriam.

Je souhaite enfin remercié tous les membres du départment Électronique et tous ceux de la division Videocommunications avec lesquels j'ai eu le plaisir de travailler et en particulier Nicolas, Emmanuel, Christophe, Mahieddine, Eric, Xavier et Julien.

Résumé

L'évolution conjointe de l'intégration des circuits et des protocoles de communications numériques permet maintenant d'envisager la création d'un circuit "décodeur canal" unique pour toute une classe de démodulations numériques. Un tel composant permettrait, grâce à sa versatilité, de répondre aux besoins de multiples applications. Il pourrait donc bénéficier de la réduction de coût de fabrication liée à la production en grande série.

Afin de définir les besoins d'un tel récepteur, les algorithmes de décodage canal liés à plusieurs supports de communications (câble, satellite et terrestre) pour l'application "télévision numérique" ont été étudiés. Cette étude nous a conduit naturellement à proposer une réduction des primitives de traitement à un petit nombre de fonctionnalités fondées sur le filtrage.

La proposition résultante est une architecture parallèle hétérogène pour les communications numériques dénommée "Parallel-Digital Communication Processor" (P-DCP) dont l'élément central est une unité de calcul dédiée au filtrage numérique appelée "Communication Co-Processor" (CCP). Cette architecture parallèle s'appuie également sur un réseau d'interconnexion statique correspondant aux contraintes du décodage canal (faible latence, haut débit, localité des communications) et intégrant un mécanisme permettant de gérer les communications dynamiques requises par la fréquence variable des symboles.

Cette architecture permet d'envisager la généricité et la modularité du décodage canal multi-standard. Elle doit permettre d'accélérer la conception d'un produit supportant plusieurs applications. L'étape suivante sera alors la conception d'un "modem universel" servant d'interface entre n'importe quel terminal de communication (télévision, PC, console de jeux, téléphone) et le support de communication (câble, antenne satellite, ligne téléphonique, ligne électrique).

Abstract

Both the evolution of IC integration and digital communication protocols make it possible to think of the design of a "channel decoder" unique for a whole class of digital demodulation. Such a component, owing to its versatility, would allow to cope with the requirements of multiple applications. Thus, it could benefit from the reduction of the production cost linked to the mass production.

In order to define the requirements of such a receiver, channel decoding algorithms corresponding to several communication links (cable, satellite and terrestrial) for the "digital television" have been studied. This study leads naturally to a reduction of basic functions to filtering.

The resulting proposal is an heterogeneous parallel generic architecture for digital communication called "Parallel-Digital Communication Processor" (P-DCP), which basic component is an execution unit, called "Communication Co-Processor" (CCP), optimized for filtering operations. This parallel architecture is also based on a static interconnection network fully compliant with channel decoding constraints (low latency, high data rate, local communications) integrating a mechanism that copes with the dynamic communications required by the variable symbol rate.

This architecture allows to consider the genericity and the modularity of a multi-standard channel decoder. It should help accelerating the design of multiple purpose products. Then, the next step will be the conception of a "universal modem" used as an interface between any communication terminal (TV, PC, play station, telephone) and any communication link (cable, satellite, phone line, electrical line).

Table des matières

Liste des figures **vii**

Liste des tableaux **xi**

Introduction **1**

I Caractérisation des opérations de démodulation numérique **5**

I.1 Description algorithmique globale des applications 5

I.1.1 Démodulation de la transmission par satellite 6

I.1.2 Démodulation de la transmission par câble 6

I.1.3 Démodulation de la transmission par voie hertzienne 7

I.2 Le filtrage 9

I.2.1 Filtrage anti-repliement 9

I.2.2 Filtrage de l'interpolateur 10

I.2.3 Filtre de Nyquist 11

I.2.4 Égaliseur 11

I.2.5 Transposition en Bande de base 12

I.2.6 Interpolation fréquentielle et temporelle 12

I.3 Les boucles de contre-réaction 13

I.3.1 Les boucles à verrouillage de phase numériques 14

I.3.1.a Les détecteurs 14
I.3.1.b Le filtre de boucle 15
I.3.1.c Les oscillateurs contrôlés numériquement 15
I.3.1.d La correction 15
I.3.2 Les fonctions d'autocorrélation 16
I.3.3 Le Contrôle Automatique de Gain (CAG) 17
I.4 La correction d'erreur 17
I.4.1 Le décodeur de Viterbi 17
I.4.2 Le décodeur de Reed-Solomon 18
I.5 Caractéristiques des communications numériques 19
I.5.1 Répartition de la puissance de calcul par application 19
I.5.2 Comparaison des besoins en puissance de calcul 20
I.5.3 Parallélisme de sous-mots 21
I.5.4 Analyse des besoins en mémoire 22
I.6 Conclusion 22

II État de l'art des architectures pour les communications numériques 25

II.1 Les architectures câblées 25
II.2 Les processeurs programmables 26
II.3 Les architectures adaptées 27
II.4 Les circuits programmables de filtrage 27
II.4.1 Circuits configurables pour le filtrage 28
II.4.2 Circuits de filtrage à base de FPGA 28
II.4.3 Processeurs de filtrage 29
II.4.4 Caractéristiques des circuits programmables de filtrage 29

III Une solution aux problèmes de filtrage : le CCP **33**

III.1 Spécifications du “Communication Co-Processor” 33

III.1.1 Fonctions requises 33

III.1.1.a Filtres RIF standards 34

III.1.1.b Filtres symétriques 34

III.1.1.c Filtres antisymétriques 35

III.1.1.d Filtres adaptatifs 35

III.1.1.e Filtres avec décimation 36

III.1.1.f Équation générale d’un filtre 36

III.1.1.g Multiplication complexe 36

III.1.1.h Transformée de Fourier 37

III.1.2 Fonctionnalités requises 37

III.1.2.a Multiplexage de filtres 37

III.1.2.b Calcul itératif 37

III.1.2.c Cascadabilité 38

III.1.2.d Parallélisme de sous-mots 38

III.1.2.e Type des données 38

III.2 Conception de l’architecture du CCP 38

III.2.1 Notations 39

III.2.2 La partie opérative 41

III.2.2.a Un opérateur de filtrage 41

III.2.2.b Multiplication complexe 46

III.2.2.c Transformée de Fourier 47

III.2.2.d Structure complète de la partie opérative 48

III.2.2.e Mise à jour des coefficients 49

III.2.3 La mémoire 49

III.2.3.a Mémoire des résultats 50
III.2.3.b Nombres de ports d'entrée/sortie 50
III.2.3.c Architecture mémoire 51
III.2.4 Réorganisation des données 53
III.2.5 Le contrôleur 56
III.2.5.a Machine à états pour la réalisation d'un filtre 57
III.2.5.b Influence de la décimation 58
III.2.5.c Influence du parallélisme de sous-mots 60
III.2.5.d Influence du multiplexage de filtres 61
III.3 Plateforme de validation 63
III.4 Estimation de la complexité du CCP 65
III.5 Conclusion 67

IV Présentation de l'architecture P-DCP 69

IV.1 Conception de l'architecture 69
IV.1.1 Description du P-DCP 70
IV.1.2 Choix des blocs de calcul 71
IV.1.3 Le réseau d'interconnexion 72
IV.1.4 Mise en œuvre de la fréquence de travail 73
IV.2 Modélisation des communications 75
IV.2.1 Mise en œuvre du drapeau "valide" 76
IV.2.1.a Le drapeau valide et les multiplexeurs 76
IV.2.1.b Le drapeau valide et les unités de calcul 77
IV.2.1.c Le drapeau valide et la décimation 77
IV.2.2 Des communications à double vitesse 78
IV.3 Modélisation de la complexité matérielle 80
IV.3.1 Estimation de la complexité matérielle des composants 80

IV.3.1.a Le contrôle automatique de gain 82
IV.3.1.b La récupération d'horloge 82
IV.3.1.c La récupération de porteuse 84
IV.3.1.d L'égaliseur 84
IV.3.2 Le CCP 84
IV.3.2.a Calcul du nombre de CCP nécessaires 84
IV.3.2.b Complexité du CCP 86
IV.3.2.c Réalisation de la FFT 86
IV.3.3 Estimation de la complexité du réseau d'interconnexion 86
IV.3.4 Taille du programme de contrôle 87
IV.3.5 Complexité totale 88
IV.4 Conclusion 89

V Validation et résultats 91

V.1 Méthodologie de validation de l'architecture 92
V.2 La synthèse d'architecture 92
V.3 Modélisation conjointe de l'architecture et de l'application 94
V.3.1 Extraction des caractéristiques de l'architecture 95
V.3.2 Extraction des caractéristiques de l'application 96
V.3.2.a Lancement des calculs et décimation 96
V.3.2.b Générateur 98
V.3.3 Assignation des composants 98
V.4 Phase 1 : Ordonnancement des calculs et des communications 98
V.4.1 Réservation des ressources 101
V.4.2 Résolution des conflits 101
V.4.3 Évolution possible de la notion de conflit 102
V.5 Phase 2 : Simulation comportementale 103

V.5.1 Présentation du fonctionnement général du programme de validation 103

V.5.2 Récupération des résultats 104

V.5.3 Mémorisation des résultats 104

V.5.4 Transfert des données 104

V.5.5 Réalisation des calculs 106

V.6 Résultats 106

V.6.1 Phase 1 : Ordonnancement des calculs et des communications . . 107

V.6.1.a Résolution des conflits 108

V.6.1.b Latence des boucles 109

V.6.2 Phase 2 : Simulation comportementale 110

V.7 Conclusion 111

Conclusion et perspectives **113**

Bibliographie **117**

ANNEXES **124**

A Exemple de modélisation conjointe d'architecture et d'application **125**

A.1 Description de l'architecture 125

A.2 Description de l'application 126

A.3 Description de l'assignation des composants 127

Liste des figures

1 Schéma d'un système de communications numériques 2

I.1 Schéma d'un démodulateur satellite 7

I.2 Schéma d'un démodulateur câble 7

I.3 Structure d'un symbole OFDM 8k 8

I.4 Schéma d'un démodulateur terrestre 9

I.5 Structure récursive du filtrage anti-repliement 10

I.6 Structure de l'interpolateur 11

I.7 Structure de l'égaliseur récursif avec décision dans la boucle 12

I.8 Interpolations temporelle et fréquentielle 13

I.9 Structure d'une boucle à verrouillage de phase numérique 14

I.10 Structure du Contrôle Automatique de Gain 17

I.11 Schéma du décodeur de Reed-Solomon 19

I.12 Puissance de calcul 20

I.13 Puissance de calcul 20

I.14 Puissance de calcul du terrestre 20

I.15 Besoins des applications en fonction de la fréquence symbole 21

I.16 Besoins en mémoire pour la démodulation terrestre 22

III.1 Réalisation de filtres symétriques 35

III.2 Architecture générale du CCP 39

III.3 Graphe de dépendance pour un filtre réel de 4 coefficients sur 16 bits . 40

III.4 Filtre complexe 16 bits 43

III.5 Filtre réel 16 bits 44

III.6 Filtre semi-complexe 16 bits 45

III.7 Filtre réel 8 bits 46

III.8 Multiplication complexe 8 bits 47

III.9 Structure du papillon de radix-2 (Opérations complexes) 48

III.10 Structure finale simplifiée de la partie opérative 48

III.11 Bloc de mise à jour 49

III.12 Influence du type des données et de la décimation sur la lecture des données de la voie aller dans la mémoire 51

III.13 Partitionnement maximal 53

III.14 Partitionnement partiel 53

III.15 organisations des données et des coefficients requises 54

III.16 Graphe de dépendance pour un filtre réel sur 16 bits 55

III.17 Machine à états pour un filtre 58

III.18 Graphe de dépendance pour un filtre complexe avec décimation par 2 . 59

III.19 Machine à états pour un filtre avec décimation par 2 60

III.20 Graphe de dépendance pour un filtre complexe sur 8 bits 61

III.21 Machine à états pour un filtre avec un parallélisme de sous-mots 61

III.22 Exemple de multiplexage de filtre 62

III.23 Décomposition de la latence du CCP 63

III.24 Plateforme de simulation du CCP 65

III.25 Complexité du CCP en fonction de la longueur des filtres supportés . . 66

IV.1 Architecture générique P-DCP 70

IV.2 Réseau d'interconnexion style fonctionnel 72

IV.3 Réseau d'interconnexion complet 73

IV.4 Diagramme de génération de l'horloge 74

IV.5 Mise en œuvre de la génération de l'horloge 74

IV.6 Exemple d'utilisation du drapeau "valide" 76

IV.7 Multiplexeur pour les communications 76

IV.8 Mise en œuvre du drapeau valide dans les composants 77

IV.9 Influence des créneaux de communications et de la décimation sur la conception des composants 78

IV.10 Répartition des créneaux de communications 79

IV.11 Interface haut débit/bas débit 79

IV.12 Architecture P-DCP pour la démodulation DVB 87

V.1 Exemple d'algorithme 95

V.2 Exemple d'architecture 95

V.3 Modélisation des composants de l'architecture 96

V.4 Modélisation des fonctions de l'application 97

V.5 Assignation des composants 99

V.6 Algorithme du programme d'ordonnancement 100

V.7 Algorithme du programme de simulation comportemental 105

V.8 Assignation des fonctions pour la démodulation câble 107

V.9 Assignation des fonctions pour la démodulation satellite 108

Liste des tableaux

I.1 Caractéristiques et besoin des applications 23

I.2 Évolutions technologiques dans l'industrie du semiconducteur [1] 24

II.1 Fonctionnalités des circuits 30

III.1 Répartition des besoins en mémoire des filtres 51

III.2 Complexité des blocs constituant le CCP 66

IV.1 Complexité des opérateurs de base en nombre de portes NAND 81

IV.2 Complexité du bloc de contrôle automatique de gain 82

IV.3 Complexité du détecteur de Gardner 82

IV.4 Complexité du filtre de boucle 83

IV.5 Complexité du NCO 83

IV.6 Complexité du mélangeur de l'interpolateur 83

IV.7 Complexité du bloc de calcul d'aide pour l'égaliseur 84

IV.8 Puissance de calcul requise pour les algorithmes de filtrage 85

IV.9 Complexité du réseau d'interconnexion 87

IV.10 Complexité du P-DCP 88

V.1 Latence des boucles de contre-réaction dans le câble 109

V.2 Latence des boucles de contre-réaction dans le satellite 110

A.1 Fichier de description de l'architecture 125

A.2 Fichier de description des connexions entre les fonctions de l'application 126

A.3 Description des fonctions génératrices 127

A.4 Description de l'assignation des composants 127

INTRODUCTION

Depuis quelques années, la télévision numérique connaît une véritable explosion avec la création de bouquets satellite (Canal Satellite Numérique, TPS, Via Digital, Telepiu, Sky Digital, Direct TV, PrimeStar...), câble (Lyonnaise Câble, Mediaone...) et terrestre (ONDigital en Angleterre). À terme, il est prévu qu'elle remplace complètement la télévision analogique.

Ce développement rapide est dû, d'une part, aux nombreux avantages intrinsèques au format numérique (compression de l'information, qualité constante, interactivité...), d'autre part, au progrès de la technologie, et en particulier, à la possibilité d'intégrer maintenant des fonctions complexes de traitement numérique à bas coût dans des circuits intégrés CMOS.

Les différents media de transmission de la télévision numérique (câble, satellite, hertzien) utilisent les mêmes principes, donc les mêmes types de traitement que l'ensemble des systèmes de communications numériques. Rappelons, dans le cas général, le principe d'une chaîne de transmission numérique *(Figure 1)*.

Dans un premier temps, on effectue la capture des informations provenant d'une source quelconque et sa conversion en un signal numérique grâce au *capteur*. Une représentation binaire idéale de ce signal doit utiliser le moins de bits d'information possible. Une représentation efficace du signal de sortie doit donc comporter un minimum de redondance. Le procédé qui convertit de manière efficace un signal en une séquence binaire est appelé *codage source*.

Le signal ainsi obtenu doit ensuite être transmis par l'intermédiaire d'un medium de communication (câble, satellite, ligne téléphonique, ondes hertziennes...). Ce canal de transmission n'étant pas parfait, des erreurs peuvent s'immiscer dans la transmission, si bien que le signal reçu ne sera pas identique au signal émis. Pour corriger ces erreurs (bruit du canal, interférences, échos...), des bits de redondance sont ajoutés lors du *codage correcteur d'erreurs*. Ensuite, le signal qui est en bande de base (la bande de fréquence utilisée à la fréquence porteuse nulle), devra être transposé en un signal modulé à une fréquence porteuse dépendante du medium de communication. La *modu-*

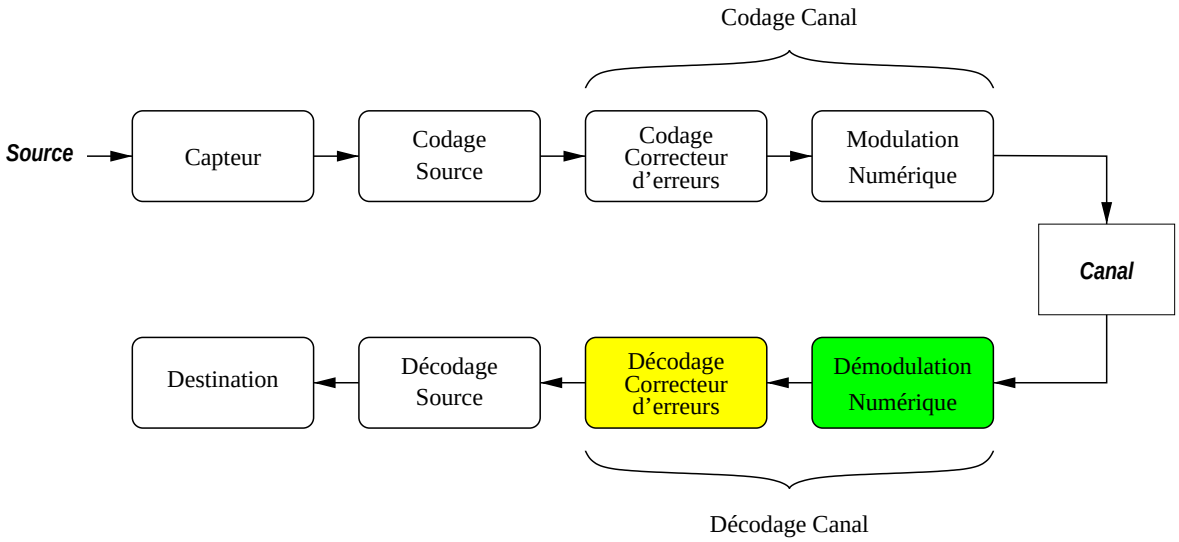


FIGURE 1 : Schéma d'un système de communications numériques

lation exécute cette tâche. L'ensemble de ces opérations, codage correcteur d'erreurs et modulation, forme le *codage canal* puisqu'il s'agit des opérations effectuées en fonction des caractéristiques du canal.

En réception, pour reconstruire le signal émis initialement par la source, il faut effectuer les transformations inverses de celles effectuées à l'émission. Ainsi, le signal reçu par l'intermédiaire du *canal* de transmission est transposé en bande de base et une estimation des symboles émis est effectuée par le *démodulateur*. Les erreurs de transmission sont détectées et corrigées dans la mesure du possible par le *décodage correcteur d'erreurs* grâce aux informations de redondance ajoutées lors du codage correcteur d'erreurs. La reconstruction du signal original est alors effectuée avec le *décodage source*. Enfin, le signal arrive à *destination* où il peut être affiché ou traité grâce à l'appareil adéquat.

Pour la télévision numérique, la puissance de calcul requise par les traitements situés dans le décodeur dépasse la dizaine de milliards d'opérations par seconde (Gops). Or, à partir de quelques milliards d'opérations par seconde, des solutions logicielles utilisant des processeurs de traitement du signal (DSP : Digital Signal Processor) n'ont plus la puissance suffisante pour effectuer les traitements nécessaires. Aussi, jusqu'à présent, un circuit intégré spécifique (ASIC : Application Specific Integrated Circuit) était conçu pour chaque application de manière à atteindre la puissance de calcul requise avec un coût raisonnable pour le marché grand public. Ainsi, chaque nouvelle application, chaque évolution dans les normes ou chaque amélioration entraînait la conception d'un nouveau circuit en mobilisant des ressources importantes.

Pourtant, avec les dernières technologie CMOS (0,25μm ou 0,18μm), il est maintenant envisageable de concevoir un circuit intégré unique pour effectuer le "décodage canal" (démodulation et correction des erreurs) des différents standards de transmission de la télévision numérique. Un tel circuit, qui incluerait des parties matérielles dédiées

et du logiciel offrirait de nombreux avantages :

- une réduction des coûts par facteur d'échelle,
- un gain en surface par la réutilisation du matériel grâce à une fréquence d'horloge plus élevée,
- une facilité accrue pour tester sur le terrain les nouveaux algorithmes grâce à l'utilisation d'un prototype programmable,
- une intégration rapide des nouveaux algorithmes dans de nouveaux produits ("time to market").

L'étude d'un tel circuit représente l'objet de cette thèse qui est effectuée en collaboration avec les Laboratoires d'Electronique Philips. Afin d'obtenir un résultat à moyen terme, nous chercherons à concevoir un décodeur canal multi-standard pour la télévision numérique européenne [2] dont les standards ont été définis par le consortium européen DVB (Digital Video Broadcasting). En effet, dans un récepteur de télévision numérique compatible avec ce standard, la partie démodulation du décodage canal est le seul élément du domaine numérique qui change avec le canal de transmission. La réalisation d'un récepteur multi-standard passe donc par la réalisation d'un démodulateur multi-standard.

Toutefois, il est important de ne pas perdre de vue l'objectif à plus long terme qui consiste à concevoir une architecture de codeur/décodeur canal "universel", aussi bien pour la télévision numérique que pour les autres applications de communications numériques (Internet haut débit (ADSL : Asynchronous Digital Subscriber Line), radio (DAB : Digital Audio Broadcasting)...).

Tout d'abord, dans le premier chapitre, nous caractériserons, en terme de puissance de calcul, les opérations utilisées dans les algorithmes et les applications couvertes par l'étude.

Ensuite, dans le chapitre II, un aperçu des architectures parallèles utilisées dans les communications numériques sera présenté.

Le chapitre III décrira un coprocesseur de filtrage, composant rendu nécessaire par la prédominance des algorithmes de filtrage dans le domaine des communications numériques.

Le chapitre IV présentera alors une étude architecturale. Elle proposera et estimera en complexité, coût et performances, une ou plusieurs architectures configurables, répondant aux besoins identifiés dans le chapitre I.

Le chapitre V s'attachera à la validation de l'architecture retenue avec des applications réelles. Il s'agira de définir une plate-forme de démonstration afin de vérifier l'adéquation entre l'architecture et les différentes applications envisagées.

Au terme de cette étude, les résultats obtenus seront résumés et les perspectives d'évolution de l'architecture présentées.

Chapitre I

Caractérisation des opérations de démodulation numérique

Le but des communications numériques est de transmettre un signal numérique quelconque d'un point à un autre de la manière la plus fiable possible. Or, il existe différents canaux de transmission (radio, câble, satellite, lignes téléphoniques...) ayant chacun leurs caractéristiques. Par exemple, le signal satellite est stable mais faible (émetteur lointain de faible puissance) alors que le signal câble est puissant mais soumis aux échos. Il existe donc plusieurs techniques de modulation pour utiliser au mieux les caractéristiques de ces canaux. En particulier, on peut différencier les modulations monoporteuses qui modulent une seule porteuse à un haut débit et les modulations multiporteuses qui modulent plusieurs porteuses à un faible débit. Ces différentes techniques de modulation correspondent à des algorithmes différents en modulation et en démodulation.

Ce sont quelques algorithmes de démodulations, ceux utilisés par Philips pour réaliser des décodeurs compatibles avec le standard européen DVB (Digital Video Broadcasting), que nous étudierons et caractériserons dans ce chapitre, en vue d'une réalisation matérielle. En effet, la norme définit l'émetteur (codage correcteur d'erreurs, modulation, structure de la trame) mais toute liberté est laissée à l'équipementier pour construire le récepteur. Il n'est toutefois pas question ici d'expliquer en détail la démodulation et chacun des algorithmes utilisés, mais uniquement de recenser les types de traitement à effectuer pour obtenir une base de travail nécessaire à la conception d'une architecture programmable.

I.1 Description algorithmique globale des applications

Trois canaux de transmission sont envisagés pour la diffusion des programmes de télévision numérique en Europe : la transmission par câble, la transmission par satellite

et celle par voie hertzienne. Comme les algorithmes utilisés dans la démodulation des données transmises diffèrent en fonction du canal, il est nécessaire de présenter chacune des démodulations séparément.

I.1.1 Démodulation de la transmission par satellite [3]

La modulation utilisée est une Modulation par Déplacement de Phase à 4 états (MDP4). Il suffit donc de 2 bits pour coder les 4 symboles correpondant aux 4 états. La fréquence d'échantillonnage est de 65 MHz alors que la fréquence symbole est variable, comprise entre 2 et 32 MHz.

Le démodulateur satellite *(Figure I.1)* est composé de 5 parties principales :

- Le Contrôle Automatique de Gain (CAG) permet d'utiliser au mieux la plage du Convertisseur Analogique Numérique (CAN).
- La génération IQ transpose le signal modulé en bande de base à l'aide d'une multiplication. Cette opération transforme le signal réel entrant dans le système en un signal complexe composée d'une voie "In phase" (la partie réelle du signal complexe) et d'une voie "in Quadrature" (la partie imaginaire du signal complexe), d'où la terminologie utilisée pour décrire cette fonction.
- Le Filtrage (filtre anti-repliement, filtre de l'Interpolateur, Nyquist) assure la minimisation de l'Interférence Entre Symboles (IES), la réduction du bruit...
- La synchronisation consiste à récupérer la phase et la fréquence de l'échantillonnage (récupération de l'horloge) ou de la porteuse (récupération de porteuse) grâce à des boucles de contre-réaction. Pour que la synchronisation puisse agir correctement, de fortes contraintes sur la latence des calculs sont nécessaires.
- La correction d'erreurs utilise un codage en blocs (Reed-Solomon) et un codage convolutif pour corriger les erreurs générées par la transmission. Le décodage des codes convolutifs est effectuée en utilisant l'algorithme de Viterbi.

I.1.2 Démodulation de la transmission par câble [4]

La modulation utilisée pour la transmission par câble est une modulation monoporteuse : la Modulation en Amplitude avec Quadrature (MAQ) avec une constellation de taille 16, 64, 256... Pour une MAQ-256, chaque symbole est codé sur 8 bits. La fréquence d'échantillonnage est fixée à 28,92 MHz et la fréquence symbole est variable, comprise entre 1 et 7 MHz.

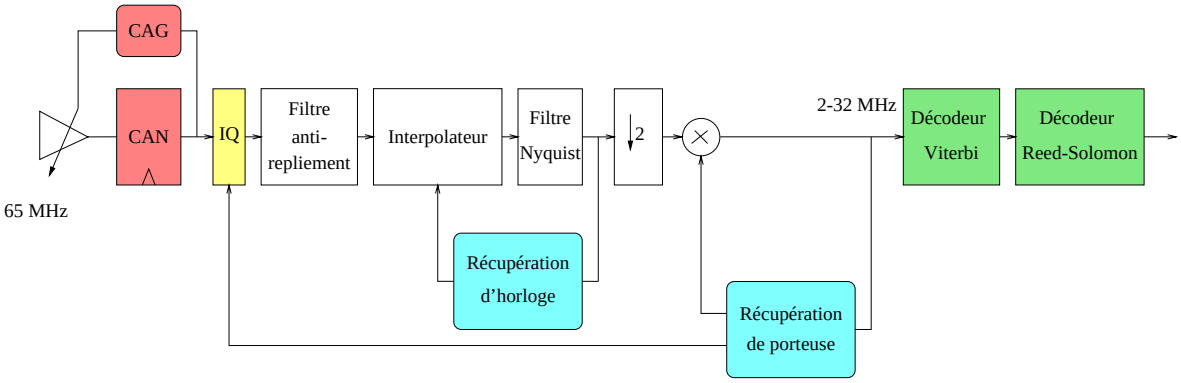


FIGURE I.1 : *Schéma d'un démodulateur satellite*

Le standard DVB pour le câble est basé sur le standard pour le satellite si ce n'est l'utilisation d'une modulation MAQ à la place d'une modulation MDP4. Par conséquent, le démodulateur câble *(Figure I.2)* est semblable au démodulateur satellite. Quelques modifications ont toutefois été apportées au système pour tenir compte des particularités des media de communication :

- Le démodulateur satellite possède en plus un décodeur de Viterbi. Celui-ci sert à décoder le code convolutif utilisé par la modulation pour supporter des rapports signal à bruit faibles dus aux faibles puissances utilisées en émission.
- Le démodulateur câble possède, en plus, un égaliseur pour compenser les imperfections du canal et en particulier les échos dus à des désadaptations d'impédance de ligne.

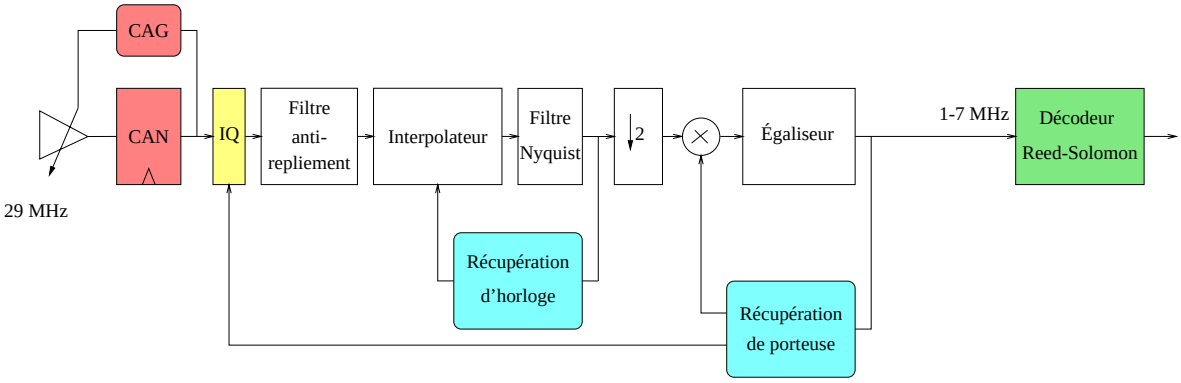


FIGURE I.2 : *Schéma d'un démodulateur câble*

I.1.3 Démodulation de la transmission par voie hertzienne [5]

La modulation utilisée est une modulation multiporteuse dont le principe est de répartir un signal à haut débit sur un nombre important de porteuses orthogona-

les modulant chacune un signal à faible débit (OFDM : Orthogonal Frequency Division Multiplexing). Cette technique offre une bonne protection contre la sélectivité du canal, la propagation multi-chemins et l'interférence entre canaux. La fréquence d'échantillonnage est fixée à 9,14 MHz mais les parties réelles et complexes sont multiplexées lors de l'émission, ce qui donne un signal d'entrée de 18,28 MHz. Un symbole contient 2048 (2k) ou 8192 (8k) échantillons en fonction de la qualité du canal de transmission. Il est composé d'une partie utile dont la durée est fixe Tu (en fonction du mode de transmission utilisé) et d'un intervalle de garde Tg dont la durée est paramétrable *(Figure I.3)*. Cet intervalle de garde est un prolongement cyclique de la partie utile du symbole. Il sert à éliminer l'interférence entre symboles (IES).

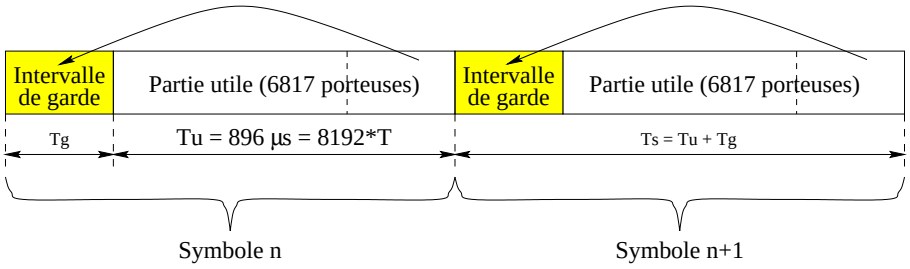


FIGURE I.3 : *Structure d'un symbole OFDM 8k*

Le démodulateur terrestre *(Figure I.4)* se compose de plusieurs algorithmes qui réalisent les fonctions suivantes :

- le Contrôle Automatique de Gain (CAG) permet d'utiliser la plage maximale du CAN,
- la génération IQ transpose le signal modulé en bande de base,
- la transformée de Fourier (FFT : Fast Fourier Transform) réalise la démodulation. Elle correspond à la FFT inverse réalisée sur les porteuses lors de la modulation,
- la synchronisation positionne correctement le début de la FFT. Cette tâche est effectuée par la récupération d'horloge et la récupération de porteuse,
- l'estimation du canal analyse la réponse du canal en utilisant des porteuses pilotes spécifiques dont la valeur et la position dans le symbole sont connues (estimation partielle du canal) et interpole cette réponse sur l'ensemble des porteuses (interpolations temporelle et fréquentielle),
- la correction d'erreurs utilise un décodage de Viterbi et un décodage de Reed-Solomon pour corriger les erreurs générées par la transmission.

Une description plus détaillée des algorithmes utilisés dans le démodulateur est disponible dans [6].

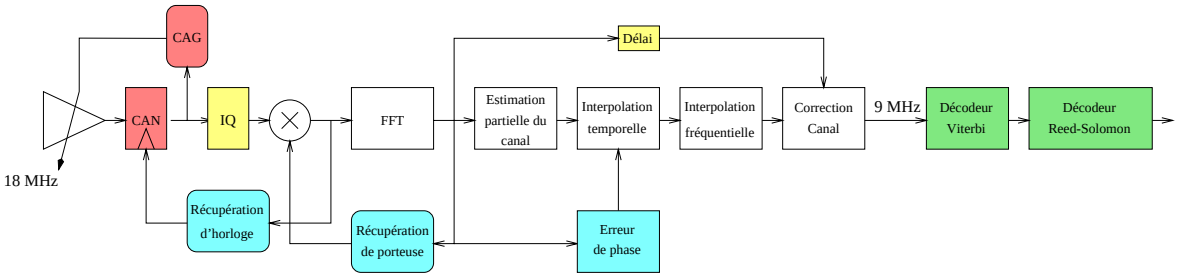


FIGURE I.4 : *Schéma d'un démodulateur terrestre*

Cette rapide description algorithmique des applications permet de répartir les différents algorithmes utilisés dans le décodage canal, en trois catégories que nous détaillerons ensuite :

- les algorithmes de type filtrage qui effectuent l'essentiel de la démodulation,
- les boucles de contre-réaction qui synchronisent le démodulateur avec le signal émis,
- la correction d'erreurs.

I.2 Le filtrage

L'essentiel de la démodulation monoporteuse s'effectue grâce à une succession de filtres ayant tous un rôle précis. Il s'agit des filtres anti-repliement, des filtres interpolateurs, des filtres de Nyquist et des égaliseurs.

I.2.1 Filtrage anti-repliement

Il s'agit d'un filtre passe-bas destiné à éviter le repliement de spectre lors d'un changement de fréquence d'échantillonnage. Par la même occasion, il filtre les canaux adjacents non désirés ainsi que les bruits de quantifications introduits par la conversion analogique/numérique.

La structure du filtre anti-repliement (AAF : Anti Aliasing Filter) utilisée est celle d'une succession de filtres symétriques à coefficients fixes en conjonction avec des décimations par un facteur 2 *(Figure I.5)*. En fait le filtre est appliqué au signal avant chaque décimation par 2 pour éviter le repliement de spectre. Cette approche permet d'utiliser un seul filtre anti-repliement bien que la fréquence symbole et donc la bande du signal utile soient variables. En effet, ce filtre anti-repliement est conçu pour éviter le repliement de spectre lorsqu'une décimation d'un facteur inférieur à 2 est appliquée.

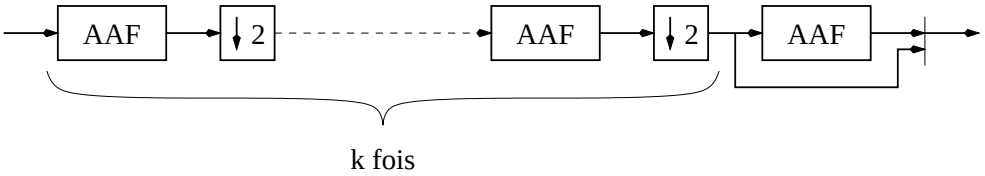


FIGURE I.5 : *Structure récursive du filtrage anti-repliement*

Le nombre d'étages, k, est choisi afin que la fréquence d'échantillonnage en sortie, f, soit strictement supérieure à deux fois la fréquence symbole tout en étant inférieure à quatre fois la fréquence symbole : $2 \times F_s < f \leq 4 \times F_s$. Comme cela, l'interpolateur qui suit ne fera pas une décimation d'un facteur supérieur à 2 pour obtenir le signal à 2 fois la fréquence symbole utilisé par le Nyquist. Ainsi, le filtre anti-repliement (celui placé après les k étages) peut encore servir pour éviter un repliement du spectre lors de la décimation qui va être réalisée par l'interpolateur.

Ce dernier filtre n'est pas toujours nécessaire. Il le sera si la décimation à réaliser dans l'interpolateur est proche de 2, plus précisément si $(3 - \alpha) \times F_s < f$ où α est le facteur de roll-off. Le facteur de roll-off est un paramètre de conception du filtre de Nyquist *(Section I.2.3)*. La valeur de ce facteur influence la largeur de la bande de fréquence du signal. Il est donc important lors du repliement de spectre.

I.2.2 Filtrage de l'interpolateur

L'usage de l'interpolateur est de sous-échantillonner le signal de la fréquence f (fraction de la fréquence d'échantillonnage) à la fréquence f' (multiple de la fréquence symbole). Le rapport entre f et f' n'étant pas entier, les échantillons utiles correspondant à f' sont interpolés à partir des échantillons à la fréquence f. Pour effectuer ce changement de fréquence, l'interpolateur reconstruit le signal continu afin de l'échantillonner à nouveau à la bonne fréquence. La reconstruction du signal continu est réalisée par un filtre passe-bas dont la réponse impulsionnelle est un sinus cardinal. L'approximation de ce filtre idéal est réalisée par un filtre à retard de phase variable (VPD : Variable Phase Delay) développé par Philips [7] *(Figure I.6)*. Cette partie utilise une combinaison d'un filtre passe-bas (filtre symétrique dont les coefficients sont C_{L1}, C_{L2}...) et d'un différentiateur (filtre antisymétrique dont les coefficients sont C_{d1}, C_{d2}). La longueur du filtre passe-bas est choisie en fonction de la qualité de l'approximation voulue. μ est un signal de contrôle généré par la boucle de contre-réaction effectuant la récupération d'horloge et qui indique la position du nouvel échantillon à reconstruire par rapport aux échantillons initiaux.

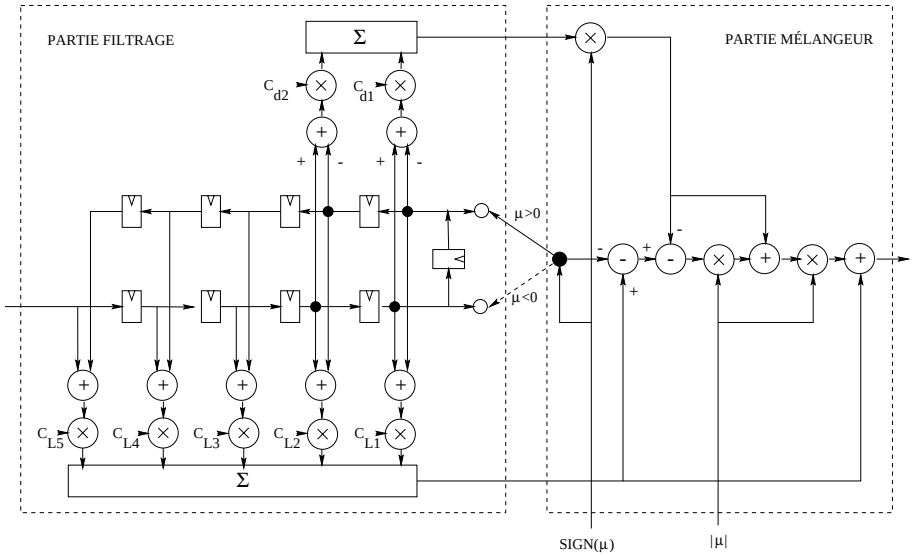


FIGURE I.6 : *Structure de l'interpolateur*

I.2.3 Filtre de Nyquist

Le filtre de Nyquist a pour but de limiter la bande passante du signal tout en évitant l'interférence entre symboles (IES). Pour cela, il doit satisfaire le premier critère de Nyquist, qui demande que la réponse temporelle du signal présente des zéros aux instants multiples de la période symbole [8].

I.2.4 Égaliseur

Un égaliseur est requis pour la réception du signal transmis par le câble afin de corriger la sélectivité du canal. Celui-ci est un égaliseur récursif avec décision dans la boucle (DFE : Decision Feedback Equalizer). Il se compose donc de deux filtres, l'un causal (le filtre avant) et l'autre anti-causal (le filtre arrière), d'un additionneur et d'un bloc de décision qui a pour rôle d'estimer la valeur des symboles en fonction des échantillons qu'il reçoit *(Figure I.7)*.

De plus, cet égaliseur est adaptatif. L'algorithme d'adaptation est une version signée de l'algorithme des moindres carrés (LMS : Least-Mean-Square) [8], ce qui permet de réduire considérablement la complexité de la réalisation matérielle puisque des multiplications sont remplacées par des opérateurs "ou exclusif". Il en résulte, cependant, un accroissement du temps de convergence de l'algorithme mais les performances restent acceptables. Cet algorithme effectue le calcul suivant :

$$w_k(n+1) = w_k(n) - \Delta \times sgn(\epsilon(n)) \times sgn(x^*(n-k)) \tag{I.1}$$

où Δ désigne le pas de l'adaptation, $x^*(n-k)$ le complexe conjugué de $x(n-k)$ et $\epsilon(n)$ l'erreur sur le résultat, le signe d'un complexe étant le complexe composé du signe de

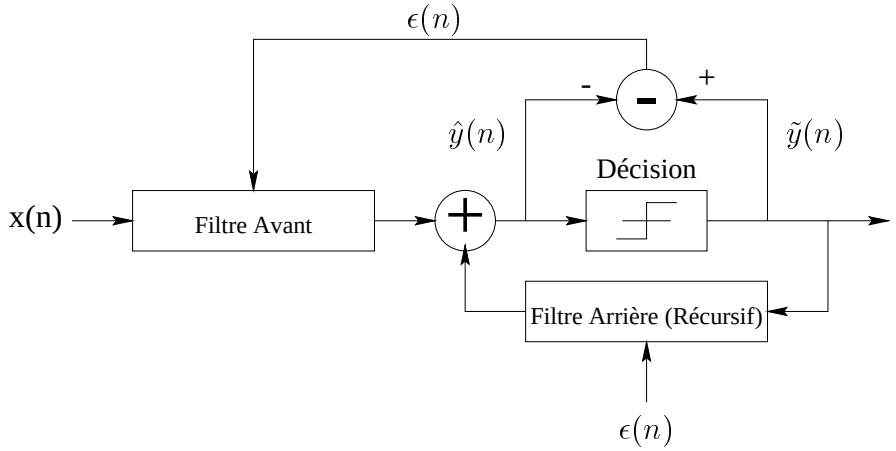


FIGURE I.7 : *Structure de l'égaliseur récursif avec décision dans la boucle*

sa partie réelle et du signe de sa partie imaginaire. Pour calculer l'erreur sur le résultat, il faut effectuer la différence entre le symbole transmis $\tilde{y}(n)$ et l'estimation de la valeur de ce symbole à la sortie du filtre adaptatif $\hat{y}(n)$:

$$\epsilon(n) = \tilde{y}(n) - \hat{y}(n) \tag{I.2}$$

Les filtres décrits auparavant étaient tous des filtres communément utilisés dans les démodulations monoporteuses. Les démodulations multiporteuses comme celle utilisée dans la démodulation terrestre utilisent moins de filtrage parce que la démodulation est essentiellement réalisée grâce à la FFT.

I.2.5 Transposition en Bande de base

Cette fonction est effectuée par une multiplication contrôlée par la boucle de récupération de porteuse dans les démodulations câble et satellite. Pour la modulation terrestre, une autre technique est employée. Elle consiste à multiplexer les voies I et Q après avoir décalé la voie Q d'un demi-échantillon grâce à un filtre symétrique. Alors, la transposition s'effectue en démultiplexant I et Q et en décalant à nouveau la voie Q d'un demi-échantillon grâce au même filtre, tandis que l'échantillon de la voie I est retardé d'un échantillon.

I.2.6 Interpolation fréquentielle et temporelle

L'interpolation temporelle et fréquentielle réalisée dans la démodulation terrestre sert à étendre à l'ensemble des porteuses d'un symbole, l'estimation du canal qui est réalisée sur un nombre restraint de porteuses pilotes éparpillées (scattered pilots) sur

plusieurs symboles OFDM. L'interpolation temporelle utilise une mémoire pour interpoler les porteuses comprises entres les porteuses pilotes de deux symboles voisins *(Figure I.8)*. Il s'agit d'une interpolation par un facteur 4. Ensuite à partir de ce nouvel ensemble, l'interpolation fréquentielle détermine la valeur des autres porteuses d'un même symbole grâce à un filtre dont le facteur d'interpolation est 3, ce qui correspond à la réalisation de 3 filtres standards.

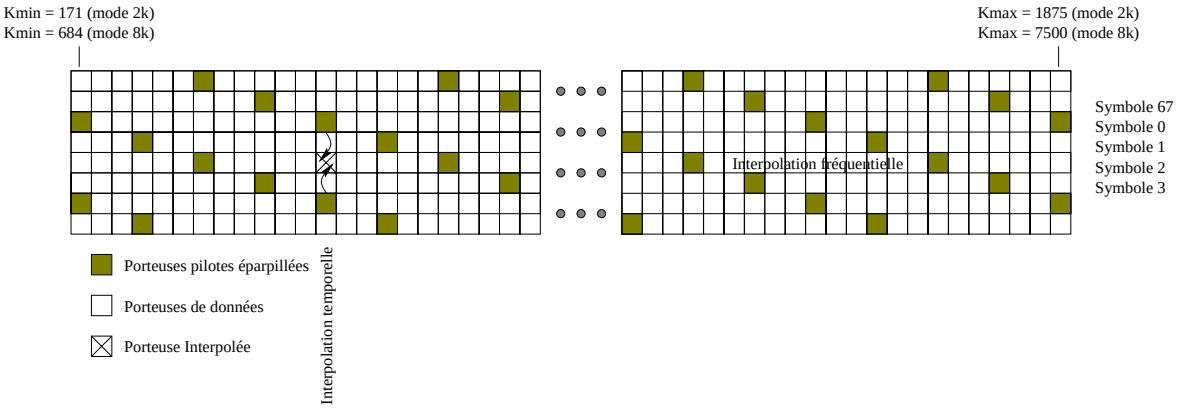


FIGURE I.8 : *Interpolations temporelle et fréquentielle*

La *figure I.8* représente la structure d'une trame OFDM composée de plusieurs symboles et indique la position des porteuses pilotes utilisées pour estimer le canal.

Avec la description des opérations de filtrage des démodulations multiporteuses se termine la description des principales parties de la chaîne directe de la démodulation. Il reste à décrire les boucles de contre-réaction qui servent à synchroniser les démodulations.

I.3 Les boucles de contre-réaction

Deux types de boucles de contre-réaction sont utilisés dans les démodulations numériques étudiées :

- Des boucles de synchronisation qui ont pour fonction de récupérer le rythme et la porteuse. Il s'agit souvent de boucles à verrouillage de phase numériques (DPLL : Digital Phase Locked Loop) pour les démodulations monoporteuses alors qu'il s'agit de calcul de fonctions d'autocorrélation servant à trouver la position des porteuses pilotes pour les démodulations multiporteuses.

- Des boucles de contrôle de gain pour utiliser au mieux la dynamique des données.

I.3.1 Les boucles à verrouillage de phase numériques

Ces boucles ont pour fonction de synthétiser un signal de référence ayant la même phase et la même fréquence que le signal qui a servi à moduler le signal reçu afin de démoduler correctement le signal. En fonction des spécifications systèmes et des performances requises pour les boucles de contre-réaction, les algorithmes utilisés peuvent être plus ou moins complexes [9].

Il reste néanmoins possible de décomposer les boucles à verrouillage de phase numériques en 4 parties distinctes *(Figure I.9)* :

- un détecteur de phase qui calcule l'erreur de phase du signal d'entrée,
- un filtre de boucle qui est un filtre passe-bas qui sert à lisser les fluctuations de la sortie du détecteur de phase,
- un oscillateur contrôlé numériquement (NCO : Numerically Controlled Oscillator) qui synthétise un signal de référence,
- une correction du signal en fonction du signal de référence synthétisé. Cette transformation est habituellement un changement de la phase du signal pour le remettre en phase avec le signal émis.

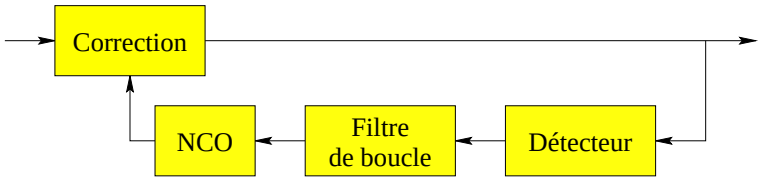


FIGURE I.9 : *Structure d'une boucle à verrouillage de phase numérique*

I.3.1.a Les détecteurs

Il existe de nombreuses variétés de détecteurs. Seuls ceux utilisés dans les démodulations étudiées sont présentés ici.

Le détecteur de Gardner

Le détecteur de Gardner [10] est utilisé dans le câble et le satellite pour le calcul de la récupération d'horloge. Il analyse la puissance du signal reçu à la sortie du filtre de Nyquist et décide si l'échantillonnage a été effectué en avance ou en retard par rapport à l'instant idéal d'échantillonnage. En effet, la puissance du signal atteint son maximum

lorsque l'instant d'échantillonnage est idéal. Pour effectuer cette analyse de puissance, la fonction suivante est réalisée :

$$\begin{aligned} y(n) = &((x^I(2\times n) - x^I(2\times n-2)) \times x^I(2\times n-1)) \\ &+((x^Q(2\times n) - x^Q(2\times n-2)) \times x^Q(2\times n-1)) \end{aligned} \tag{I.3}$$

avec $(x^I(n), x^Q(n))$ la $n^{ième}$ donnée complexe.

Le détecteur de phase et de fréquence

Le détecteur utilisé pour la récupération de porteuse est un détecteur qui permet de corriger des erreurs sur la phase des symboles (contre-réaction sur la multiplication suivant le filtre de Nyquist) ainsi que l'erreur sur la fréquence et sur la phase de la porteuse (contre-réaction sur la transposition en bande de base). Le résultat du détecteur dépend de la position du symbole reçu sur la constellation utilisée par la modulation [11].

I.3.1.b Le filtre de boucle

En fonction du détecteur utilisé et du format du signal généré par celui-ci (sur un ou plusieurs bits), différentes classes de filtres doivent être employées. Les filtres les plus souvent utilisés à cet endroit sont des filtres à réponse impulsionnelle infinie (RII) qui peuvent s'écrire sous la forme :

$$y(n) = \sum_{i=1}^{M} a_i \times y(n-i) + \sum_{i=0}^{N-1} b_i \times x(n-i) \tag{I.4}$$

Un tel filtre est dit d'ordre M et la DPLL est alors d'ordre $M+1$.

I.3.1.c Les oscillateurs contrôlés numériquement

Un NCO (Numerically Controled Oscillator) est un compteur à pas variable et à seuil variable. À chaque cycle d'horloge le compteur est incrémenté de la valeur du pas, et chaque fois que la valeur du compteur dépasse le seuil, une remise à zéro du compteur est effectuée. La phase du NCO est la valeur de ce compteur. Cette phase est utilisée pour générer une base de temps (récupération d'horloge) ou une correction (récupération de porteuse).

I.3.1.d La correction

La correction effectuée dépend de la fonction qui doit être réalisée par la boucle de contre-réaction. Il s'agit souvent d'une modification de la phase ou de la fréquence de

la porteuse. Mais il peut aussi s'agir de l'interpolation.

La transposition en bande de base

Il s'agit d'une multiplication complexe qui transpose le signal d'une fréquence intermédiaire (IF) en un signal en bande de base grâce à la phase donnée par le NCO, ϕ_{NCO} :

$$y(n) = x(n) \times e^{2j\pi \times \phi_{NCO}} \tag{I.5}$$

Le mélangeur de l'interpolation

Il s'agit d'échantillonner à nouveau le signal d'entrée qui est à une fréquence fraction de la fréquence d'échantillonnage ($\frac{F_e}{k}$), en un signal à 2 fois la fréquence symbole ($2F_s$), grâce à la sortie du NCO qui donne le bon rythme. Ceci est réalisé par un mélangeur dont la structure a été présentée à la *figure I.6*.

La correction de phase

Il s'agit d'une multiplication complexe qui corrige la phase du symbole :

$$y(n) = x(n) \times e^{2j\pi \times \phi_{NCO}} \tag{I.6}$$

I.3.2 Les fonctions d'autocorrélation

La synchronization du système de communication terrestre utilise différentes porteuses pilotes réparties dans les symboles et ayant des valeurs connues. Il existe différentes sortes de porteuses pilotes :

- les pilotes TPS (Transmission Parameter Signalling) qui transportent les paramètres utilisés dans le codage canal.
- Les pilotes fixes (continual pilots) dont la position et la valeur sont connues, identiques pour chaque symbole.
- Les pilotes éparpillés (scattered pilots) dont la position et la valeur sont connues, mais varient d'un symbole à l'autre comme l'indique la *figure I.8*.

Soit $R_{n,k}$ la k-ième donnée du n-ième symbole d'une trame OFDM. Alors on calcule :

$$cor(n) = \sum_{k \in E} R_{n,k} \times R^*_{n-1,k} \tag{I.7}$$

où E est un ensemble de porteuses pilotes et $cor(n)$ l'autocorrélation des pilotes correspondant au symbole n. Si les calculs s'exécutent sur le bon ensemble, alors le résultat de la somme atteindra un pic.

Ainsi, il est possible de réaliser la synchronisation temporelle (déterminer le début d'un symbole) du système grâce aux échantillons contenus dans l'intervalle de garde qui sont identiques aux derniers échantillons du symbole. Il est également possible de réaliser la synchronisation fréquentielle (déterminer précisément la position des différentes porteuses dans un symbole) en utilisant un ensemble de porteuses ayant des valeurs et des positions prédeterminées (les pilotes fixes et les pilotes éparpillés).

I.3.3 Le Contrôle Automatique de Gain (CAG)

Cet algorithme calcule la puissance moyenne du signal afin de contrôler l'amplitude du gain à appliquer au signal. Selon la puissance du signal, le gain de l'amplificateur est augmenté ou réduit *(Figure I.10)*.

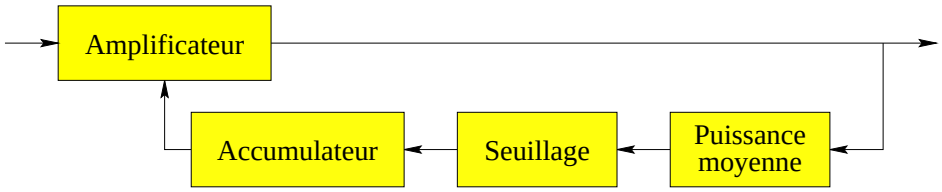


FIGURE I.10 : *Structure du Contrôle Automatique de Gain*

I.4 La correction d'erreur

I.4.1 Le décodeur de Viterbi

L'algorithme de Viterbi [12] est un algorithme de décodage optimal pour les codes convolutifs [13]. Il reconstitue a posteriori l'évolution du codeur la plus probable et en déduit la séquence de bits d'informations qui ont fait évoluer le codeur de cette manière. Il est composé de trois parties :

- La première consiste à calculer les métriques de branches qui correspondent à la distance entre les symboles reçus et les transitions possibles.
- La deuxième effectue la mise à jour des métriques de noeuds par Addition Comparaison et Sélection en fonction des métriques de branches.

- La dernière partie s'occupe de la gestion du chemin survivant qui permet de décoder un bit d'information reçu en fonction des décisions prises lors de la réception des symboles qui le suivent.

L'implantation de cet algorithme est abondamment traité dans la littérature [14–16]. Une démarche unificatrice des différentes méthodes est proposée dans [17].

I.4.2 Le décodeur de Reed-Solomon

Le standard DVB pour le câble, le satellite et le terrestre utilise le même code de Reed-Solomon (RS(188,204)). Ce code permet de corriger jusqu'à $t = 8$ octets erronés sur un bloc de 204 octets comprenant 188 octets d'information et $2t$ octets de redondance. Le décodage s'effectue en trois étapes *(Figure I.11)* :

- La première étape consiste à calculer le syndrome correspondant au message. Ce syndrome se compose de $2t$ éléments (octets) et dépend uniquement des erreurs modifiant le mot de code (le bloc de 204 octets). Si tous les éléments du syndrome sont nuls, alors il n'y a aucune erreur à corriger.
- La deuxième étape localise les erreurs. Pour cela, il suffit d'appliquer l'algorithme d'Euclide (le calcul du plus grand commun diviseur) à partir de l'équation clé du décodage [18] :

$$\sigma(x)S(x) \equiv \omega(x)[x^{2t}] \tag{I.8}$$

  où $S(x)$ est le syndrome, $\sigma(x)$ le localisateur d'erreurs et $\omega(x)$ l'évaluateur d'erreurs.
- La dernière étape évalue l'amplitude de chaque erreur e_i en utilisant l'équation :

$$e_i = -\alpha^i \frac{\omega(\alpha^{-i})}{\sigma'(\alpha^{-i})} \tag{I.9}$$

  lorsque $\sigma(\alpha^{-i}) = 0$ où α est une racine 256ième de l'unité.

Ensuite, il reste à corriger l'erreur par une addition avec le mot de code qui a été conservé en mémoire pendant le temps des calculs.

Toutes les opérations du décodeur s'effectuent dans le corps de Galois GF(256) si bien que les opérations de multiplications et d'additions nécessitent soit un accès dans une mémoire, soit un opérateur spécifique.

De nombreuses implantations de cet algorithme peuvent être trouvées dans la littérature [19–22].

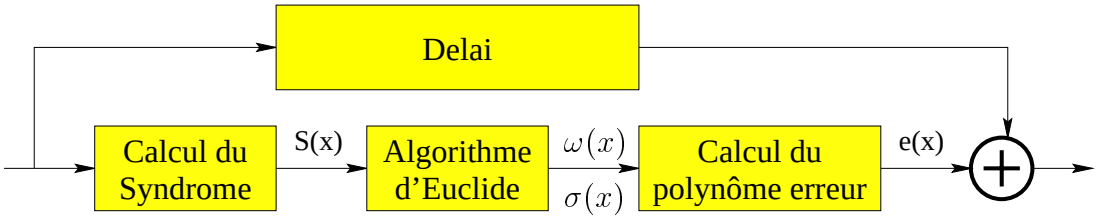


FIGURE I.11 : *Schéma du décodeur de Reed-Solomon*

Comme toutes les applications présentées utilisent le même décodeur de Reed-Solomon (RS(188,204)) et qu'il utilise des opérations spécifiques (dans le corps de Galois), cette partie du système est séparée du reste des algorithmes de décodage canal. En effet, la réalisation de ce décodeur n'a en aucun cas besoin de la flexibilité apportée par une architecture programmable puisque le même décodeur peut déjà être utilisé tel quel par l'ensemble des applications. Le même raisonnement s'applique au décodeur de Viterbi, si ce n'est qu'il n'est pas utilisé dans la transmission par câble.

I.5 Caractéristiques des communications numériques

Maintenant que les différentes démodulations ont été décrites brièvement, il est possible de résumer les puissances de calcul et les puissances des échanges avec la mémoire mise en jeu afin de mettre en évidence des synergies entre les différents algorithmes des applications. On obtiendra ainsi une base de départ pour le développement d'une architecture programmable pour le décodage canal.

I.5.1 Répartition de la puissance de calcul par application

Différents modes de fonctionnement sont possibles. Ils dépendent de la fréquence symbole et ils ont une influence sur la longueur du filtre anti-repliement. Seules les estimations de puissances de calcul nécessaires dans le pire des cas *(Figure I.12 et I.13)* sont présentées ici. En raison des hautes fréquences symboles utilisées dans le satellite, les puissances de calcul requises sont très importantes (environ 12 Gops). Dans ces estimations de puissance de calcul, une opération correspond à une addition, une multiplication, une opération logique... sur des données réelles sans tenir compte de la taille de ces données.

Dans les *figure I.12 et I.13*, l'interpolateur désigne en fait les filtres de l'interpolateur et les multiplications désignent les fonctions qui effectuent uniquement une multiplication complexe (la transposition en bande de base ou la correction suivant le Nyquist).

En ce qui concerne la démodulation terrestre, la FFT et le filtrage (transposition en bande de base et interpolation fréquentielle) représentent 70% des calculs *(Figure I.14)*.

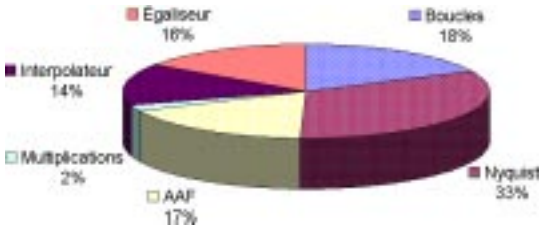


FIGURE I.12 : *Puissance de calcul (câble)*
Fréquence symbole de 7 MHz
Puissance totale de 4,5 Gops

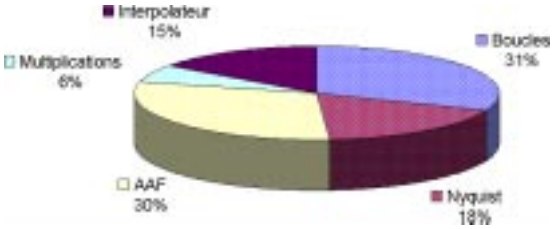


FIGURE I.13 : *Puissance de calcul (satellite)*
Fréquence symbole de 24 MHz
Puissance totale de 11,5 Gops

Mais la puissance de calcul requise ne dépasse pas 1,5 Gops, ce qui est faible par rapport aux besoins du satellite.

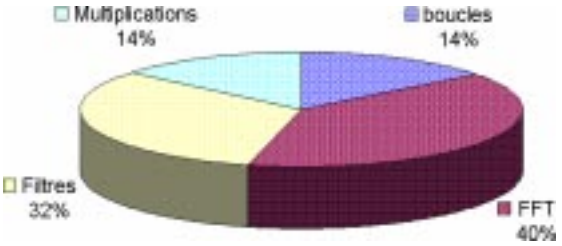


FIGURE I.14 : *Puissance de calcul du terrestre*
Taille symbole de 8K
Puissance totale de 1,5 Gops

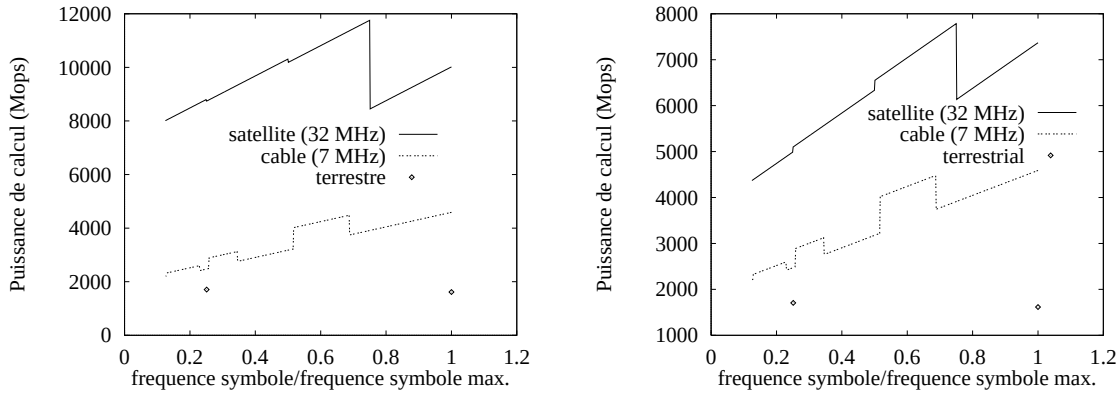
I.5.2 Comparaison des besoins en puissance de calcul

Une première analyse semble montrer que le satellite demande une puissance de calcul beaucoup plus importante que les autres transmissions *(Figure I.15(a))*. En fait, cet écart peut être réduit en utilisant la technique du parallélisme de sous-mots puisque l'application la plus coûteuse en puissance de calcul est aussi celle qui utilise les données les plus petites. En effet, les symboles du satellite comportent moins de bits d'information (2 bits) que les autres symboles (8 bits pour le câble, 10 pour le terrestre).

La *Figure I.15(a)* représente les puissances de calcul requises par les différentes applications en fonction des différentes fréquences symboles utilisées. Pour pouvoir comparer ces applications dont les fréquences symboles sont différentes, l'axe des x est normalisé par la fréquence symbole maximale de chaque application qui apparaît entre parenthèse dans la légende de la figure.

Pour le câble et le satellite, la puissance de calcul de la plupart des algorithmes est une fonction linéaire croissante, soit de la fréquence d'échantillonnage (fixe), soit de la fréquence symbole (variable). Ainsi, les courbes représentant la puissance de calcul

en fonction de la fréquence symbole devraient être des fonctions linéaires croissantes. Les discontinuités des courbes sont liées au fonctionnement du filtre anti-repliement *(Figure I.5)* dont le nombre d'étages de filtrage dépend de la fréquence symbole. L'ajout d'un étage de filtrage sans décimation augmente considérablement la puissance de calcul du filtrage anti-repliement sans changer celle des autres fonctions. Par contre, l'ajout d'un étage de filtrage avec décimation réduit également la puissance de calcul de l'interpolateur situé aprés le filtre anti-repliement. Cette réduction étant plus importante que l'augmentation due au filtre anti-repliement, une réduction notable apparaît. Pour le câble, il y a 6 combinaisons de 4 filtres anti-repliement possibles, donc 5 discontinuités et pour le satellite, il y en a 3. Pour le terrestre, il n'y a que 2 modes de fonctionnement qui correspondent à la taille du symbole, 2k ou 8k échantillons, dont l'impact se situe essentiellement, en ce qui concerne la puissance de calcul, sur la FFT.



(a) *Sans parallélisme de sous-mots* (b) *Avec parallélisme de sous-mots*

FIGURE I.15 : *Besoins des applications en fonction de la fréquence symbole*

I.5.3 Parallélisme de sous-mots

Le parallélisme de sous-mots [23] consiste à séparer un opérateur (resp. un mot) de w bits de large en p ($p \geq 2$) opérateurs (resp. sous-mots) de $\frac{w}{p}$ bits de large. En fait ce parallélisme s'apparente à un parallélisme SIMD (Single Instruction stream, Multiple Data stream). Ce parallélisme de sous-mots permet d'augmenter la puissance de calcul d'un processeur (dans les cas où une faible précision suffit) à un faible coût (quelques multiplexeurs supplémentaires dans les opérateurs et quelques instructions supplémentaires dans le jeu d'instructions).

Ce parallélisme peut être avantageusement utilisé dans les communications numériques puisque les différentes applications utilisent différentes précisions (8 ou 16 bits). Mais ce parallélisme entraîne quelques délais lorsqu'il faut attendre plusieurs données pour créer un mot (un paquet de données), ce qui augmente la latence des boucles de contre-réaction qui est pourtant critique dans la synchronisation.

Ainsi, la latence des boucles de contre-réaction devient inacceptable si l'on utilise

le parallélisme de sous-mots en permanence. Par contre, si le parallélisme de sous-mots n'est pas utilisé dans ces boucles, alors l'augmentation de latence est à peine perceptible. Et comme la partie la plus importante de la puissance de calcul des applications se trouve dans le filtrage et non dans les boucles de contre-réaction, il n'est pas gênant de ne pas utiliser le parallélisme de sous-mots dans ces boucles.

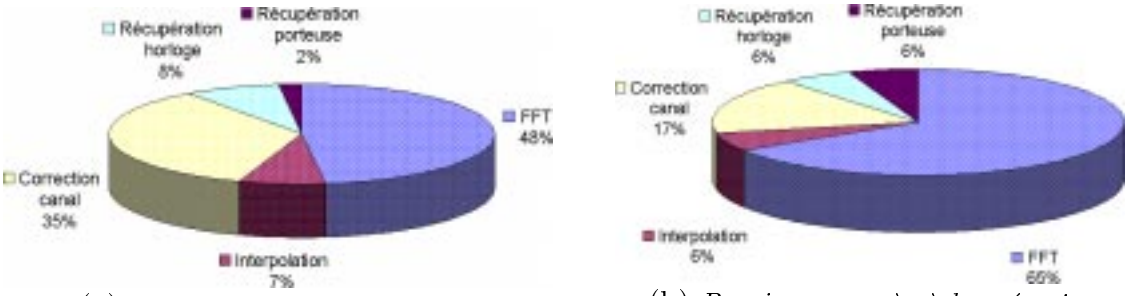
Ainsi, on obtient une nouvelle distribution des puissances de calcul requises par les applications *(Figure I.15(b))* qui remplace la distribution précédente *(Figure I.15(a))*. Dans l'ancienne distribution, une opération correspondait à une opération sur une donnée quelle qu'elle soit alors que, dans la nouvelle, elle correspond à une opération sur des données 16 bits ou deux opérations sur des données 8 bits. Ainsi, la puissance de calcul requise par le satellite est fortement réduite.

Le parallélisme de sous-mots peut donc être utilisé efficacement pour améliorer les performances de notre architecture.

I.5.4 Analyse des besoins en mémoire

Pour les démodulations câble et satellite, le besoin en mémoire est très faible. Seules des tables constantes sont requises dans les boucles de contre-réaction.

Par contre, pour la démodulation terrestre, une importante quantité de mémoire (150 Ko) est nécessaire *(Figure I.16(a))*, pour la FFT bien sûr, mais aussi pour la correction du canal et la synchronisation, puisqu'il faut mémoriser plusieurs symboles. La bande passante requise par les accès à cette mémoire est elle aussi importante avec ses 330 Mo/s *(Figure I.16(b))* surtout dus aux accès répétés de la FFT à la mémoire.



(a) *Besoins en espace mémoire*
Taille mémoire totale de 150 Ko

(b) *Besoins en accès à la mémoire*
Débit mémoire total de 330 Mo/s

FIGURE I.16 : *Besoins en mémoire pour la démodulation terrestre*

I.6 Conclusion

Si on étudie attentivement les algorithmes mis en œuvre dans les applications de démodulation présentées en début de chapitre (sans compter les décodeurs de Viterbi

et de Reed-Solomon qui utilisent des opérations spécifiques et qui sont identiques pour les différentes démodulations présentées), on s'aperçoit que l'essentiel des calculs (entre 70% et 90%) est réalisé dans les filtres ou les FFT.

La conception d'une architecture programmable pour le décodage canal qui fait l'objet de cette thèse devra donc porter un soin particulier à la mise en œuvre des filtres et de la FFT. C'est évident pour les filtres qui demandent jusqu'à 8 Gops dans le satellite alors que la FFT ne demande que 600 Mops. Néanmoins, comme la FFT nécessite des accès mémoire à une vitesse critique, sa conception devra également être étudiée avec attention.

Le tableau I.1 présente un récapitulatif des paramètres important des applications ainsi que des besoins requis par ces algorithmes.

	Satellite	Câble	Terrestre
Fréquence d'échantillonnage	65 MHz	28,92 MHz	9,14 MHz
Fréquence symbole max.	32 MHz	7 MHz	9,14 MHz
Précision des données	8 bits	10 bits	12 bits
Puissance de calcul	11,5 Gops	4,5 Gops	1,5 Gops
Latence	Critique	Critique	Sans importance
Taille mémoire	X	X	150 Ko
Accès mémoire	X	X	330 Mo/s

TABLEAU I.1: *Caractéristiques et besoin des applications*

Le *tableau I.2* présente les évolutions technologiques prévues par la "Semiconductor Industry Association" dans le domaine des semiconducteurs pour les années à venir.

Ces prévisions montre que la technologie permettant de réaliser des microprocesseurs ayant les capacités de réaliser le décodage canal grâce à des fréquences de travail supérieures au GigaHertz sont presque disponibles.

Néanmoins, une fréquence de travail de l'ordre de 100 MHz a été imposée par Philips. Le projet étant exploratoire et non stratégique, une technologie sans risque et peu chère à développer était donc requise.

Year of First Product Shipment	1997	1999	2002	2005	2008	2011
Technology Node						
Dense Lines (DRAM Half-Pitch)(nm)	250	180	130	100	70	50
Isolated Lines (MPU Gates)(nm)	200	140	100	70	50	35
Logic (high-volume, cost-performance: MPU)						
Logic transistors/cm^2 (packed, including on-chip SRAM)	3.7M	6.2M	18M	39M	84M	180M
Logic (low-volume: ASIC)						
Usable transistors/cm^2 (auto layout)	8M	14M	24M	40M	64M	100M
Chip Frequency (MHz)						
On-chip local clock (high performance)	750	1250	2100	3500	6000	10000
On-chip, across-chip clock (high performance)	375	1200	1600	2000	2500	3000
On-chip, across-chip clock (high performance, ASIC)	300	500	700	900	1200	1500
Chip Size (mm^2)						
DRAM	280	400	560	790	1120	1580
Microprocessor	300	340	430	520	620	750
ASIC [max litho field area]	480	800	900	1000	1100	1300

TABLEAU I.2: *Évolutions technologiques dans l'industrie du semiconducteur [1]*

Chapitre II

État de l'art des architectures pour les communications numériques

Comme on l'a vu dans le chapitre précédent, les démodulations numériques monoporteuses (MAQ, MDP) sont essentiellement composées de filtres de différents types qui demandent des puissances de calcul importantes. Les démodulations numériques multiporteuses (OFDM), elles, sont principalement composées d'une transformée de Fourier. Compte tenu des technologies actuelles, l'utilisation d'architectures dédiées est alors souvent obligatoire.

Dans ce chapitre, nous présenterons l'état de l'art des circuits pour la démodulation numérique. Le filtrage représentant le cœur de la démodulation, une attention particulière sera portée sur les architectures programmables pour le filtrage.

II.1 Les architectures câblées

Pour les modulations monoporteuses utilisées pour le câble et le satellite, il a été montré que le filtrage représentait l'essentiel des calculs, si bien que la conception de certains démodulateurs est presque réduite à la conception d'un filtre [24, 25]. Il faut dire que le calcul direct des filtres (un multiplieur et un additionneur par coefficient du filtre) implique un nombre important de multiplieurs, ce qui mène à des solutions "surdimensionnées" avec les technologies actuelles. Lorsque les coefficients des filtres sont fixes il est donc souvent fait appel aux techniques de conception de filtres sans multiplieur [26, 27] pour réduire la complexité des circuits. Cela permet de remplacer un multiplieur par quelques additionneurs, ce qui est clairement optimum pour réaliser un filtre à haut débit. C'est ainsi qu'il est possible de faire des démodulateurs MAQ complexes (en puissance de calcul théorique) [24,28] ainsi que des démodulateurs MDP-4 pour le satellite [29] à des coûts (en surface) très faibles [24, 28].

Malheureusement, ces architectures souffrent d'un manque de versatilité (recherchée dans notre étude). En effet, un nouveau circuit doit être réalisé à chaque évolution de la

norme de transmission ou à chaque progrès dans les algorithmes de démodulation. De plus, les circuits de filtrage sans multiplieur ne peuvent pas être utilisés pour effectuer les filtres adaptatifs qui sont nécessaires pour réduire l'effet de la propagation multi-chemins (transmission hertzienne) ou la distorsion du spectre (transmission par câble). En effet, la valeur des coefficients des filtres adaptatifs varie continuellement dans le temps. L'utilisation de vrais multiplieurs s'impose alors dans l'architecture [25,30–33].

Pour les modulations multiporteuses comme celles utilisées dans la démodulation terrestre, c'est la mise en œuvre des transformées de Fourier qui est primordiale. Ses besoins en puissance de calcul et en accès mémoire sont tels qu'ils nécessitent actuellement un circuit dédié complet [34,35].

Une première approche pour atteindre l'objectif de la r'ealisation d'un démodulateur de télévision numérique multi-standard consiste donc à juxtaposer les trois types de macro-blocs dans une technologie future plus efficace : un macro-bloc pour réaliser la démodulation satellite et ses longs filtres à coefficients constants, un autre macro-bloc pour réaliser la démodulation câble et son filtre adaptatif et un dernier macro-bloc pour réaliser la démodulation terrestre et sa FFT.

Cette solution est clairement sous-optimale en terme de complexité puisque un seul macro-bloc sur trois est actif à un moment donné alors que les algorithmes sont similaires, en particulier pour les démodulations câble et satellite qui utilisent massivement les multiplications et les additions. Cette solution a quand même de nombreux avantages : un temps de conception du circuit réduit au minimum, un besoin nul en recherche et en développement, une faible consommation puisque seul le bloc utilisé doit être alimenté.

Elle ne présente pourtant aucun intérêt pour nous dans l'optique de la thèse puisqu'on ne peut pas espérer concevoir un "démodulateur universel" en associant simplement ensemble des démodulateurs dédiés.

II.2 Les processeurs programmables

Avec les progrès technologiques, il apparaît sans cesse des microprocesseurs de plus en plus puissants ayant vocation à remplacer, à terme, les circuits dédiés dans la plupart des domaines et en particulier dans celui qui nous intéresse, les communications numériques [36–38]. C'est le cas de la nouvelle famille de DSPs de Texas Instruments, les TMS320C6x [39] et ses 1600 Mops obtenus grâce à une architecture VLIW (Very Long Instruction Word).

Cependant, ces processeurs n'ont pas encore la puissance suffisante, pour des raisons technologiques, pour pouvoir gérer la dizaine de Gops que nécessite la démodulation satellite. Pour cela il faut encore utiliser des architectures programmables adaptées aux communications. Ces architectures mélangent des parties complètement programmables pour leur flexibilité et des parties plus spécialisées pour les algorithmes demandant

une importante puissance de calcul (filtres, FFT).

De plus, le marché de la démodulation pour la télévision numérique est suffisamment étendu pour justifier à lui seul le développement d'un ASIC dédié puisque les coûts de développement sont facilement compensés par le gain sur les coûts de production dû à la surface minimale d'une telle solution.

II.3 Les architectures adaptées

La société OREN Semiconductor a annoncé en août 1998 le premier circuit de démodulation multi-standard pour la télévision numérique américaine [40]. Son architecture utilise une combinaison de composants dédiés et programmables, mais son architecture détaillée n'étant pas diffusée, il n'est pas possible de l'analyser.

Dans le domaine du codage vidéo par contre, ce type d'architecture est abondamment utilisé [41, 42], ainsi que dans les nouveaux Média Processeurs [43] qui sont constitués de plusieurs unités de calcul spécifiques architecturées autour d'un processeur VLIW (TriMedia de Philips, MSP-1 de Samsung, MediaProcessor de MicroUnity [44]) ou Risc (Mpact 2 de Chromatic).

Comme les architectures dédiées sont trop peu flexibles pour pouvoir être utilisées dans la conception d'un démodulateur universel et comme les architectures programmables sont trop peu puissantes pour réaliser la démodulation dans le domaine de la télévision numérique, seules les architectures adaptées peuvent, actuellement, prétendre être utilisées pour réaliser une architecture programmable pour le décodage canal.

Pour effectuer cette architecture programmable adaptée à la démodulation de la télévision numérique, il est nécessaire de s'intéresser de près à la manière de réaliser une architecture programmable pour effectuer l'élément clé de la démodulation : le filtrage.

II.4 Les circuits programmables de filtrage

La différence principale entre les circuits dédiés de filtrage et les circuits programmables de filtrage est le caractère programmable des coefficients stockés dans une mémoire. De plus, les circuits programmables peuvent aussi être configurés afin de réaliser différents types de filtres comme les filtres symétriques, les filtres interpolateurs ou décimateurs, les filtres adaptatifs, les filtres complexes.

Parmi les circuits programmables de filtrage, on peut distinguer trois familles : les circuits configurables qui sont des circuits de filtrage dont seuls les coefficients sont

programmables et qui utilisent une architecture systolique, les circuits à base de FPGA (Field Programmable Gate Array) et les processeurs de filtrage qui s'apparentent à des processeurs de traitement du signal (DSP) avec un jeu d'instructions réduit au minimum pour réaliser des filtres avec une meilleure efficacité que les DSP normaux.

II.4.1 Circuits configurables pour le filtrage

Ces circuits se différencient entre eux par la conception des multiplieurs qui est au cœur du filtrage. Quelques-uns utilisent de vrais multiplieurs [45–50]. D'autres utilisent des plans de bits pour augmenter la régularité du circuit et ainsi réduire sa surface [51–55]. Des solutions digit série sont aussi envisagées pour un meilleur compromis entre débit et surface [56]. Les EPROM (Erasable Programmable Read Only Memory) sont une autre possibilité de réalisation des multiplieurs [57–61] ainsi que les RAM (Random Acces Memory) lorsqu'il s'agit d'effectuer des filtres adaptatifs [62]. Finalement, la dernière technique emploie les puissances de 2 pour coder efficacement les coefficients [63]. En particulier, la représentation CSD (Canonic Signed Digit) des coefficients se trouve être très intéressante puisqu'elle réduit au minimum le nombre de puissances de 2 requises pour coder un coefficient [64–66], ce qui transforme un multiplieur en quelques décaleurs de bits et additionneurs.

Certains des circuits présentés ci-dessus peuvent calculer un large éventail (en taille) de filtres numériques à hauts débits grâce à leurs propriétés de cascadabilité. Ils ont pour but d'obtenir la plus petite taille, le plus grand nombre de coefficients ou la plus grande vitesse de fonctionnement. En contrepartie, leur flexibilité est limitée puisqu'ils ne supportent qu'un nombre limité de types de filtres. De plus, les architectures utilisant les EPROM ou la représentation CSD ne peuvent pas calculer les filtres adaptatifs, puisqu'ils n'utilisent pas les coefficients directement. En effet, elles nécessitent un codage des coefficients (CSD) ou un ensemble de multiplications possibles (l'EPROM mémorise les résultats de toutes les multiplications possibles entre une donnée et un coefficient fixé).

II.4.2 Circuits de filtrage à base de FPGA

Une nouvelle méthode pour réaliser les filtres utilise les FPGA. Cela a été rendu possible par les améliorations apportées à leur capacité et à leur vitesse de fonctionnement. Il a fallu trouver des fonctions élémentaires de calcul de filtres qui puissent être incluses dans un bloc logique configurable [67].

Ces mises en œuvre ont un avantage comparé aux précédentes : elles sont vraiment flexibles du fait même de la nature des FPGA, si bien qu'une reconfiguration à la volée peut être envisagée pour gérer n'importe quel filtre. Cependant, la réalisation d'un filtre sur FPGA est encore loin de pouvoir concurrencer la réalisation de circuits

dédiés pour le filtrage. Il s'agit plutôt d'offrir une alternative à l'utilisation de DSP dans des sytèmes demandant une puissance de filtrage de quelques Mops [68]. Il n'est donc pas envisageable d'utiliser une telle approche pour réaliser des filtres correspondant à plusieurs centaines de Mops.

II.4.3 Processeurs de filtrage

Pour obtenir un maximum de flexibilité sans perdre trop en performance, des processeurs dédiés au filtrage ont été conçus [69–74]. Plutôt que d'utiliser un multiplieur pour chaque coefficient, un multiplieur est capable de réaliser les multiplications correspondant à plusieurs coefficients de manière itérative. Ainsi, un petit nombre de multiplieurs peut être suffisant pour calculer un filtre contenant un nombre important de coefficients, à condition que le débit ne soit pas trop élevé. Un compromis doit être trouvé entre le nombre d'itérations possibles par multiplieur (le nombre de données et de coefficients à stocker) et la puissance de calcul disponible (le nombre de multiplieurs utilisables).

Les solutions privilégiant la puissance de calcul [69, 70, 73] permettent jusqu'à 4 itérations par multiplieur avec des données et des coefficients stockés dans des registres ou des bancs de registres de petites tailles répartis autour de chaque multiplieur. Ces solutions découlent des circuits configurables pour le filtrage avec comme intention de profiter au maximum des multiplieurs de plus en plus rapides que la technologie mets à leur disposition.

Les solutions privilégiant le nombre d'itérations (la taille des filtres) [71, 72, 74] utilisent un nombre réduit de multiplieurs et des RAM pour stocker les données et les coefficients au moindre coût pour pouvoir en stocker le plus possible. Ces solutions découlent des processeurs de traitement du signal avec l'utilisation d'une partie mémoire et d'une partie opérative séparée mais des possibilités réduites au simple calcul de filtre.

Ces réalisations peuvent être utilisées dans beaucoup d'applications grâce à leur propriété de cascadabilité. Grâce aux itérations, ces processeurs peuvent réaliser des filtres ayant jusqu'à 256 coefficients [71] alors que les circuits configurables n'atteignent que 64 coefficients [45].

II.4.4 Caractéristiques des circuits programmables de filtrage

La plupart des circuits programmables de filtrage sont conçus pour calculer un seul type de filtre, mais de manière efficace. Ils ont pour fonction de calculer uniquement des filtres interpolateurs [52], des filtres adaptatifs [70, 74] ou des filtres symétriques [47, 57, 75]. Seuls quelques circuits ont une certaine flexibilité qui leur permet de supporter plusieurs types d'opérations de filtrage [45, 72] *(Tableau II.1)*.

	FIR	+	-	˜	$\mathbb{C}$	$\downarrow$	FFT	//
Grayver (UCLA), 1998 [45]	X			X	X		X	X
Chen (University of Taiwan), 1998 [46]	X							X
AT76C002 (ATMEL), 1996 [75]	X	X				X		X
Casagrande (SGS-THOMSON), 1993 [57]	X	X						X
Reuver (University of Berlin), 1992 [51]	X							
Hatamian (ATT), 1990 [47]	X	X	X			X		X
Evans (Princeton University), 1990 [63]	X							
Hartley (General Electric), 1989 [56]	X							X
Stearns (LSI LOGIC), 1988 [50]	X							X
Noll (SIEMENS), 1987 [54]	X							
Denk (LUCENT), 1998 [69]	X					X		X
Nicol (LUCENT), 1997 [70]	X			X				
AT76C001 (ATMEL), 1996 [71]	X							X
Kwentus (UCLA), 1992 [72]	X	X	X		X	X		
Cai (LSI LOGIC), 1990 [73]	X					X	X	X
Borth (MOTOROLA), 1988 [74]	X			X				X

TABLEAU II.1: *Fonctionnalités des circuits*

+ : filtres symétriques, - : filtres antisymétriques
˜ : filtres adaptatifs, $\mathbb{C}$: filtres complexes
$\downarrow$: filtres avec décimation, // : cascadabilité

Le *tableau II.1* résume l'ensemble des fonctionnalités de ces circuits qui peuvent être utilisées dans la démodulation.

Parmi les circuits les plus flexibles se trouve [45] qui est un circuit configurable spécialement conçu pour les communications numériques. Il est ainsi capable de réaliser la plupart des fonctions de filtrage des applications de démodulation (filtres RIF réels ou complexes, filtres adaptatifs, transformée de Fourier).

Ce circuit, fonctionnant à 60 MHz, est constitué de 64 unités de calcul, chaque unité se composant principalement d'un multiplieur et d'un additionneur pour calculer la contribution d'un coefficient au calcul d'un filtre. Ce circuit développe donc une puissance de calcul de 7,7 Gops, ce qui devrait être suffisant pour effectuer l'ensemble des calculs de filtrage des applications de démodulation étudiées au chapitre I. Le seul défaut de ce circuit réside dans son incapacité à réaliser plusieurs de ces fonctions sur la même instance du circuit, ce qui est pourtant nécessaire.

Il serait néanmoins possible de réaliser le filtrage par bloc et de reconfigurer le circuit pour pouvoir réaliser un filtre différent sur un bloc de données différent. Malheureusement cette solution ne peut pas convenir aux contraintes de latence imposées par les applications que l'on cherche à réaliser. Par conséquent, pour utiliser ce type de circuit en démodulation, il faut en prévoir autant qu'il y a de fonctions à réaliser,

quelle que soit la puissance de calcul de la fonction en question. Mais, dans ces applications, un maximum de 8 filtres est possible, avec les filtres anti-repliement, les filtres de l'interpolateur, le filtre de Nyquist et l'égaliseur. Il faudrait donc, en pratique, utiliser 8 circuits comme celui-ci alors qu'un seul suffit si l'on ne considère que les puissances de calcul.

Il faudrait en fait un bloc de filtrage qui puisse réaliser plusieurs filtres appartenant à la même application afin d'utiliser au maximum la puissance de calcul développée par ce bloc. Ainsi, le nombre de blocs utilisés dépendra uniquement de la puissance de calcul globale voulue et non du nombre de filtres présents dans les applications. C'est la définition d'un tel bloc qui fait l'objet du chapitre suivant.

Chapitre III

Une solution aux problèmes de filtrage : le CCP

Comme nous l'avons vu dans le chapitre I, le filtrage à réponse impulsionnelle finie (RIF) est l'élément clé du processus de démodulation (filtre de Nyquist, filtres anti-repliement). Il faut donc s'attacher à réaliser efficacement ces algorithmes de filtrage. C'est le but du coprocesseur de filtrage (CCP : "Communication Co-Processor") que nous avons développé [76–78].

La description de ce coprocesseur commencera par les spécifications requises par les applications de démodulation numérique. Ensuite, l'architecture interne ainsi que l'estimation de la complexité matérielle du CCP seront présentées.

III.1 Spécifications du "Communication Co-Processor"

III.1.1 Fonctions requises

Les applications à réaliser étant bien définies, nous en avons déduit les types de filtres qui doivent être réalisée par le CCP. Mais deux fonctions supplémentaires, de la même famille que les filtres et qui sont utilisées dans la démodulation, ont également avantage à être intégrées dans le CCP : la multiplication complexe et la transformée de Fourier.

Les filtres utilisés dans les applications de démodulation ont été décrits à la *section I.2*. Il s'agit de filtres à réponse impulsionnelle finie (RIF) standards, de filtres symétriques et antisymétriques, de filtres adaptatifs et de filtres à décimation. Dans la description de tous les filtres, les données et les coefficients sont indifféremment réels ou complexes.

III.1.1.a Filtres RIF standards

Le filtre RIF standard est la forme la plus générale du filtre RIF *(Équation III.1)*. Les autres types de filtres (symétriques, décimation) ne sont que des filtres RIF ayant certaines spécificités qui permettent de réduire la quantité effective de calculs à réaliser. Par conséquent, il s'agit du filtre à supporter en priorité.

$$y(n) = \sum_{k=0}^{L-1} x(n-k) \times w_k \tag{III.1}$$

où $x(n)$, $y(n)$ représentent respectivement la donnée d'entrée et le résultat au cycle n. w_k est le $k^{ième}$ coefficient du filtre de longueur L. Ainsi, L données et L coefficients sont nécessaires pour calculer un résultat.

III.1.1.b Filtres symétriques

Les filtres symétriques sont des filtres dont les coefficients sont symétriques, ce qui permet d'envisager des gains importants sur le nombre de multiplications et de registres nécessaires.

En effet, $\lceil \frac{L}{2} \rceil$ coefficients, $\lceil \frac{L}{2} \rceil$ multiplications et $L-1$ additions sont nécessaires pour réaliser un filtre symétrique de longueur L. $\lceil \frac{L}{2} \rceil$ est l'arrondi à l'entier supérieur de $\frac{L}{2}$. L'équation d'un filtre symétrique s'écrit :

$$y(n) = \sum_{k=0}^{\lceil \frac{L}{2} \rceil - 1} (x(n-k) + x(n-L+k+1)) \times w_k \tag{III.2}$$

Cette équation s'applique totalement aux filtres pairs (L pair). Pour les filtres impairs, une autre équation est généralement utilisée :

$$y(n) = \sum_{k=0}^{\frac{L-3}{2}} (x(n-k) + x(n-L+k+1)) \times w_k + x(n - \frac{L-1}{2}) \times w_{\frac{L-1}{2}} \tag{III.3}$$

Celle-ci sépare la contribution du coefficient central ($w_{\frac{L-1}{2}}$) : si l'on regarde la contribution de ce ce coefficient dans l'*équation III.2*, on obtient

$$(x(n - \frac{L-1}{2}) + x(n - \frac{L-1}{2})) \times w_{\frac{L-1}{2}}$$

au lieu de $x(n - \frac{L-1}{2}) \times w_{\frac{L-1}{2}}$. Il suffit donc d'utiliser un coefficient central réduit de moitié par rapport au coefficient central utilisé habituellement pour que l'*équation III.2* soit correcte. Cela permet de n'utiliser qu'une seule équation, l'*équation III.2*, au lieu de deux dès lors que les filtres symétriques sont concernés.

La mise en œuvre directe de l'*équation III.2* donne une structure comme celle de la *figure III.1*. Les données $x(n-k)$ sont appelées les données "aller" alors que les données $x(n-L+k+1)$ sont les données "retour" parce qu'elles circulent dans le sens inverse.

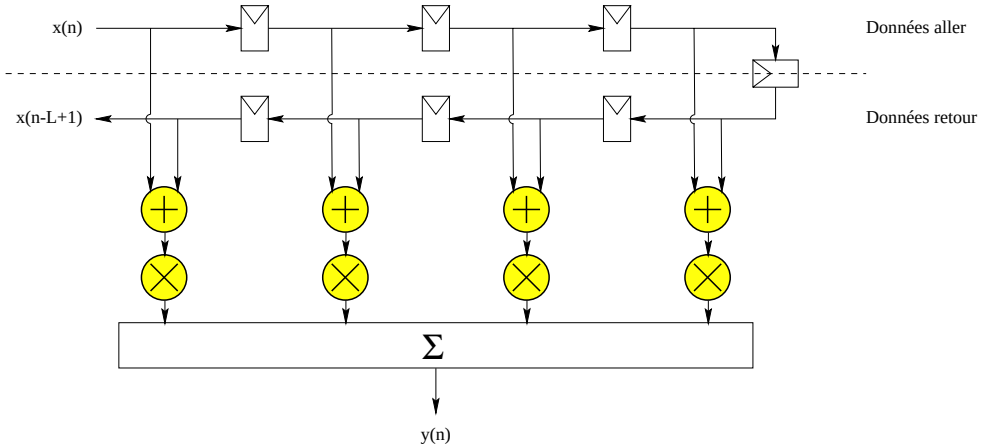


FIGURE III.1 : *Réalisation de filtres symétriques*

III.1.1.c Filtres antisymétriques

Les filtres dits antisymétriques sont des filtres dont les coefficients sont antisymétriques. Les coefficients situés de part et d'autre du coefficient central du filtre sont donc opposés alors qu'ils étaient égaux pour les filtres symétriques. Ce type de filtre utilise donc la même structure que les filtres symétriques tout en remplaçant une partie des additionneurs par des soustracteurs. L'équation d'un filtre antisymétrique s'écrit :

$$y(n) = \sum_{k=0}^{\lceil \frac{L}{2} \rceil - 1} (x(n-k) - x(n-L+k+1)) \times w_k \qquad \text{(III.4)}$$

III.1.1.d Filtres adaptatifs

Il s'agit de filtres qui peuvent suivre les changements du canal de transmission en mettant à jour leurs coefficients afin de minimiser le signal d'erreur. Il existe plusieurs algorithmes pour mettre à jour les coefficients.

Comme le seul filtre adaptatif qui doit être réalisé par notre architecture est l'égaliseur du câble présenté *section I.2.4*, l'algorithme d'adaptation des coefficients mis en

œuvre dans le coprocesseur de filtrage sera celui correspondant à ce filtre, à savoir la version signée de l'algorithme LMS (Least Mean Square) qui est un algorithme fréquemment utilisé.

Néanmoins, dans le cadre de la réalisation d'un "démodulateur universel" un coprocesseur de filtrage plus général utilisant directement l'algorithme LMS devrait être étudié. Il en résulterait soit une complexité accrue pour le processeur de filtrage, un multiplieur complet supplémentaire étant requis pour l'adaptation des coefficients, soit une réduction de ses performances, les mêmes multiplieurs réalisant à la fois les calculs des résultats et les mises à jours des coefficients [74].

III.1.1.e Filtres avec décimation

On ne considère ici que les décimations où le rapport de décimation est un nombre entier. La décimation d'un signal consiste à ne conserver qu'un nombre réduit d'échantillons de ce signal. Cela correspond donc à réduire la fréquence des échantillons. Un filtre avec décimation est un filtre suivi d'une étape de décimation, ce qui veut dire qu'une partie des résultats calculés par le filtre ne sera en fait pas pris en compte. Il n'est donc pas nécessaire de la calculer. De cette façon, la puissance de calcul requise pour mettre en œuvre un filtre avec décimation est réduite par rapport à celle requise pour le même filtre sans décimation. L'équation d'un filtre avec décimation s'écrit :

$$y(n) = \sum_{k=0}^{L-1} x(dn-k) \times w_k \tag{III.5}$$

où d est le facteur de décimation.

III.1.1.f Équation générale d'un filtre

En regroupant les équations des différents types de filtres, on peut obtenir une équation générique pour la partie calcul du résultat :

$$y(n) = \sum_{k=0}^{l-1} (x(dn-k) + \Delta \times x(dn-L+k+1)) \times w_k(n) \tag{III.6}$$

avec $\Delta = +1, -1, 0$ en fonction du type du filtre (respectivement symétrique, antisymétrique ou standard) et l le nombre de coefficients différents utilisés dans les calculs.

III.1.1.g Multiplication complexe

Elle peut être considérée comme un filtre complexe d'un coefficient. Et comme les multiplications complexes, qui correspondent à 4 multiplications et 2 additions

réelles, représentent plus de 600 Mops dans la démodulation satellite *(Figure I.13)*, elles constituent une fonction importante de l'application. C'est pourquoi il est souhaitable qu'elles soient supportées par le CCP. D'autant plus, qu'elles sont aussi utilisées dans l'égaliseur de la démodulation câble puisqu'il s'agit d'un filtre complexe.

III.1.1.h Transformée de Fourier

La transformée de Fourier représente l'élément principal d'une démodulation multiporteuse. Comme sa structure n'est pas très éloignée de celle des filtres *(équation III.7)*, il est aisément envisageable d'utiliser une structure matérielle identique pour réaliser à la fois les filtres RIF et les transformées de Fourier. En effet, il s'agit dans les deux cas d'effectuer des sommes de produits.

$$y(n) = \sum_{k=0}^{L-1} x(k) e^{-\frac{2j\pi kn}{L}}, n = 0...L-1 \tag{III.7}$$

III.1.2 Fonctionnalités requises

L'analyse des besoins des applications cibles en terme de filtrage, réalisée à la *section I.5*, nous amène aux fonctionnalités suivantes.

III.1.2.a Multiplexage de filtres [79]

Dans les applications étudiées, plusieurs fonctions, demandant des puissances de calcul différentes, doivent être réalisées dans un nombre minimal de composants afin d'optimiser leur utilisation. En particulier, il doit être possible de multiplexer des filtres hétérogènes, par exemple, dans le câble, un filtre symétrique semi-complexe (données complexes, coefficients réels) avec décimation de 17 coefficients à 14,46 MHz (filtre anti-repliement) avec un filtre antisymétrique semi-complexe de 2 coefficients à 7,23 MHz (un filtre de l'interpolateur). Cette fonctionnalité ne se trouve pas dans les processeurs de filtrage existant et constitue donc l'originalité de ce composant spécifique.

III.1.2.b Calcul itératif

Le calcul itératif d'un filtre est la caractéristique principale des processeurs de filtrage. Il consiste à calculer le résultat d'un filtre en plusieurs cycles. La partie opérative du processeur est réutilisée pour calculer un résultat de manière itérative. Cela permet de réduire le nombre d'opérateurs nécessaires pour calculer un filtre.

III.1.2.c Cascadabilité

De plus, pour être en mesure d'exécuter des filtres demandant une importante puissance de calcul, il doit être possible de répartir le calcul de différentes parties d'un même filtre sur différents CCP, l'un des CCP servant alors d'interface avec l'extérieur pour recevoir les nouvelles données et sortir les résultats.

La cascadabilité du filtre a été étudiée au LEP par d'autres personnes. La description de cette fonctionnalité n'est donc pas incluse dans le document.

III.1.2.d Parallélisme de sous-mots

Alors que les processeurs sont conçus avec un chemin de données d'une largeur fixe (16, 32 bits par exemple), les applications qu'ils exécutent peuvent se contenter d'utiliser des calculs avec une précision plus faible (8, 16 bits). Ainsi, plutôt que de sous-utiliser le matériel (données sur 8 bits, chemin de données sur 16 bits par exemple), il est possible de regrouper plusieurs données afin qu'elles n'utilisent qu'un seul mot processeur ce qui permet d'optimiser l'efficacité du processeur. La complexité des opérateurs s'en trouve augmentée pour qu'ils puissent supporter ces nouvelles opérations. Cette approche est déjà largement utilisée dans les processeurs intégrant des instructions multimedia [80].

III.1.2.e Type des données

En plus de la dynamique configurable des données, sur 8 ou 16 bits, les données peuvent être réelles ou complexes. Cela accroît encore le nombre de possibilités pour les opérations et donc la complexité des opérateurs.

III.2 Conception de l'architecture du CCP

Comme tout processeur, le CCP *(Figure III.2)* est composé de 3 parties principales :

- la partie opérative qui va définir les possibilités du processeur,
- la mémoire qui va stocker les données d'entrée, les coefficients et les calculs intermédiaires,
- le contrôleur qui devra fournir les bonnes instructions à chaque élément du processeur.

À côté de ces blocs principaux, nous retrouverons un mécanisme de mise à jour des coefficients pour les filtres adaptatifs *(Section III.2.2.e)*. Nous verrons également que, pour faire le lien entre les données lues par blocs en mémoire et la partie opérative, un mécanisme de réorganisation est utilisé *(Section III.2.4)*.

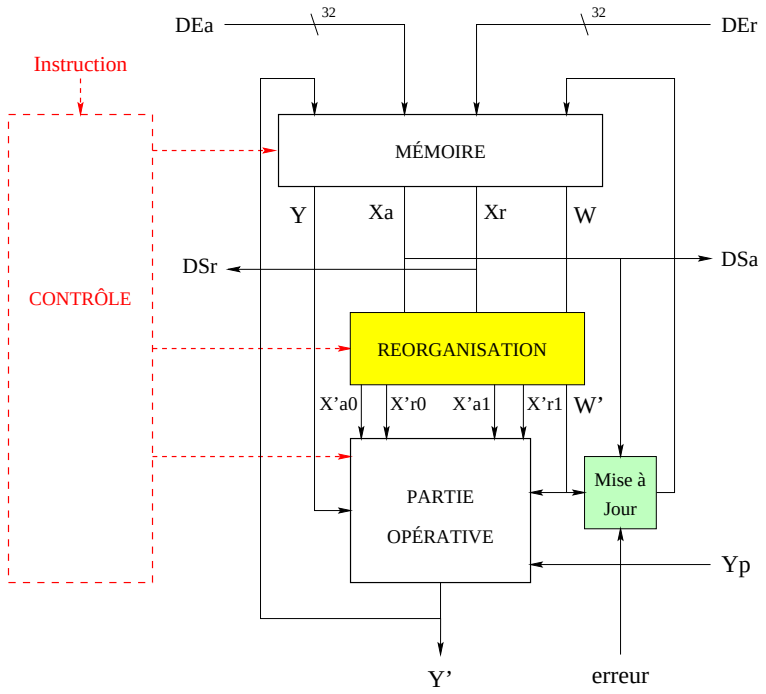


FIGURE III.2 : *Architecture générale du CCP*

Dans la *figure III.2*, DEa est le port d'entrée des données sur la voie aller, c'est-à-dire les données que reçoit le coprocesseur. Comme les données peuvent être complexes avec une précision de 16 bits, la taille du bus utilisé est de 32 bits pour l'entrée et la sortie des données. DEr, DSr, DSa et Yp sont utilisés lorsque les coprocesseurs sont mis en cascade. DEr est le port d'entrée des données sur la voie retour venant du CCP de droite, DSa est le port de sortie des données sur la voie aller et DSr est le port de sortie des données sur la voie retour. Yp est le port d'entrée des résultats partiels venant du CCP de droite et qui vont être ajoutés aux calculs effectués dans ce processeur pour obtenir le résultat final.

Mais avant de décrire en détail l'architecture du CCP, je vais commencer par une présentation rapide de l'architecture globale afin de définir les notations utilisées dans ce chapitre.

III.2.1 Notations

Le calcul d'un filtre à partir de son équation la plus simple comme l'*équation III.1* semble assez simple. Mais lorsque l'on utilise un certain degré de parallélisme dans les

calculs pour accélérer la cadence des traitements, la représentation du séquencement des calculs devient plus difficile.

Une façon de représenter l'algorithme de filtrage est d'utiliser un graphe de dépendance comme ceux utilisés dans [81]. La *figure III.3* présente un graphe de dépendance du calcul d'un filtre. Elle permet de donner une représentation graphique de l'ensemble des calculs que réalise le processeur de filtrage et de présenter les relations entre les différentes notations utilisées dans ce chapitre.

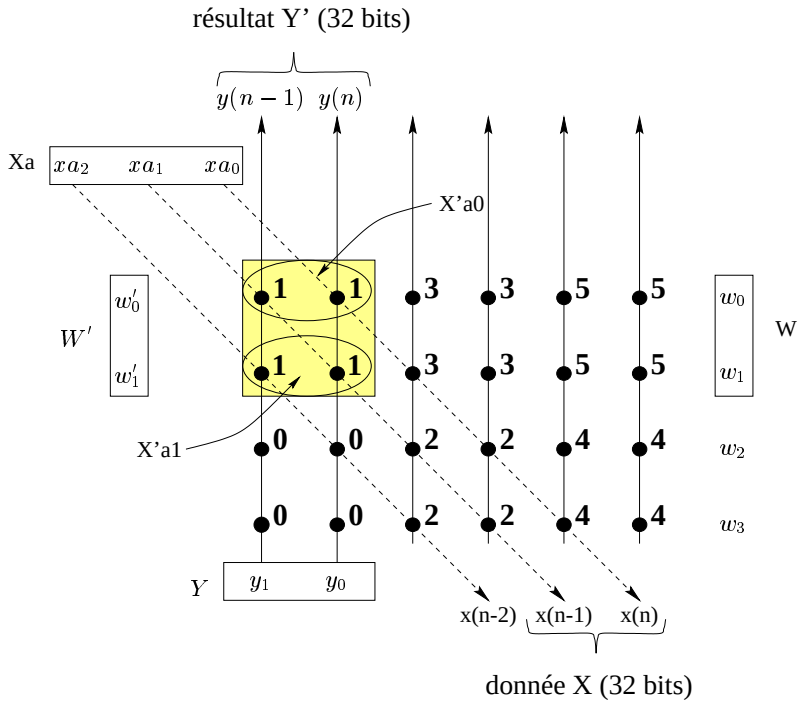


FIGURE III.3 : *Graphe de dépendance pour un filtre réel de 4 coefficients sur 16 bits ($n_c = 2$, $p_y = 2$)*
Un point correspond à la contribution d'un coefficient au calcul d'un filtre. Les nombres situés au dessus à droite de ces points indiquent l'instant t du début du calcul de cette contribution. Le rectangle grisé contient l'ensemble des opérations calculées lors d'une itération (à un même instant t). Les flèches verticales représentent la progression du calcul des résultats tandis que les flèches en diagonale représentent la propagation des données, ou plutôt, la liste des instants où sont utilisées les données.

w_k représente le $k^{ième}$ coefficient du filtre.

w'_l représente le $l^{ième}$ coefficient utilisé lors d'une itération.

n_c représente le nombre de coefficients utilisés à chaque itération.

W' représente le vecteur de coefficients contenant les n_c coefficients utilisés.

$x(n)$ correspond à la $n^{ième}$ donnée reçue par le filtre.

xa_i correspond à la $i^{ième}$ donnée consécutive lue lors du calcul d'un filtre à un instant donné (une itération), xa_0 étant la donnée la plus récente des données xa_i.

Xa est le vecteur de données contenant les xa_i.

$X'a_l$ est, à chaque itération, le vecteur de données multiplié par le coefficient w'_l.

C'est la même chose pour les données de la voie retour, excepté qu'elles sont indiquées avec un indice r au lieu de a.

$y(n)$ est le résultat du filtre correspondant à $x(n)$, c'est-à-dire que parmi les données utilisées pour calculer $y(n)$, $x(n)$ est la plus récente.

Y est le vecteur de résultats intermédiaires qui va être utilisé pour obtenir le nouveau vecteur de résultats intermédiaires Y'.

p_y est le nombre de résultats dans un vecteur résultat.

III.2.2 La partie opérative

La partie opérative du coprocesseur de filtrage doit être conçue pour supporter l'ensemble des types de fonctions (filtres et autres) qui ont été définies dans ses spécifications.

III.2.2.a Un opérateur de filtrage

Comme il a été montré précédemment, l'une des principales contraintes pour réaliser des filtres pour la démodulation numérique est la latence (le temps qui s'écoule entre l'arrivée d'une donnée et la sortie du résultat correspondant). En effet, plus les latences sont faibles, plus les boucles de contre-réaction peuvent réagir rapidement et corriger les erreurs de synchronisation.

Prenons l'équation générique d'un filtre RIF *(équation III.6)* et modifions-la légèrement pour laisser apparaître deux parties distinctes :

$$y(n) = (x(dn) + \Delta \times x(dn - L + 1)) \times w_0 + \sum_{k=1}^{L-1} (x(dn - k) + \Delta \times x(dn - L + k + 1)) \times w_k \quad \text{(III.8)}$$

La deuxième partie de l'équation ne comporte que des données déja présentes lors du calcul de $y(n-1)$. Ainsi, sur les L additions (pour les filtres symétriques), les L multiplications et les $L-1$ accumulations qui doivent être exécutées pour obtenir un résultat, seulement une addition, une multiplication et une accumulation séparent l'arrivée d'une donnée de la sortie d'un résultat.

Ainsi, afin d'obtenir une latence minimale pour le calcul des filtres, il faut pouvoir calculer la contribution d'un coefficient, quelle qu'il soit, en une seule itération. Le pire cas, en complexité, est un coefficient complexe dans un filtre complexe symétrique avec une précision maximale des données et des coefficients (16 bits).

Filtre complexe 16 bits

La partie opérative calcule donc la contribution d'un coefficient, $w' = (w'^R, w'^I)$ où w'^R et w'^I sont respectivement la partie réelle et la partie imaginaire de w', pour un résultat, y' :

$$y'^R = y^R + w'^R \times (xa_0^R + xr_0^R) - w'^I \times (xa_0^I + xr_0^I) \quad \text{(III.9a)}$$
$$y'^I = y^I + w'^R \times (xa_0^I + xr_0^I) + w'^I \times (xa_0^R + xr_0^R) \quad \text{(III.9b)}$$

Par conséquent, on obtient une unité de calcul *(Figure III.4)*, qui en mode 16 bits, devrait contenir :

- 2 additionneurs pour l'addition complexe de 2 données pour le filtre symétrique,
- 4 multiplieurs, un additionneur et un soustracteur pour la multiplication complexe,
- 2 accumulateurs pour l'accumulation complexe.

En fait, cette partie opérative comporte encore deux additionneurs supplémentaires qui effectuent deux fois l'addition complexe requise par l'optimisation pour les filtres symétriques. Ces additionneurs supplémentaires sont en fait requis pour pouvoir utiliser au maximum les multiplieurs lorsque les coefficients ne sont pas complexes, comme pour un filtre réel symétrique avec une précision de 16 bits *(Figure III.5)*. L'utilisation, pour réaliser les filtres complexes, de ces deux additionneurs redondants permet de conserver inchangées les connexions entre les différents opérateurs de la partie opérative.

Filtre réel 16 bits

Pour un tel filtre, 2 résultats sont calculés par cycle puisque le chemin de données est dimensionné pour faire passer une donnée complexe, ce qui correspond à 2 données réelles. Pour utiliser 4 multiplieurs dans le calcul de 2 résultats, il est donc nécessaire de calculer la contribution de 2 coefficients pour chaque résultat :

$$y'_0 = y_0 + w'_0 \times (xa_0 + xr_1) + w'_1 \times (xa_1 + xr_0) \quad \text{(III.10a)}$$
$$y'_1 = y_1 + w'_0 \times (xa_1 + xr_2) + w'_1 \times (xa_2 + xr_1) \quad \text{(III.10b)}$$

On peut remarquer qu'il faut en effet 4 additionneurs pour effectuer les 4 additions supplémentaires requises pour le filtre symétrique *(Figure III.5)*.

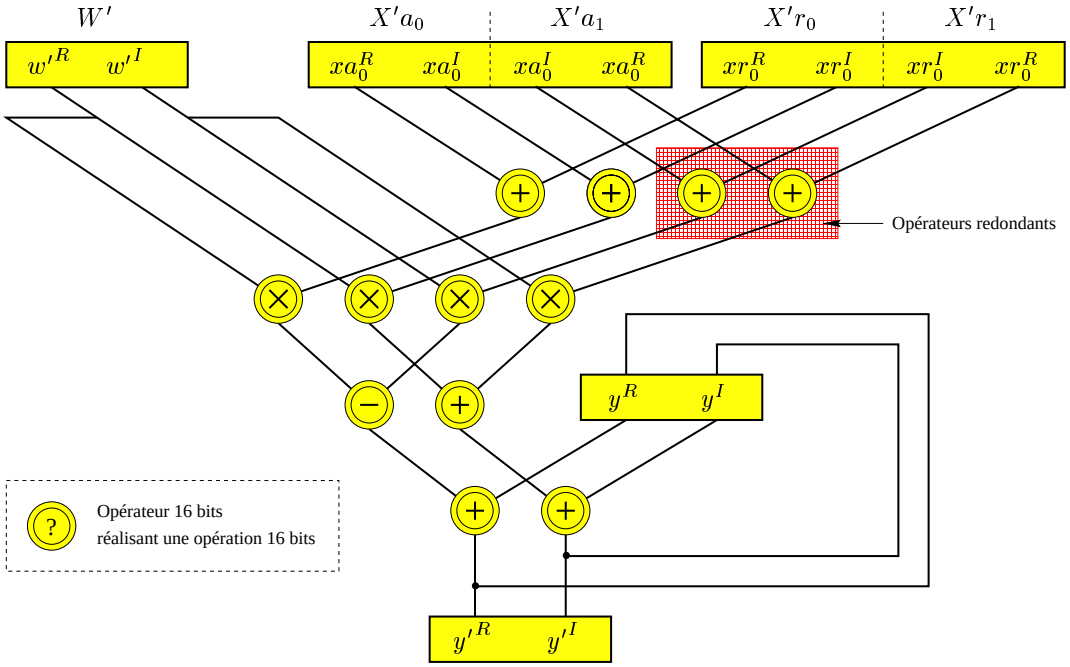


FIGURE III.4 : *Filtre complexe 16 bits ($p_y = 1$, $n_c = 1$)*
Structure de la partie opérative pour réaliser un filtre complexe symétrique avec une précision de 16 bits pour les données et les coefficients. Les opérateurs (multiplications, additions, soustractions) sont entourés par 2 cercles concentriques pour indiquer qu'il ne s'agit pas de simples opérateurs. En effet, ils peuvent également être utilisés pour effectuer des calculs avec des données ayant une précision plus petite (8 bits).

Filtre semi-complexe 16 bits

En mode 16 bits, entre le filtre complexe et le filtre réel, il reste le filtre semi-complexe, c'est-à-dire un filtre avec des données complexes et des coefficients réels. Il s'agit donc de réaliser deux filtres réels utilisant les mêmes coefficients mais des données différentes. Ce filtre a besoin de 2 multiplieurs par coefficient, et comme un seul résultat est calculé à chaque cycle (données complexes), 2 coefficients sont nécessaires *(Figure III.6)* pour utiliser pleinement la puissance de calcul offerte par le CCP. Il en résulte l'équation suivante :

$$y'^R = y^R + w'_0 \times (xa_0^R + xr_1^R) + w'_1 \times (xa_1^R + xr_0^R) \quad \text{(III.11a)}$$
$$y'^I = y^I + w'_0 \times (xa_0^I + xr_1^I) + w'_1 \times (xa_1^I + xr_0^I) \quad \text{(III.11b)}$$

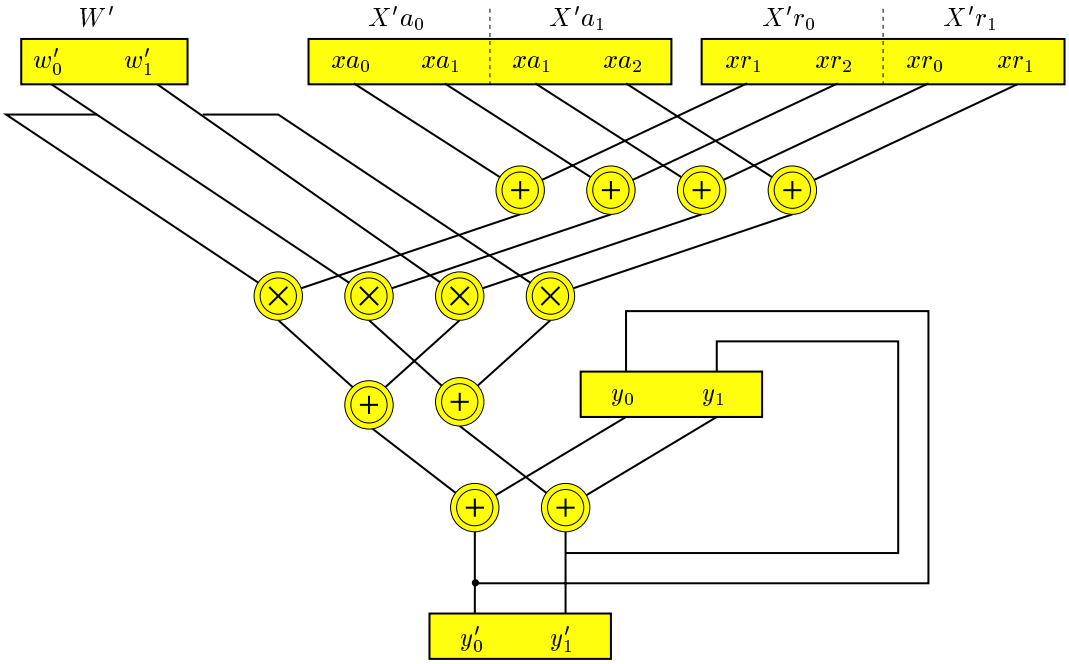


FIGURE III.5 : *Filtre réel 16 bits ($p_y = 2$, $n_c = 2$)*
Structure de la partie opérative pour réaliser un filtre réel symétrique avec une précision de 16 bits pour les données et les coefficients.

Filtre 8 bits

Pour des données sur 8 bits, le nombre d'opérateurs est doublé puisqu'un opérateur 16 bits peut être utilisé comme deux opérateurs 8 bits. Mais le nombre de données dans un mot de résultat est également doublé puisque les données occupent deux fois moins de place. Il en résulte que le nombre de coefficients utilisés ne change pas en fonction de la précision des données *(Figure III.7)*.

Équation générale de l'opérateur de filtrage

Tous les exemples abordés ont pris en compte un filtre symétrique. Or, un certain nombre d'autres types de filtres doit être supporté :

- les filtres antisymétriques, qui requièrent la réalisation de soustractions à la place des additions effectuées avant les multiplieurs,
- les filtres RIF standards, qui n'utilisent pas ces additionneurs/soustracteurs, ce qui correspond à utiliser des valeurs nulles pour les données xr_i.
- les filtres avec décimation, qui n'ont aucune influence sur la structure de la partie opérative.

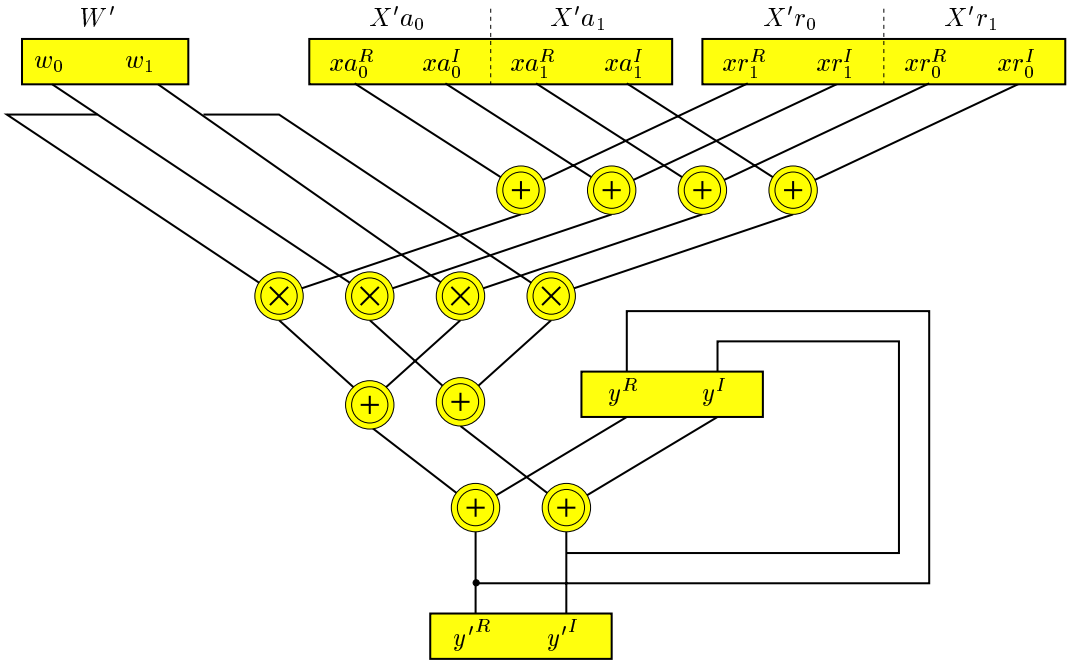


FIGURE III.6 : *Filtre semi-complexe 16 bits ($p_y = 1$, $n_c = 2$)*
Structure de la partie opérative pour réaliser un filtre semi-complexe (données complexes, coefficients réels) symétrique avec une précision de 16 bits pour les données et les coefficients.

Finalement, l'ensemble des équations que peut réaliser la partie opérative peut être résumé en une équation de la forme :

$$Y' = Y + w'_0 \times (X'a_0 + \Delta \times X'r_0) \pm w'_1 \times (X'a_1 + \Delta \times X'r_1) \qquad \text{(III.12)}$$

où $\Delta = +1, -1, 0$ en fonction du type de filtre, Y et Y' sont les vecteurs (mots de 32 bits) correspondant aux données y_i et y'_i, $X'a_0$ et $X'r_0$ sont les vecteurs correspondant aux données xa_i et xr_i multipliées par le coefficient w'_0 et $X'a_1$ et $X'r_1$ sont les vecteurs correspondant aux données xa_i et xr_i multipliées par le coefficient w'_1.

La partie opérative obtenue est donc indépendante du type de filtre réalisé dans le sens ou les connexions entre les opérateurs sont fixes, quelque soit le type de filtre utilisé. La dépendance se situe au niveau des opérateurs même qui ont des configurations différentes en fonction du format des données. En effet, la taille des données est nécessaire pour savoir si on utilise les opérateurs 16 bits tels quels ou en tant que doubles opérateurs 8 bits. Le type des données est également nécessaire pour décider si certains opérateurs effectueront une addition ou une soustraction (pour les multiplications complexes et les filtres antisymétriques).

Il a été spécifié dans la *section III.1.1* que d'autres fonctions devaient également être supportées par ce coprocesseur, à savoir les multiplications complexes et les transformées de Fourier. Il faut donc étudier ce que ces fonctions supplémentaires vont apporter comme modifications à la structure de la partie opérative.

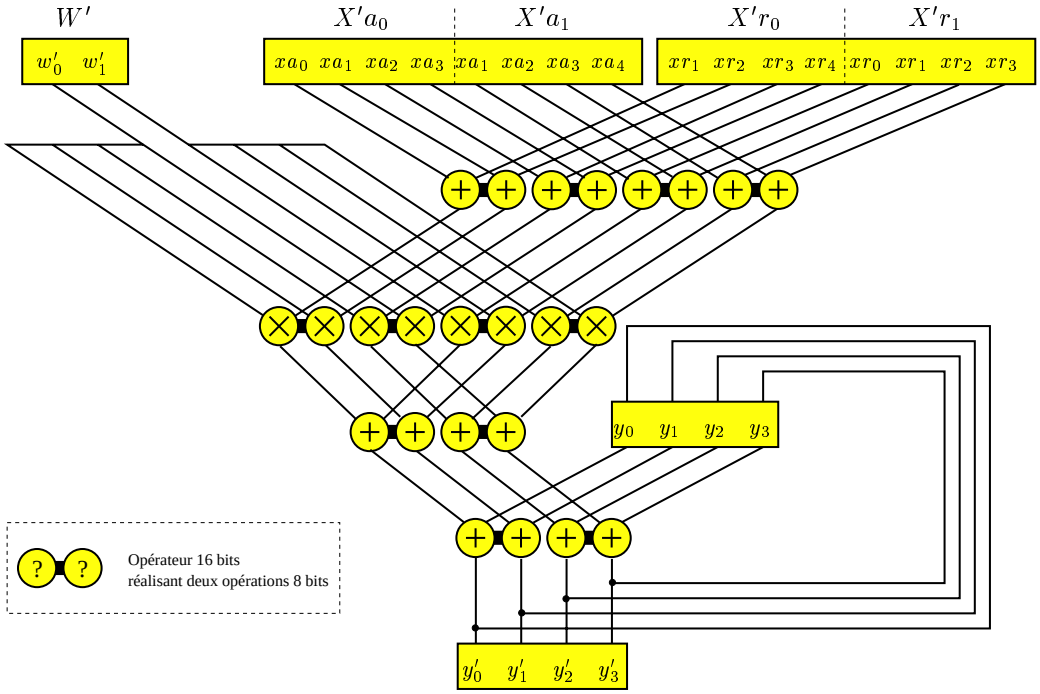


FIGURE III.7 : *Filtre réel 8 bits ($p_y = 4$, $n_c = 2$)*
Structure de la partie opérative pour réaliser un filtre réel symétrique avec une précision de 8 bits pour les données et les coefficients. Les opérateurs 8 bits sont reliés 2 par 2, pour indiquer qu'il s'agit de l'utilisation 8 bits d'un même opérateur 16 bits.

III.2.2.b Multiplication complexe

Le coprocesseur supporte déjà les filtres complexes, et les multiplications complexes sont simplement des filtres complexes ne comportant qu'un seul coefficient complexe. Ainsi, l'*équation III.12* définissant les opérations réalisées par la partie opérative est tout à fait compatible avec la réalisation des multiplications complexes, du moins pour des données complexes sur 16 bits pour lesquelles il n'y a pas de parallélisme de données. Par contre, dans le cas de données sur 8 bits, un filtre complexe calculera la contribution d'un coefficient pour 2 résultats alors qu'une multiplication complexe aura besoin de l'équivalent de 2 "coefficients" pour calculer également 2 résultats.

Il sera donc nécessaire de différencier le calcul d'une multiplication complexe de celui d'un filtre complexe. Cette fonctionnalité supplémentaire demandera seulement d'effectuer une réorganisation sur les coefficients.

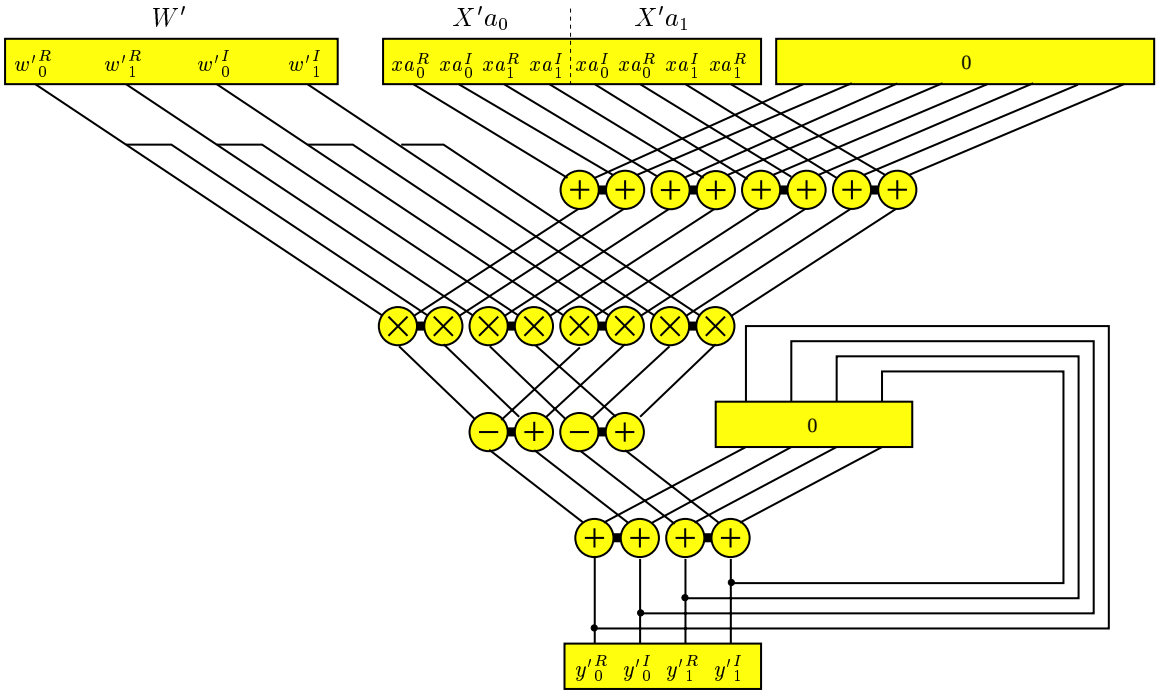


FIGURE III.8 : *Multiplication complexe 8 bits ($p_y = 2$, $n_c = 2$)*
Structure de la partie opérative pour réaliser une multiplication complexe avec une précision de 8 bits pour les données et les coefficients. Les opérateurs 8 bits sont reliés 2 par 2, pour indiquer qu'il s'agit de l'utilisation 8 bits d'un même opérateur 16 bits.

III.2.2.c Transformée de Fourier

La puissance de calcul requise pour réaliser une Transformée de Fourier et ses accès à la mémoire contenant des données et des coefficients est assez importante. Pour réduire ces besoins, des factorisations sont possibles. Elles mènent à ce que l'on appelle une Transformée de Fourier Rapide (FFT : Fast Fourier Transform) [82]. La version de cet algorithme la plus communément utilisée est celle utilisant une décomposition en papillons de radix-2. La *figure III.9* représente la structure d'un tel papillon. En utilisant $\frac{L}{2}\log_2 L$ fois cette structure avec les coefficients et les données appropriés, on obtient le résultat attendu.

Or, le papillon correspond, en partie, à une multiplication et une addition complexe ce qui est exactement une des possibilités de la partie opérative. Par contre, une soustraction complexe supplémentaire est également nécessaire, ainsi que la génération de 2 résultats au lieu d'un seul.

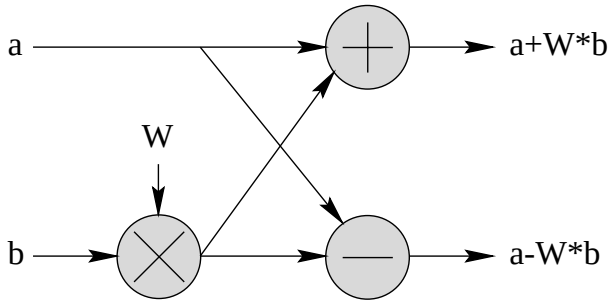


FIGURE III.9 : *Structure du papillon de radix-2 (Opérations complexes)*

III.2.2.d Structure complète de la partie opérative

Toutes les configurations possibles de la partie opérative ont donc maintenant été étudiées. Celle-ci est divisible en 3 parties successives *(Figure III.10)*.

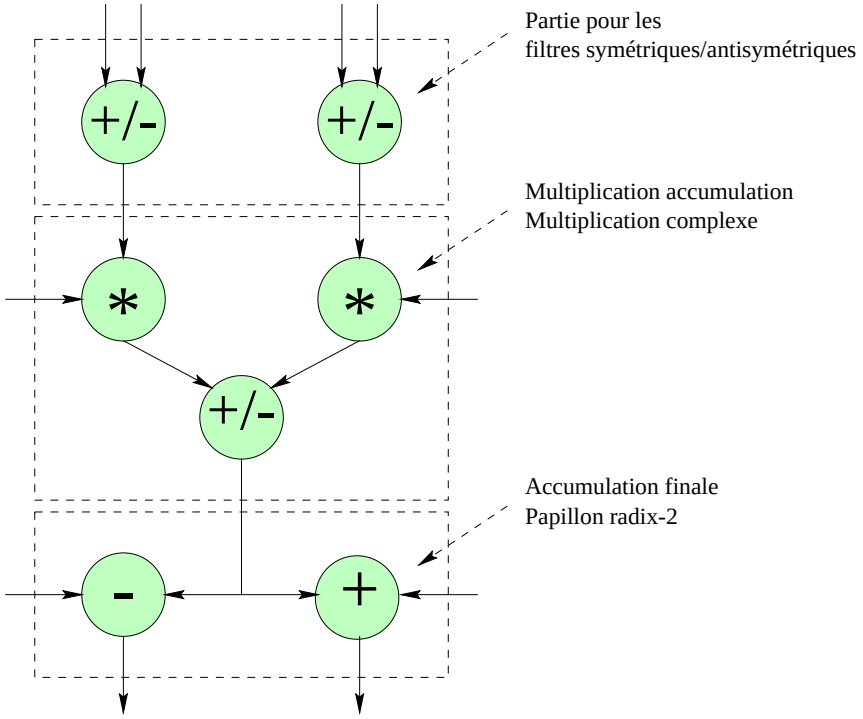


FIGURE III.10 : *Structure finale simplifiée de la partie opérative*
Chaque opérateur présenté ici exécute des calculs en parallèle sur des vecteurs de données. La précision des données est de 8 ou 16 bits et les vecteurs de données contiennent jusqu'à 32 bits en entrée de la partie opérative.

- La première partie est composée d'additionneurs/soustracteurs qui sont utilisés pour réduire le nombre de multiplieurs nécessaires pour réaliser les filtres symétriques et antisymétriques.

- La deuxième partie est constituée de multiplieurs et d'additionneurs/soustracteurs pour réaliser des sommes de multiplications et des multiplications complexes.
- Enfin, la dernière partie est composée d'additionneurs pour effectuer l'accumulation des résultats, ainsi que de soustracteurs pour réaliser des papillons de FFT.

III.2.2.e Mise à jour des coefficients

Ce bloc met en œuvre l'algorithme présenté par l'*équation I.1* pour des données et des coefficients complexes. Il est donc constitué, comme le montre la *figure III.11*, d'un additionneur/soustracteur pour effectuer la mise à jour, d'un multiplieur complexe 1 bit pour effectuer la multiplication des signes et d'un registre à décalage pour conserver le signe des données en attendant d'obtenir l'erreur et de commencer la mise à jour.

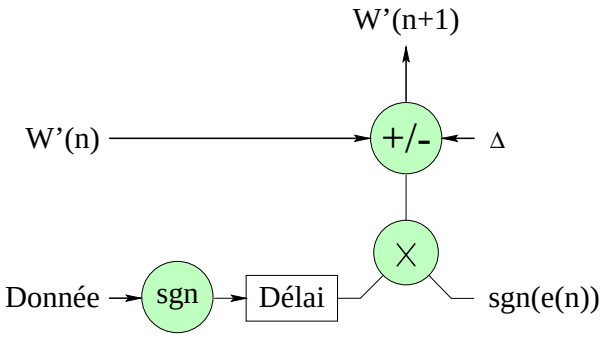


FIGURE III.11 : *Bloc de mise à jour*

La multiplication complexe des signes n'a que quatre résultats possibles qui sont (2,0), (-2,0), (0,2) et (0,-2). Ainsi, selon le résultat de cette multiplication, soit la partie réelle, soit la partie imaginaire du coefficient va être mise à jour, et le pas de correction est 2Δ. Ceci montre bien que le résultat de la multiplication des signes va servir uniquement à configurer l'additionneur/soustracteur. Il n'est pas nécessaire d'effectuer une nouvelle multiplication avec Δ comme pourrait le laisser supposer l'*équation I.1*.

La partie opérative ayant été définie, il faut maintenant dimensionner la bande passante des mémoires afin que les vecteurs de données adéquats soient fournis à cette partie opérative.

III.2.3 La mémoire

Les éléments qui ont besoin d'être mémorisés sont les coefficients et les données (utilisées sur la ligne aller et sur la ligne retour) ainsi que les résultats partiels afin de

pouvoir muliplexer le calcul de plusieurs filtres.

III.2.3.a Mémoire des résultats [79]

Pour pouvoir calculer plusieurs filtres sans perdre de cycles de latence, le processeur de filtrage peut commuter de contexte à tout moment afin d'utiliser les nouvelles données reçues, passant de l'exécution d'un filtre à l'exécution d'un autre. Il faut donc pouvoir conserver le résultat partiel de chacun des filtres calculés dans le processeur.

À côté de cette mémoire conservant les résultats partiels des différents filtres, d'autres mémoires sont nécessaires pour stocker les données et les coefficients. Pour pouvoir fournir les éléments requis à chaque cycle par la partie opérative qui utilise intensivement le parallélisme des calculs, une structure complexe de mémoire est nécessaire.

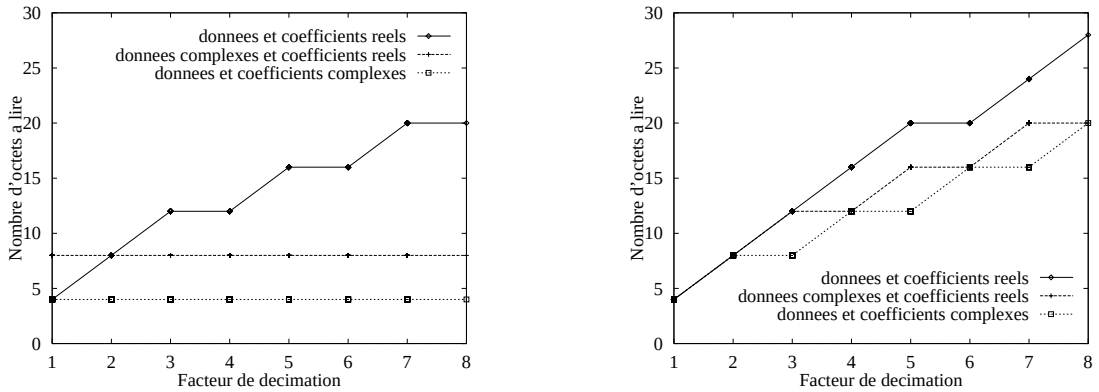
III.2.3.b Nombres de ports d'entrée/sortie

À partir de la *figure III.3* on obtient aisément les données nécessaires à une itération. Prenons un vecteur résultat $Y' = (y(n - p_y + 1), \ldots, y(n))$. Supposons que l'on réalise la dernière itération pour obtenir le résultat. On va déterminer les données nécessaires à ce calcul ou plutôt, la donnée la plus jeune et la donnée la plus ancienne utilisées. $y(n)$ a besoin des n_c données suivantes $(x(dn - n_c + 1), \ldots, x(dn))$ où n_c est le nombre de coefficients utilisés à chaque cycle pour les calculs et d est le facteur de décimation. Quant à $y(n-p_y+1)$, il a besoin de $(x(dn-dp_y+d-n_c+1), \ldots, x(dn-dp_y+d))$. Afin de réduire la complexité du calcul des adresses mémoires, les lectures s'effectuent par blocs de données consécutives. Il faut donc lire en mémoire l'ensemble des données comprises entre $x(dn)$ (donnée la plus récente) et $x(dn-dp_y+d-n_c+1)$ (donnée la plus ancienne). Le nombre de données à lire est donc : $N_d = dn-(dn-dp_y+d-n_c+1)+1 = dp_y-d+n_c$.

Le raisonnement ci-dessus est réalisé pour les données de la voie aller, mais pour les filtres symétriques qui utilisent les données de la voie retour, le calcul est identique.

Comme on l'a vu à la *section III.2.2*, le nombre de coefficients n_c nécessaires est soit 2 (si les coefficients sont réels) soit 1 (s'ils sont complexes). Ainsi, on peut tracer les courbes représentant le nombre d'octets à lire en fonction de la décimation pour chaque type de données manipulées *(Figure III.12)*.

Il apparaît clairement qu'il est coûteux, en terme de nombre de lectures, de permettre des décimations importantes. Mais, comme les besoins des applications cibles se limitent à une décimation par 2, le pire cas correspond à une lecture de 8 octets consécutifs. Ce résultat permet d'envisager des simplifications pour la génération des adresses mémoire ainsi que pour la structure de la mémoire en comparaison de ce qu'il aurait fallu pour la lecture de 8 octets de manière aléatoire (ce qui aurait été nécessaire pour fournir des données aux 8 multiplieurs utilisés en mode 8 bits).



(a) *données sur 16 bits* (b) *données sur 8 bits*

FIGURE III.12 : *Influence du type des données et de la décimation sur la lecture des données de la voie aller dans la mémoire*

III.2.3.c Architecture mémoire [83]

La mémoire nécessaire pour un filtre peut être séparée en 2 ou 3 ensembles selon que le filtre effectué est symétrique ou non. En effet, des blocs de mémoires sont nécessaires pour les coefficients, les données de la ligne aller et, éventuellement, les données de la ligne retour.

En général, les processeurs de filtrage utilisent des blocs de mémoires séparés pour les données et pour les coefficients, afin de pouvoir accéder simultanément aux données et aux coefficients correspondant. Ce n'est toutefois pas optimal puisque la répartition des besoins en mémoire varie substantiellement d'un type de filtre à l'autre. Partons d'une capacité mémoire de N données et étudions comment faire les différents types de filtres les plus longs à partir de ces N cases mémoires. Le *tableau III.1* montre les répartitions possibles, la longueur du filtre correspondant au nombre de coefficients.

	données aller	données retour	coefficients
Filtres RIF réels ou complexes	$N \times \frac{1}{2}$	0	$N \times \frac{1}{2}$
Filtres symétriques réels ou complexes	$N \times \frac{1}{3}$	$N \times \frac{1}{3}$	$N \times \frac{1}{3}$
Filtres RIF semi-complexes	$N \times \frac{2}{3}$	0	$N \times \frac{1}{3}$
Filtres symétriques semi-complexes	$N \times \frac{2}{5}$	$N \times \frac{2}{5}$	$N \times \frac{1}{5}$

TABLEAU III.1: *Répartition des besoins en mémoire des filtres*

Un filtre RIF classique partage équitablement la mémoire entre données et coefficients alors qu'un filtre symétrique semi-complexe va utiliser 4 fois plus de place en mémoire pour les données que pour les coefficients.

Une solution directe serait donc de n'utiliser qu'une seule mémoire pour conserver l'ensemble des données et des coefficients. Toutefois, cette solution implique soit un goulet d'étranglement de la bande passante, soit d'utiliser une mémoire avec 3 ports

d'écriture et 3 ports de lecture. Cette solution n'est pas efficace car la taille d'une mémoire dépend du nombre de données à stocker, mais également, et ce n'est pas négligeable, du nombre de ports d'entrée/sortie [84].

Par conséquent, il peut être intéressant, en fonction de la capacité mémoire nécessaire, de trouver un partitionnement de celle-ci afin de réaliser les différents types de filtres les plus longs. D'après le *tableau III.1*, on remarque que la mémoire de taille N doit pouvoir être considérée comme 2 mémoires identiques (filtres RIF réels), 3 mémoires identiques (filtres symétriques réels) ou 5 mémoires identiques (filtres symétriques semi-complexes). Le plus petit commun multiple de ces 3 nombres étant 30, un partitionnement de la mémoire en 30 petites mémoires identiques donnerait une flexibilité maximale. Parmi ces 30 sous-mémoires, quelques-unes ont toujours le même comportement et peuvent donc être regroupées, ce qui permet de réduire le nombre de mémoires. Un ensemble comprenant 6 mémoires de tailles différentes *(Figure III.13)* est ainsi obtenu. Pour obtenir ce résultat, l'algorithme suivant a été utilisé :

Étape 0 : Génération du tableau de répartition de N cases mémoires en fonction du type de filtres supportés *(Tableau III.1)*.

Étape 1 : Calcul de la capacité d'une mémoire.
La capacité de la mémoire est la plus petite valeur non nulle du tableau, M.

Étape 2 : Choix de l'utilisation de la mémoire.
Pour chaque ligne du tableau (type de filtre), on choisit la colonne comportant la plus petite valeur (type d'élément à stocker).

Étape 3 : Mise à jour du *tableau III.1*.
La valeur trouvée à l'étape 1 est retranchée des valeurs trouvées à l'étape 2. Un nouveau tableau est ainsi obtenu exprimant la répartition des $N - M$ cases mémoires restantes.

Les 3 dernières étapes sont répétées jusqu'à ce que le dernier tableau obtenu ne comporte que des valeurs nulles. L'étape 2 permet également de définir les multiplexeurs décrits à la *figure III.13*.

Ce morcellement de la mémoire demande un générateur d'adresses plus compliqué (pour déterminer quelles mémoires doivent être utilisées en fonction de l'adresse). De plus, comme le coût en surface d'une case mémoire augmente pour des petites mémoires, le partitionnement d'une mémoire en plusieurs petites augmente la surface occupée. Il faut donc faire un compromis entre d'un côté un nombre restreint de mémoires ayant un coût en surface par case intéressant mais un taux d'utilisation des cases réduit, et de l'autre, un partitionnement maximal qui entraîne un coût en surface par case élevé mais un taux d'utilisation des cases maximal.

En fonction des filtres à réaliser, on peut se concentrer sur un nombre moins important de type de filtres. Par exemple, si on ne considère pas les filtres symétriques

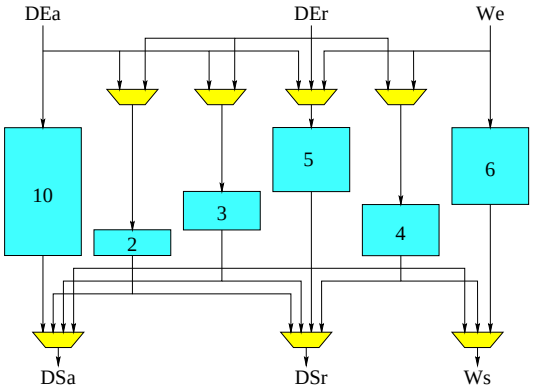


FIGURE III.13 : *Partitionnement maximal*
Les nombres dans chaque bloc correspondent à la capacité de la mémoire correspondante, l'unité étant $\frac{N}{30}$.

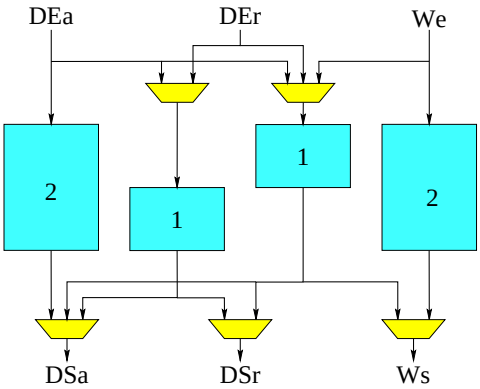


FIGURE III.14 : *Partitionnement partiel*
Les nombres dans chaque bloc correspondent à la capacité de la mémoire correspondante, l'unité étant $\frac{N}{6}$.

semi-complexes (la dernière ligne du tableau) on obtient une structure mémoire un peu plus simple avec 4 blocs seulement *(Figure III.14)*.

Un nombre assez important de partitionnements des mémoires est donc possible en fonction des cibles visées par le CCP. Dans notre cas, les filtres symétriques semi-complexes de petites tailles sont prédominants, ce qui ne justifie pas de fragmenter la mémoire en nombreux petits morceaux. Une solution utilisant donc tout simplement 3 mémoires est retenue, une pour les coefficients, une pour les données de la ligne aller et une pour les données de la ligne retour. Néanmoins, lors de l'implantation matérielle d'un CCP avec des valeurs précises pour la quantité de mémoire requise, il sera envisageable d'etudier les différentes solutions possibles pour choisir la plus compacte.

Comme la partie opérative utilise les données dans un certain ordre qui dépend du type de filtres à réaliser alors que la partie mémoire lit un bloc contigu de données il est nécessaire d'utiliser un bloc de réorganisation qui va servir d'interface entre ces deux parties.

III.2.4 Réorganisation des données

Ce bloc transforme la structure des vecteurs de données lus dans les mémoires pour qu'ils soient bien ordonnés avant d'être transmis à la partie opérative. Il suffit de regarder les organisations de données et de coefficients utilisés selon les types des filtres pour comprendre que cette réorganisation est nécessaire *(Figure III.15)*. Cette figure représente les organisations utilisées dans les filtres pris en exemple dans la *section III.2.2*.

Dans cette section, les équations permettant d'obtenir $X'a_0$ et $X'a_1$ en fonction des

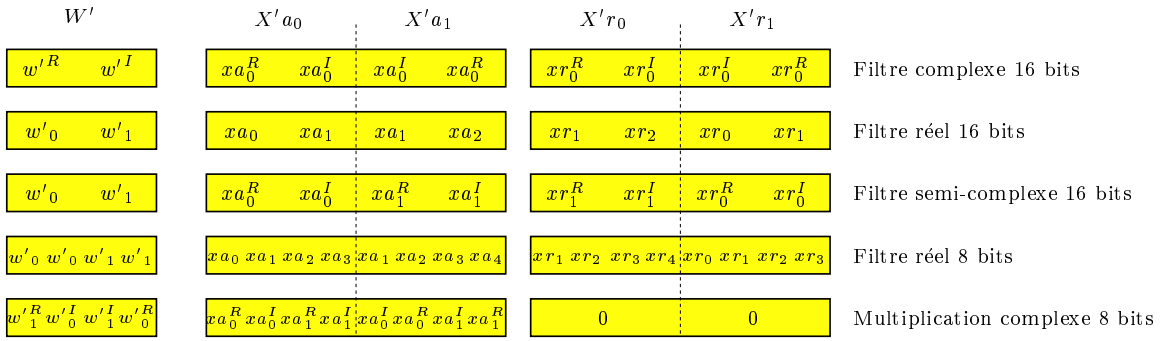


FIGURE III.15 : *organisations des données et des coefficients requises*

données lues dans la mémoire xa vont être définies.

À partir de l'équation générale d'un filtre *(III.6)*

$$y(n) = \sum_{k=0}^{l-1} (x(dn-k) + \Delta \times x(dn-L+k+1)) \times w_k(n)$$

et de l'équation décrivant la partie opérative du CCP *(III.12)*,

$$Y' = Y + w'_0 \times (X'a_0 + \Delta \times X'r_0) \pm w'_1 \times (X'a_1 + \Delta \times X'r_1)$$

on obtient les indices des données requises pour les calculs.

Pour cela on commence par établir la relation qui existe entre les indices des données correspondant à l'équation générale d'un filtre et les vecteurs de données lus dans le banc de registres, en supposant que la première donnée (xa_0) lue dans la mémoire contenant les données de la voie aller soit la donnée la plus récente requise ($x(d \times n - k)$ pour le calcul de la contribution du coefficient w_k au calcul de $y(n)$). Le vecteur Xa est constitué de données consécutives, ce qui conduit à $\boxed{xa_i = x(d \times n - k - i)}$. De même pour la lecture dans la mémoire contenant les données de la voie retour, xr_0 est la donnée la plus récente requise et correspond à $x(d \times n - L + k + n_c)$ *(Figure III.16)*. On a donc $\boxed{xr_i = x(d \times n - L + k + n_c - i)}$.

Il a été montré précédemment à la *section III.2.2* sur la partie opérative que les calculs se déroulent légèrement différemment selon que les coefficients sont réels ou complexes. Les deux cas sont donc étudiés séparément.

Coefficients réels ($n_c = 2$)

La partie opérative calcule la contribution des coefficients w_k et w_{k+1} pour le calcul de plusieurs résultats partiels successifs (en fonction du parallélisme de sous-mots utilisé).

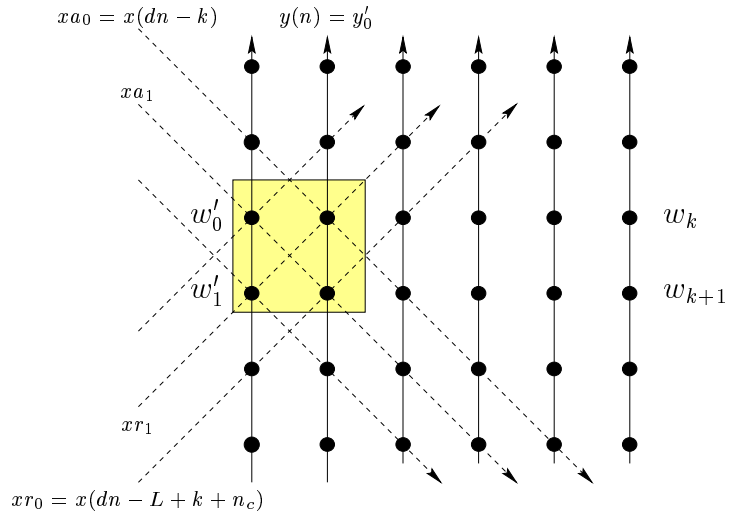


FIGURE III.16 : *Graphe de dépendance pour un filtre réel sur 16 bits ($n_c = 2$, $p_y = 2$) Un point correspond à la contribution d'un coefficient au calcul d'un filtre. Le rectangle grisé contient l'ensemble des opérations calculées au même instant (une itération). Les flèches verticales représentent la progression du calcul des résultats tandis que les flèches en diagonale représentent la propagation des données.*

$$\begin{aligned} y(n-i) = y(n-i) &+ w_k \times (x(d \times (n-i) - k) + x(d \times (n-i) - L + k + 1)) \\ &+ w_{k+1} \times (x(d \times (n-i) - k - 1) + x(d \times (n-i) - L + k + 2)) \end{aligned} \tag{III.13}$$

ce qui donne donc en remplaçant $y(n-i)$ par y'_i ou y_i, w_k par w'_0 et w_{k+1} par w'_1 :

$$y'_i = y_i + w'_0 \times (xa_{d\times i} + xr_{d\times i+1}) + w'_1 \times (xa_{d\times i+1} + xr_{d\times i+1}) \tag{III.14}$$

Nous en déduisons la relation entre les éléments de Xa en sortie de la mémoire et ceux de $X'a_0$ et $X'a_1$ en entrée de la partie opérative. C'est cette opération de redistribution des données qui est réalisée dans le bloc de réorganisation des données.

$$x'a_{0i} = xa_{d\times i} \tag{III.15a}$$

$$x'a_{1i} = xa_{d\times i+1} \tag{III.15b}$$

Coefficients complexes ($n_c = 1$)

Pour des coefficients complexes, la contribution d'un coefficient demande plus d'opérations, si bien que les calculs ne s'effectuent que pour un seul coefficient. Cependant, il

est nécessaire de dissocier les calculs sur les parties réelles et imaginaires des opérations puisqu'elles n'ont pas le même comportement.

$$\begin{aligned} y(n-i)^R = y(n-i)^R &+ w_k^R \times x(d \times (n-i) - k)^R \\ &- w_k^I \times x(d \times (n-i) - k)^I \qquad \text{(III.16a)} \\ y(n-i)^I = y(n-i)^I &+ w_k^R \times x(d \times (n-i) - k)^I \\ &+ w_k^I \times x(d \times (n-i) - k)^R \qquad \text{(III.16b)} \end{aligned}$$

$$y_i^{'R} = y_i^R + w_0' \times xa_{d\times i}^R - w_1' \times xa_{d\times i}^I \qquad \text{(III.17a)}$$

$$y_i^{'I} = y_i^I + w_0' \times xa_{d\times i}^I + w_1' \times xa_{d\times i}^R \qquad \text{(III.17b)}$$

on en déduit

$$x'a_{0i} = xa_{d\times i} \qquad \text{(III.18a)}$$

$$x'a_{1i} = xa_{d\times i}^{\#} \qquad \text{(III.18b)}$$

où $x^{\#} = (x^I, x^R)$ lorsque $x = (x^R, x^I)$.

Maintenant que les parties du CCP, réalisant les manipulations sur les données pour calculer les résultats des filtres, ont été décrites, il faut s'intéresser aux mécanismes qui vont gérer toutes ces manipulations de la façon la plus efficace possible.

III.2.5 Le contrôleur

Le contrôleur génére l'ensemble des signaux qui contrôlent l'ensemble des autres parties du circuit en fonction du filtre à réaliser et des calculs internes.

Chaque fois qu'un mot de données est reçu, tout un ensemble d'opérations doit être réalisé d'une façon bien précise pour obtenir un résultat correspondant au filtre à exécuter. Pour expliquer le séquencement de ces opérations, le contrôleur va être défini sous la forme d'une machine à états finis.

Une idée du séquencement des opérations a déjà été abordée au début de la *section III.2.2*. Il s'agit de mettre en évidence la partie des calculs qui dépend du dernier mot de données arrivé afin de réduire au minimum la latence des calculs due à l'implémentation. Ainsi, il y a deux étapes principales dans le calcul du résultat d'un filtre. La première étape consiste à calculer toutes les sommes de produits possibles avant l'arrivée du mot de données nécessaire au calcul du résultat. La deuxième étape correspond au calcul du résultat final dès l'arrivée de ce mot de données. En ajoutant

une phase d'initialisation et une phase d'inactivité lorsqu'aucun calcul n'est réalisé, on obtient une machine à états composée de 4 états.

Comme plusieurs filtres sont susceptibles d'être réalisés par le même coprocesseur, les différents calculs correspondant aux différents filtres risquent d'être imbriqués, rendant assez complexe le fonctionnement de la machine à états. Dans un souci de clarté, nous allons donc commencer par étudier le comportement de la machine à états dans un cas simple, celui de la réalisation d'un seul filtre simple. Ensuite, nous étudierons l'influence de la décimation, du parrallélisme de sous-mots et du multiplexage de filtres sur le comportement de l'automate.

III.2.5.a Machine à états pour la réalisation d'un filtre [85]

Cette section présente le procédé de calcul anticipé utilisé pour obtenir rapidement les résultats des filtres dans un cas simple.

État 0 : Initialisation

Il s'agit de charger dans le coprocesseur les caractéristiques du filtre qui seront nécessaires pour les calculs (nombre de coefficients, type de filtre, type de données) ainsi que les valeurs initiales des adresses de lecture et d'écriture pour les mémoires. En effet, lorsque plusieurs filtres sont multiplexés sur le coprocesseur de filtrage, il utilisent tous la mémoire pour conserver les données et les coefficients. Pour qu'il n'y ait pas de conflits d'accès, chaque filtre va disposer de son propre espace mémoire dont l'emplacement dépendra de la taille du filtre et des valeurs initiales des adresses d'écriture.

Ensuite, le chargement des coefficients du filtre peut s'effectuer. La phase d'initialisation est alors terminée et le coprocesseur peut passer dans la phase suivante. Soit un mot de données est reçu dès la fin de la phase d'initialisation et le calcul d'un résultat doit être réalisé dans l'état appelé *ultime itération*. Soit aucune donnée n'est encore arrivée et, dans ce cas, le coprocesseur se retrouve dans un état *inactif*.

État 1 : Calcul anticipé

Il s'agit de réaliser l'ensemble des opérations qui peuvent être faites avec les données présentes dans le CCP. Cela représente en général la majorité des calculs à faire. Le coprocesseur reste dans cet état tant que des calculs peuvent encore être faits. Une fois qu'il a fini ces "pré-calculs", soit un mot de données est reçu et le calcul d'un résultat doit alors être réalisé dans l'état appelé *ultime itération*, soit aucune donnée n'est encore arrivée et, dans ce cas, le coprocesseur se retrouve dans un état *inactif*.

État 2 : Ultime itération

Cet état est atteint chaque fois qu'un nouveau mot de données est reçu. Il s'agit de réaliser les derniers calculs qui ne pouvaient pas être faits avant l'arrivée de

ce nouveau mot de données. Comme énoncé au début de la *section III.2.2*, le coprocesseur a été conçu de manière à ce que l'ensemble des calculs concernés dans cet état se fasse sans perte de temps. Ainsi, cet état ne doit durer qu'un seul cycle. En général, l'état suivant est l'état de *calcul anticipé* qui commencera les "pré-calculs" correspondant au résultat suivant. Néanmoins, si le filtre est suffisamment petit, il est possible qu'aucun "pré-calcul" ne soit requis. Dans ce cas, soit un nouveau mot de données est reçu et le coprocesseur reste alors dans cet état pour calculer le prochain résultat, soit aucune donnée n'est encore arrivée et, dans ce cas, le coprocesseur se retrouve dans un état *inactif*.

État 3 : Inactif

Cet état est atteint quand aucun calcul ne peut plus être fait, la réception d'un nouveau mot de données étant nécessaire pour continuer les calculs en passant dans l'état d'*ultime itération*. Ainsi, tant qu'aucune donnée n'est reçue, le coprocesseur reste dans cet état.

La *figure III.17* représente un exemple de machine à états dans le cas d'un filtre complexe avec une précision des données sur 16 bits.

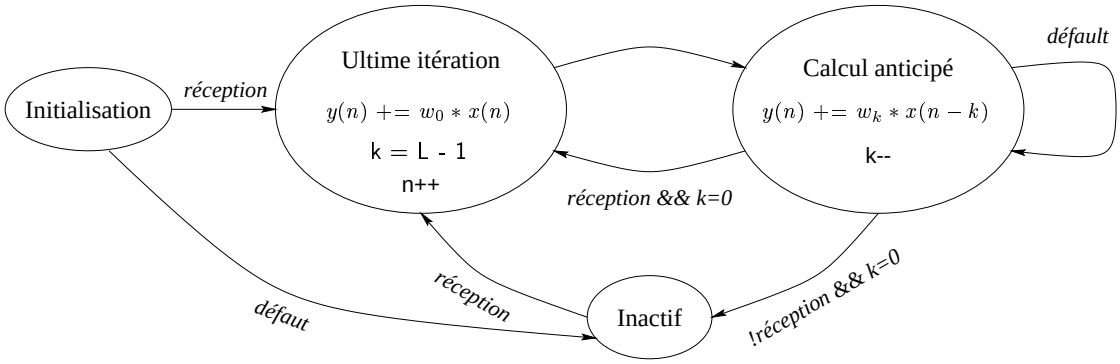


FIGURE III.17 : *Machine à états pour un filtre*

III.2.5.b Influence de la décimation

Pour un filtre standard, la réception d'une donnée déclenche automatiquement le calcul et l'émission d'un résultat. Mais pour un filtre avec décimation, la réception de certaines données ne provoque pas le calcul d'un résultat. Par conséquent, la réception de telles données ne doit pas faire passer l'automate dans l'état d'*ultime itération* comme c'était le cas pour les filtres standards. En fait, les calculs des filtres se font alors en 3 étapes :

- le calcul anticipé qui utilise les données présentes pour le calcul du résultat précédent,

- le calcul semi-anticipé qui utilise les données arrivées après le calcul du résultat précédent,
- la réception des données déclenchant l'ultime itération et finalement le calcul de la dernière itération *(Figure III.18)*.

Prenons par exemple le graphe de dépendance d'un filtre complexe de 4 coefficients sur 16 bits avec décimation par 2 *(Figure III.18)*. On suppose qu'un résultat est calculé tous les t cycles, c'est-à-dire qu'une donnée arrive tous les $\frac{t}{2}$ cycles. Il apparaît clairement que les deux dernières itérations du calcul de $y(n)$ dépendent de données qui n'étaient pas présentes lors du calcul de $y(n-1)$.

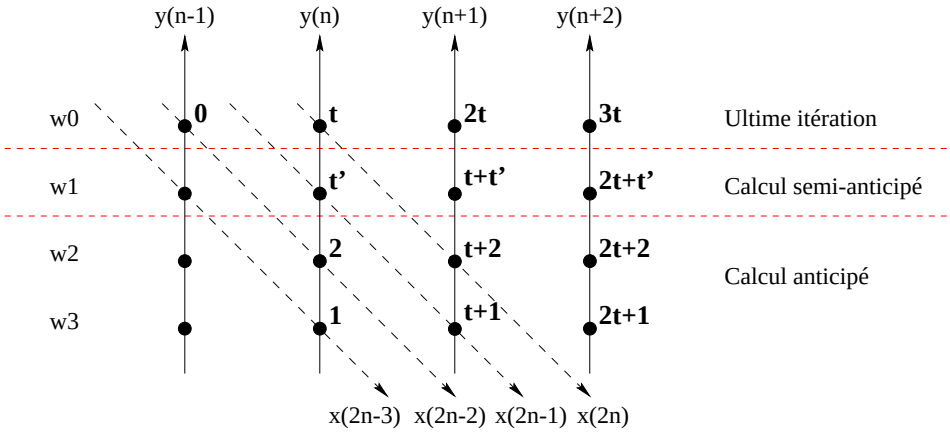


FIGURE III.18 : *Graphe de dépendance pour un filtre complexe avec décimation par 2.*
t représente le nombre de cycles entre le calcul de 2 résultats, ce qui signifie qu'une donnée arrive tous les $\frac{t}{2}$ cycles.
L'exécution du calcul semi-anticipé à l'instant t' dépend de l'arrivée de $x(2n-1)$ à l'instant $\frac{t}{2}$ ainsi que de l'avancement du calcul anticipé au moment de cette arrivée.

Le calcul semi-anticipé est toutefois un calcul anticipé puisqu'il est effectué avant la réception de la donnée provoquant la dernière itération et donc l'obtention du résultat. Mais pour l'intégrer dans l'état effectuant le calcul anticipé, il est nécessaire de faire quelques changements dans les conditions de transition entre les états.

Ainsi, il est possible de passer de l'état *Inactif* à l'état *Calcul anticipé* si la donnée utilisée pour le calcul semi-anticipé arrive alors que l'automate se trouvait dans l'état *Inactif* en train d'attendre cette donnée.

La *figure III.19* représente un exemple de machine à états dans le cas d'un filtre complexe avec décimation par 2 avec une précision des données sur 16 bits. Dans ce schéma, le calcul semi-anticipé est intégré avec les calculs anticipés grâce à la modification de quelques transitions entre les états (les flèches en pointillé) par rapport à la machine à états présentée à la *figure III.17*.

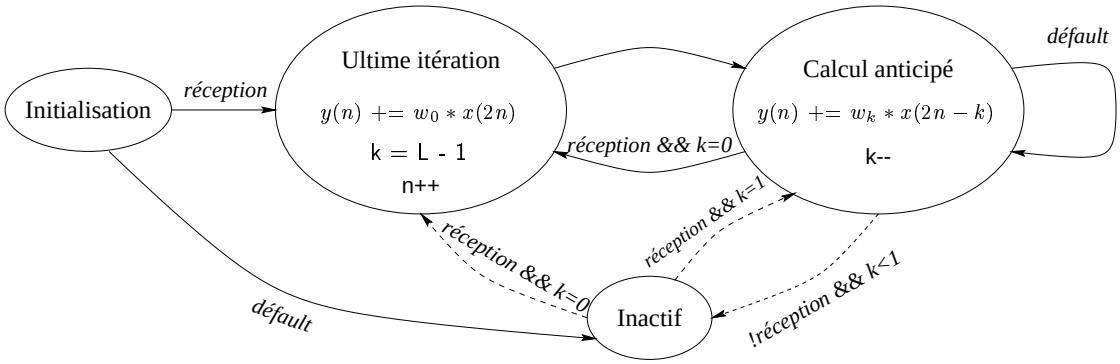


FIGURE III.19 : *Machine à états pour un filtre avec décimation par 2*

III.2.5.c Influence du parallélisme de sous-mots

La partie opérative du CCP a été conçue pour calculer la contribution d'un coefficient quelle qu'elle soit en une seule itération pour réduire au minimum la latence des filtres *(Section III.2.2)*.

Elle n'est toutefois pas suffisante lorsque le parallélisme de sous-mots a pour conséquence l'arrivée simultanée de plusieurs données qui ne peuvent pas être utilisées toutes en même temps. Il en résulte alors une augmentation de la latence de calcul du filtre.

Ce problème apparaît pour les filtres complexes 8 bits comme le montre la *figure III.20*. Dans ce cas, l'arrivée du mot de données $(x(n-1), x(n))$ déclenche la fin des calculs pour obtenir $(y(n-1), y(n))$. Il faut une itération pour calculer la contribution de w'_1 qui attendait l'arrivée de $x(n-1)$ et une itération pour calculer la contribution de w'_1 qui attendait l'arrivée de $x(n)$.

Le même problème se pose pour les filtres réels sur 8 bits où 4 données sont fournies simultanément alors que la contribution de 2 coefficients est calculée en une itération.

Pour prendre en compte l'exécution de ces filtres, il est donc nécessaire de modifier les conditions de transitions entre les états et de transformer l'état *ultime itération* pour qu'il puisse durer 2 cycles au lieu d'un seul.

La *figure III.21* représente un exemple de machine à états dans le cas d'un filtre complexe avec une précision des données sur 8 bits. Dans ce schéma, les deux dernières itérations de calcul sont réalisées lorsque la machine est dans l'état *Ultimes itérations*, grâce à la modification de quelques transitions entre les états par rapport à la machine représentée *figure III.17* (les flèches en pointillé).

L'influence du parallélisme de sous-mots et de la décimation s'additionne si bien que, pour certains filtres avec décimation, le calcul semi-anticipé dure également pendant 2 itérations.

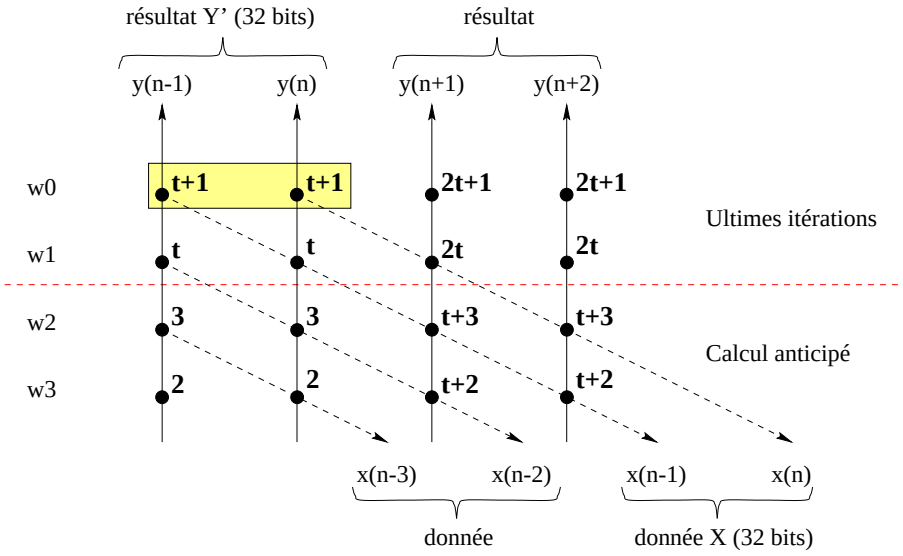


FIGURE III.20 : *Graphe de dépendance pour un filtre complexe sur 8 bits. t représente le nombre de cycles entre le calcul de 2 résultats.*

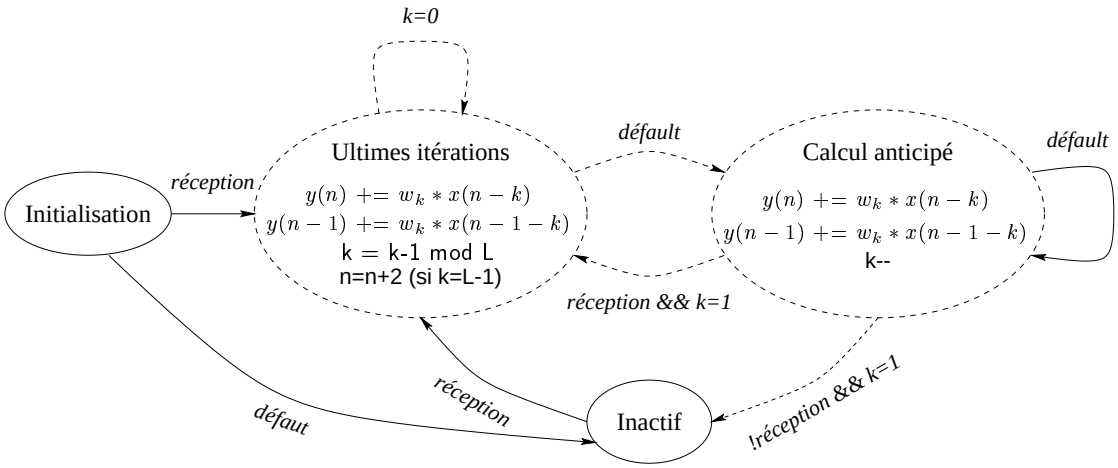


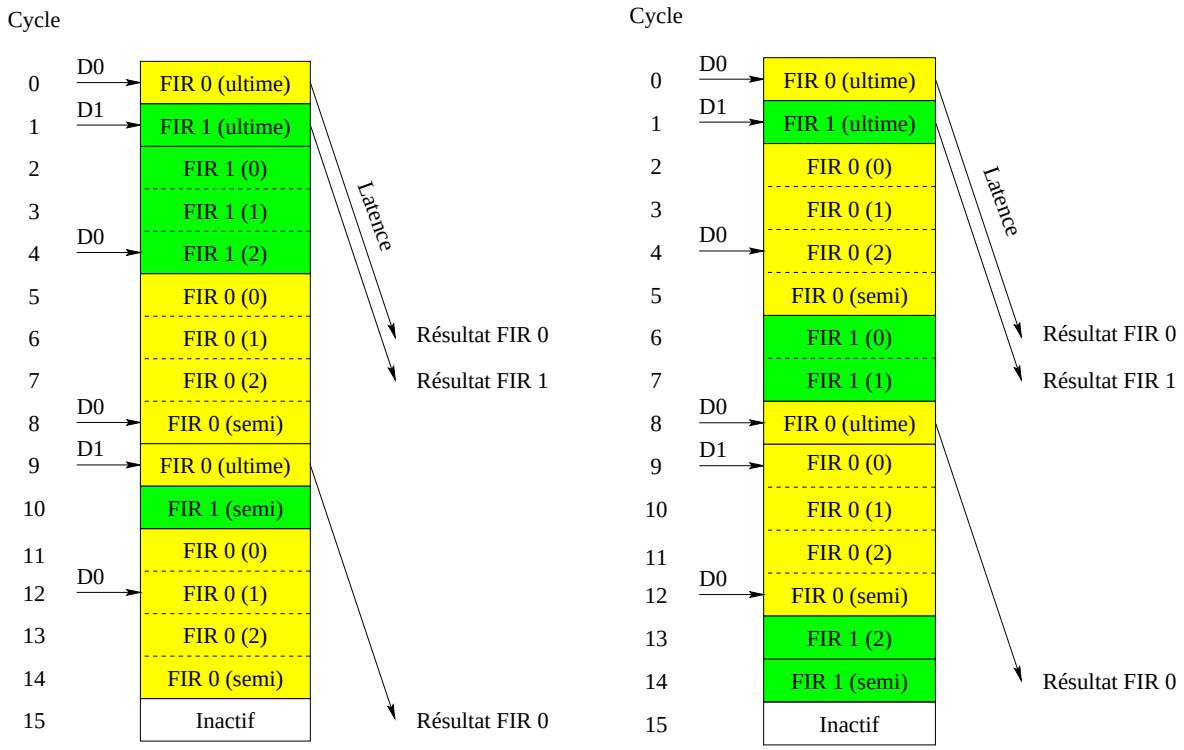
FIGURE III.21 : *Machine à états pour un filtre avec un parallélisme de sous-mots*

III.2.5.d Influence du multiplexage de filtres

Nous avons défini l'automate pour un seul filtre. Pour contrôler plusieurs filtres multiplexés, on peut considérer que l'on a un automate par filtre et qu'un seul de ces automates est actif. Ce qu'il faut, c'est décider quel est l'automate actif.

Afin de conserver la faible latence dans les calculs, l'arrivée d'une donnée censée provoquer l'ultime itération d'un filtre le rend automatiquement actif. Ce qu'il reste à organiser, ce sont les priorités pour les calculs anticipés des différents filtres pour que ceux-ci soient terminés lorsque les données arrivent.

Une solution simple est de donner une priorité fixe à chaque filtre en fonction du débit de ses résultats. En effet, plus un filtre est effectué fréquemment, plus il a besoin que ses pré-calculs le soient aussi, les calculs anticipés des filtres les plus lents s'exécutant pendant les cycles libres restants.



(a) *Priorité aux filtres lents* (b) *Priorité aux filtres fréquents*

FIGURE III.22 : *Exemple de multiplexage de filtre*

La *figure III.22* montre la différence entre des filtres fréquents prioritaires et des filtres lents prioritaires, en prenant comme exemple le filtre anti-repliement qui utilise une succession de filtre avec une décimation par deux entre chaque filtre. FIR 0 et FIR 1 représentent donc deux filtres avec décimation nécessitant chacun 5 cycles de calcul entre deux résultats. Les données $D0$ (resp. $D1$) pour FIR 0 (resp. FIR 1) arrivent tous les 4 cycles (resp. 8 cycles).

Lorsque la priorité est donnée aux filtres lents *(Figure III.22(a))*, l'exécution des pré-calculs des filtres fréquents est retardée si bien que ces pré-calculs ne sont pas finis lorsque qu'arrive une nouvelle donnée (au cycle 8). L'ultime itération de FIR 0 est alors elle aussi retardée ainsi que la sortie du résultat.

Lorsque la priorité est donnée aux filtres fréquents *(Figure III.22(b))*, l'exécution de leurs pré-calculs se fait rapidement si bien qu'ils sont terminés lorsqu'arrive une nouvelle donnée. L'ultime itération de FIR 0 peut alors être effectuée immédiatement et la sortie du résultat ce fait plus rapidement.

L'architecture du CCP ayant été présentée, il faut maintenant montrer qu'il est en

effet capable de calculer l'ensemble des filtres pour lesquels il a été conçu. Pour cela, une plateforme de validation a été écrite.

III.3 Plateforme de validation [86]

Cette plateforme de simulation a pour objet de mettre en œuvre l'architecture définie, en utilisant une description précise au cycle près du comportement du coprocesseur de filtrage, et de tester son comportement sur un ensemble de filtres le plus exhaustif possible.

Pour réaliser une description comportementale précise au cycle près du CCP, il est nécessaire d'avoir une idée de l'emplacement des niveaux de pipeline dans l'architecture. Il a été expliqué précédemment le séquencement des calculs qui utilise le procédé de calcul anticipé *(Section III.2.5)* si bien que le calcul d'un résultat est effectué immédiatement après la réception d'une donnée. En fait, la donnée va progresser au travers de l'architecture et après quelques cycles le résultat sera enfin disponible.

La *figure III.23* présente le cheminement de la donnée avec les conséquences temporelles causées par les barrières de pipeline prévues dans l'architecture, le flux de contrôle généré par la machine à état suivant le flux de données.

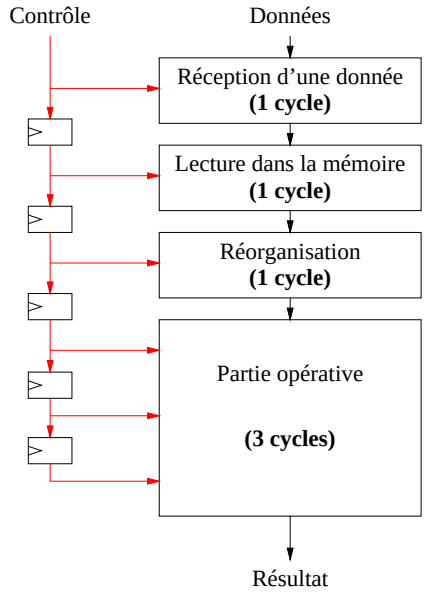


FIGURE III.23 : *Décomposition de la latence du CCP*

En premier lieu, la réception de la donnée consomme déjà un cycle de calcul. Ensuite, la lecture dans la mémoire des données supplémentaires requises par les calculs, puis la réorganisation de ces données s'effectuent également en un cycle chacune. Finalement, trois cycles sont prévus pour réaliser la partie opérative dont le chemin critique passe

par un additionneur, un multiplieur puis 2 additionneurs. Ainsi, la latence estimée du CCP est de 6 cycles.

Afin de vérifier la description comportementale du CCP, la plateforme de simulation calcule les filtres de deux manières différentes et compare les résultats obtenus.

Ainsi, on a d'un côté le calcul du filtre de référence et de l'autre côté le calcul du filtre par le CCP. Le filtre de référence calcule le résultat d'un filtre naturellement, en faisant une somme de produits, alors que le CCP simule l'ensemble des éléments définis dans l'architecture : la machine à état, la partie opérative, les mémoires et le calcul des adresses, la réorganisation des données, le parallélisme de sous-mots, le multiplexage des filtres et le calcul d'un filtre sur plusieurs CCP en cascade.

Le simulateur *(Figure III.24)* est donc constitué de :

- un créateur de filtres qui va définir des filtres (type du filtre, valeur des coefficients, débit des échantillons) et initialiser le filtre de référence et les CCP pour qu'ils exécutent le filtre ainsi créé. La création d'un filtre pour les CCP comprend aussi la répartition des coefficients sur les différents CCP si un calcul sur plusieurs CCP est prévu. Une fois que cette étape est terminée, les autres étapes s'exécutent à chaque cycle.

- Un générateur de données qui va, à certains cycles, générer des données aléatoires devant être utilisées par l'un des différents filtres créés.

- Un tableau de filtres de référence qui, lorsque des données ont été générées, calcule simplement le résultat du filtre correspondant.

- Un simulateur de CCP qui effectue, à chaque cycle, l'ensemble des calculs que réaliseraient les CCP représentés.

- Un comparateur qui compare les résultats venant des filtres de référence et du simulateur de CCP pour vérifier le bon comportement du simulateur de CCP. Cela permet de détecter les filtres qui ne sont pas correctement réalisés par les CCP.

Pratiquement tous les types de filtres (symétrique, antisymétrique, adaptatif, multiplication complexe) et toutes les fonctionalités du CCP (multiplexage, parallélisme de sous-mots, cascadabilité) ont été testées avec succès par ce simulateur. Seule la FFT n'a pas encore été mise en œuvre dans ce simulateur.

A côté de cette description comportementale, une estimation de la complexité de l'architecture a été réalisée grâce à une description VHDL synthétisable.

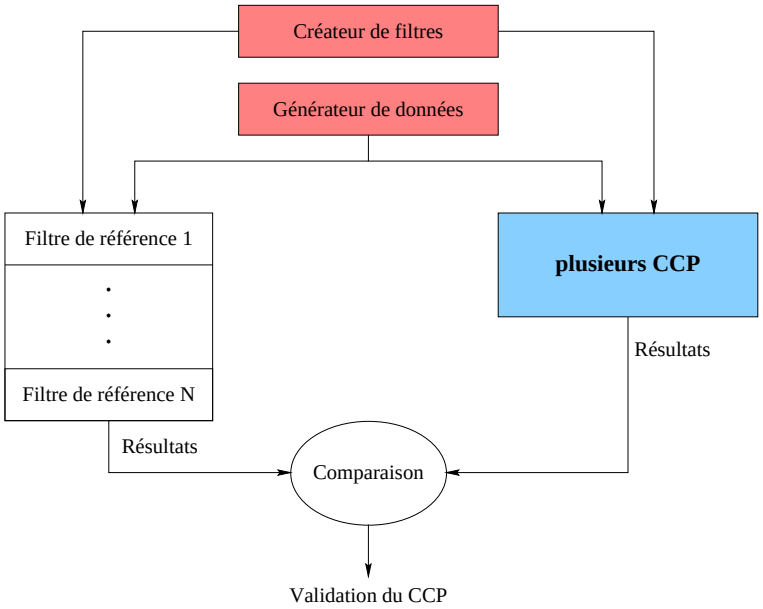


FIGURE III.24 : *Plateforme de simulation du CCP*

III.4 Estimation de la complexité du CCP [87]

L'architecture du CCP est assez générique. En effet elle a été conçue pour exécuter différents types de filtres sans se donner de contraintes sur la taille des filtres ou le nombre de filtres qu'il est possible de multiplexer. Mais, lorsqu'il s'agira de mettre en œuvre la réalisation matérielle du CCP, il faudra fixer la taille des mémoires.

Pour obtenir une estimation de la complexité globale du CCP, on va estimer la complexité des principaux blocs, à savoir la partie opérative qui exécute les multiplications et les accumulations, le bloc de réorganisation des données qui se trouve entre les mémoires et la partie opérative pour ordonner correctement les données et enfin la machine à états qui gère le séquencement des opérations et calcule les adresses utilisées par les mémoires.

Le *tableau III.2* présente le résultat de la synthèse d'un modéle VHDL presque complet du CCP, celui-ci ne comprenant toutefois pas la logique pour effectuer la mise à jour des coefficients d'un filtre adaptatif, les multiplications complexes 8 bits ou la FFT. Le modèle VHDL a été développé sous COMPASS avec une technologie 0,35μm. Cette synthèse est effectuée avec différentes tailles de mémoires en fonction du nombre de coefficients (8, 16, 24 ou 32) qui peuvent être stockés dans ces mémoires. Une organisation de la mémoire en trois blocs identiques, un pour les coefficients, un pour les données de la voie aller et un pour les données de la voie retour a été choisie. La mémoire est réalisée avec des registres à cause du nombre d'accès simultanés à ces mémoires (jusqu'à 2 lectures et une écriture).

	Nombre de portes	Chemin critique
Partie opérative	13.300	40 ns
Réorganisation	2.500	8 ns
Contrôleur	3.300	6 ns
Mémoire (8)	10.200	7 ns
Mémoire (16)	17.700	9 ns
Mémoire (24)	25.600	10 ns
Mémoire (32)	33.600	10 ns

TABLEAU III.2: *Complexité des blocs constituant le CCP*

Cette synthèse permet de vérifier que les deux parties principales au niveau de la complexité matérielle sont la partie opérative et les mémoires qui stockent les données et les coefficients. Le chemin critique de la partie opérative est important (40 ns). Ceci s'explique puisqu'elle ne comporte aucune couche de pipeline alors qu'elle contient des multiplieurs et des additionneurs qui se suivent. Un effort sérieux d'implantation est donc nécessaire pour réussir à obtenir un bloc plus rapide. Toutefois, il faut rappeler qu'il est prévu, au niveau de la description comportementale du CCP, de mettre 2 niveaux de pipeline à l'interieur de la partie opérative. Ceci permettra de réduire considérablement le chemin critique de ce bloc.

Finalement, il est possible de représenter la courbe reliant la complexité du circuit en fonction du nombre de coefficients qui pourra y être stocké *(Figure III.25)* pour vérifier que la complexité est une fonction linéaire croissante de la taille des mémoires.

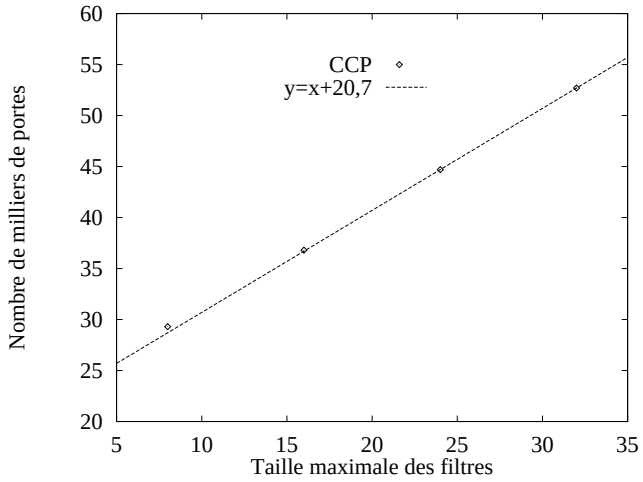


FIGURE III.25 : *Complexité du CCP en fonction de la longueur des filtres supportés*

III.5 Conclusion

Les besoins en filtrage requis par la démodulation de la télévision numérique demandaient la conception d'un composant spécifique pouvant réaliser d'une manière efficace plusieurs filtres de types différents tout en respectant des contraintes de latence importantes.

C'est un composant respectant ces contraintes, grâce à un procédé de calcul anticipé et un multiplexage des calculs combiné à un changement de contexte non pénalisant, qui a été présenté ici. Toutes les fonctionalités de ce composant requises pour les démodulations câble et satellite ont été testées. Aujourd'hui, seule la FFT requise pour la démodulation terrestre n'est pas encore supportée par le simulateur. Une fois que cette fonctionalité aura été ajoutée, il restera à réaliser un générateur de CCP qui produira un modèle VHDL du CCP désiré en fonction des paramètres choisis (taille des mémoires, fonctionalités effectivement requises). Ainsi, il sera possible de disposer de plusieurs versions du CCP, optimisées selon les besoins réels.

Maintenant, il reste à placer ce composant dans une architecture hétérogène afin de pouvoir mettre en œuvre effectivement la démodulation de la télévision numérique.

D'autres fonctionnalités peuvent être ajoutées au CCP. Par exemple, en complétant la structure d'interconnexion pour permettre le rebouclage de la sortie vers la partie opérative et/ou la mémoire de données, il deviendrait possible d'effectuer des filtres récursifs. Cette extension fait partie de nos perspectives.

Enfin, la réorganisation des données entre la mémoire et la partie opérative est un concept d'architecture nouveau. Ce type d'architecture pourrait, à notre avis, être efficacement utilisé pour les prochaines générations de processeurs de traitement du signal (DSP) : leurs performances pour la réalisation de différents types de filtres serait ainsi accrues.

Chapitre IV

Présentation de l'architecture P-DCP

Notre recherche d'une architecture flexible pour la démodulation de la télévision numérique nous a amené à identifier le problème du filtrage et à y apporter une solution par le co-processeur CCP décrit au chapitre précédent. Néanmoins, notre architecture ne se résume pas aux filtrages et de nombreux problèmes restent à résoudre :

- Comment réaliser les fonctions qui ne sont pas prises en compte par le CCP ?
- Quel type de séquencement utiliser : dynamique ou statique ?
- Quel réseau d'interconnexion choisir ?

L'architecture résultante de ces choix sera appelée P-DCP pour "Parallel-Digital Communication Processor". Il sera alors possible de réaliser les premières estimations de complexité.

Par conséquent, la conception de l'architecture va commencer par la résolution des différents problèmes qui subsistent avant de détailler le mécanisme de communication utilisé par l'architecure. On terminera par l'estimation de la complexité de l'architecture.

IV.1 Conception de l'architecture

Avant de rentrer dans les détails de la conception de l'architecture, nous allons présenter l'architecture cible que nous utilisons. Il s'agit d'une architecture hétérogène à base de coprocesseurs.

IV.1.1 Description du P-DCP

L'architecture générique du P-DCP est composée de plusieurs blocs de calcul chargés de réaliser les algorithmes demandant une puissance de calcul trop importante pour pouvoir être implantés sur un processeur général d'une façon efficace. Un bon exemple d'un tel bloc de calcul est le CCP qui a été défini précédemment. À côté de ces blocs de calcul, un ou plusieurs Processeurs Elémentaires (PE) ont pour tâche d'effectuer les algorithmes à faible débit qui n'ont pas nécessité la conception d'un bloc de calcul. Typiquement, ce processeur élémentaire pourrait être un cœur de DSP. Finalement, de la mémoire est prévue pour répondre aux contraintes de la démodulation multiporteuse utilisée pour la diffusion de la télévision numérique par ondes hertziennes. Tous ces éléments de l'architecture sont reliés entre eux par un réseau d'interconnexion.

La *figure IV.1* présente l'architecture générique du P-DCP. Cette architecture va petit à petit se préciser (définition des composants, du réseau d'interconnexion) pour être à même de supporter les différentes applications visées.

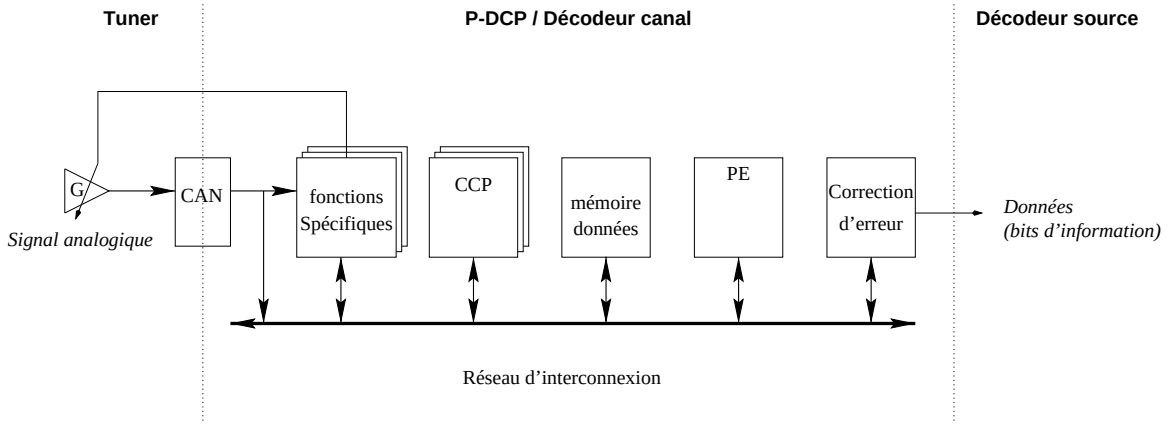


FIGURE IV.1 : *Architecture générique P-DCP*

Le dimensionnement de l'architecture (nombre de PE, de blocs de calcul, structure du réseau d'interconnexion) ne pourra être effectué que lorsque l'on s'occupera de l'ordonnancement des calculs pour résoudre les conflits d'accès au réseau d'interconnexion ou aux blocs de calcul. Il dépend également de la vitesse de fonctionnement de ces unités, c'est-à-dire du nombre de cycles d'horloge entre l'arrivée de deux échantillons de l'application. Une fréquence de travail de 130 MHz est prévue, ce qui correspond à deux fois la fréquence d'échantillonnage du satellite. En effet, l'intérêt d'une architecture programmable est dans l'utilisation d'une fréquence de travail importante par rapport à la fréquence d'échantillonnage des données, afin de pouvoir utiliser le même matériel plusieurs fois grâce à la programmation. De plus, étant donné les fréquences obtenues aujourd'hui avec les technologies actuelles dans les processeurs (450 MHz pour le pentium II d'Intel, 200 MHz pour le denier DSP de Texas Instrument, 110 MHz pour le TriMedia de Philips), un circuit utilisant une fréquence de 130 MHz est tout à fait concevable pour un marché grand public.

Il faut choisir les fonctions qui seront mises en œuvre dans un bloc de calcul et celles qui ne le seront pas. Ainsi, il sera possible de définir les interconnexions nécessaires entre ces blocs de calcul.

IV.1.2 Choix des blocs de calcul

L'utilisation de blocs de calculs permet de concevoir des architectures plus efficaces que des architectures générales. En effet, la mise en œuvre d'une fonction dans un bloc de calcul spécifique est, en général, 6 à 7 fois plus petite que dans un processeur programmable [88]. En contrepartie, le gain en performance se paye par une diminution de la flexibilité. Il faut donc trouver un compromis entre l'ajout d'un bloc de calcul pour gagner en surface et l'utilisation d'un processeur programmable pour conserver de la flexibilité.

Ainsi, les blocs de calcul sont utilisés pour les fonctions demandant un débit et une puissance de calcul importante puisque c'est pour ces fonctions que le gain en surface sera le plus important.

Comme la démodulation câble et surtout la démodulation satellite utilisent des fréquences symboles élevées (jusqu'à 32 MHz), le coût d'implantation de la moindre fonction d'une de ces applications sur un processeur programmable serait trop important pour être envisagée ici.

Par conséquent, tous les algorithmes de ces 2 démodulations seront mis en œuvre par des blocs de calcul. Attendu que les fonctions de filtrage et les multiplications complexes qui constituent l'ensemble de la chaîne directe de démodulation sont réalisées par le CCP, il faut prévoir un bloc de calcul pour chacune des boucles de contre-réaction :

- un pour le contrôle automatique de gain,
- un pour la récupération d'horloge,
- un pour la récupération de porteuse.

Il faut également un bloc de calcul pour l'égaliseur récursif avec décision dans la boucle utilisé dans la démodulation câble puisqu'il n'est pas constitué uniquement de filtrage, mais aussi de 2 additionneurs et d'une fonction de décision qui ne sont pas réalisés dans le CCP *(Figure I.7 page 12)*.

En ce qui concerne la démodulation terrestre, la puissance de calcul requise par les fonctions qui ne sont pas entièrement réalisées par le CCP ou le bloc de calcul pour le contrôle automatique de gain est assez faible (110 Mops) relativement aux 10 Gops que doit développer l'architecture P-DCP. De plus ces fonctions utilisent intensivement les multiplications complexes pour effectuer les calculs d'autocorrélation *(Section I.3.2)* si

bien qu'une partie de ces fonctions peut être prise en charge par les CCP. Au bout du compte il ne reste que 60 Mops à réaliser dans un processeur programmable. Ces calculs sont d'ailleurs déjà réalisées à l'aide d'un cœur de DSP dans le démodulateur conçu lors du projet européen DVBird [89].

Maintenant que les différentes unités de calcul de l'architecture ont été répertoriées, il est nécessaire de s'intéresser à la manière dont elles seront connectées entre elles.

IV.1.3 Le réseau d'interconnexion [90]

Les applications de démodulation câble et satellite contiennent des boucles de contre-réaction rapides pour synchroniser le récepteur avec l'émetteur. La latence des communications, au même titre que la latence des composants, est un élément critique de l'architecture. Il faut que les communications se passent le plus rapidement possible si bien qu'il est impensable d'envisager des communications par paquets ou des protocoles de routage logiciel.

Pour ces applications on remarque que la plupart des communications s'effectuent de proche en proche entre des fonctions de filtrage ou des multiplications complexes avec une structure linéaire. Seules quelques boucles de contre-réaction demandent des communications globales.

Le réseau d'interconnexion est donc constitué essentiellement de connexions locales unidirectionnelles entre les composants avec quelques connexions spécifiques supplémentaires pour les boucles de contre-réaction comme décrit dans la *figure IV.2*.

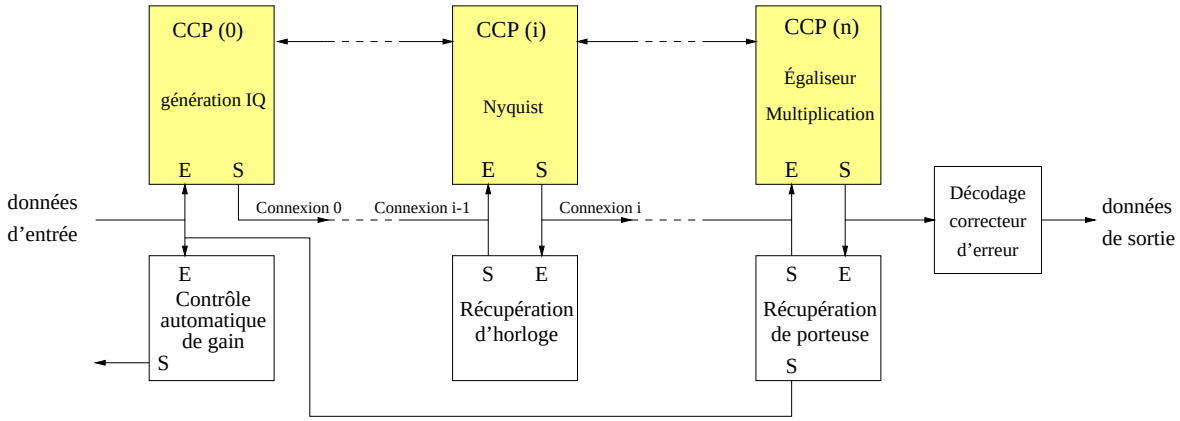


FIGURE IV.2 : *Réseau d'interconnexion style fonctionnel*

Mais, comme les fonctions ne sont pas réalisées de manière dédiée, il n'est pas possible de garantir que toutes les communications locales du point de vue de l'application le restent effectivement lorsque l'on considère l'architecture. Et ce, même si l'assignation des composants est effectuée de manière à conserver au maximum la structure

première de l'application. Par exemple, le filtre de Nyquist qui interagit avec la fonction de récupération d'horloge pourrait ne pas être réalisé sur le CCP(i) qui interagit directement avec le bloc de calcul effectuant la fonction de récupération d'horloge. Dans ce cas, il faut prévoir des connexions supplémentaires entre les blocs de calculs.

Par conséquent, les différentes connexions sont reliées les unes aux autres par des registres ce qui permet à n'importe quelle contre-réaction d'atteindre les CCP situés plus loin dans la chaîne des CCP. De plus, pour que la boucle de contre-réaction puisse également obtenir les informations en provenance d'autres CCP, une autre succession de connexions est réalisée, les données circulant dans le sens inverse *(Figure IV.3)*.

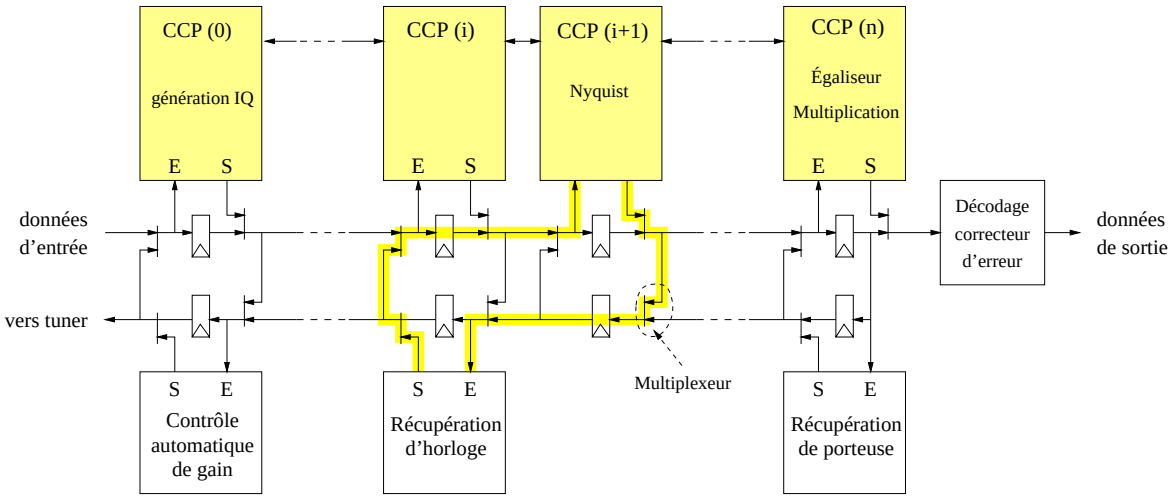


FIGURE IV.3 : *Réseau d'interconnexion complet*

La *figure IV.3* présente un schéma du réseau d'interconnexion qui relie les différents composants entre eux. Il prend comme exemple un filtre de Nyquist réalisé sur un autre CCP que le CCP prévu initialement pour l'exécuter, afin de montrer l'intérêt de l'ensemble des connexions du réseau. On peut remarquer que sur ce schéma, la connexion directe entre le bloc de calcul réalisant la récupération de porteuse et le CCP(0) réalisant la génération IQ a disparu. Cette connexion est maintenant effectuée par la nouvelle suite de connexions qui a été rajoutée.

Après le recensement des blocs de calcul à incorporer dans l'architecture et le réseau d'interconnexion à utiliser, quelques problèmes restent à résoudre, comme celui de la fréquence de travail de l'architecture, vu que différentes fréquences d'échantillonnages sont utilisées selon le medium de diffusion.

IV.1.4 Mise en œuvre de la fréquence de travail [91]

En général, la fréquence de travail d'un circuit de démodulation est choisie avec comme contrainte de représenter un multiple (n) de la fréquence d'échantillonnage.

Mais, dans notre étude, on désire pouvoir exécuter différentes applications de démodulation dont les fréquences d'échantillonnages n'ont aucun rapport entre elles. Donc, il est impossible de choisir une fréquence de travail simple. Pour ne pas devoir utiliser différentes horloges pour les différentes applications, il faut pouvoir générer une horloge effective de travail en fonction d'une horloge donnée et de l'application à réaliser. Cette horloge effective de travail correspondra exactement à une fréquence multiple de la fréquence d'échantillonnage puisqu'elle aura toujours n fronts montants pendant une période d'échantillonnage.

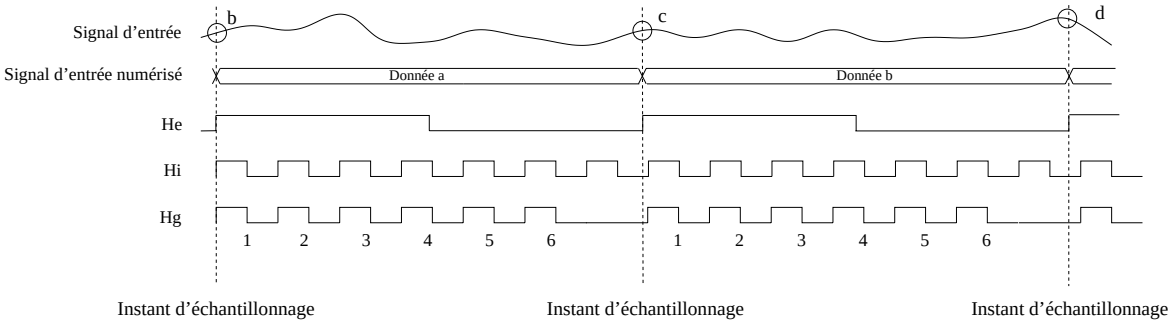


FIGURE IV.4 : *Diagramme de génération de l'horloge*

La figure IV.4 montre ce que l'on cherche à réaliser : inhiber certains fronts montants de l'horloge de travail initiale (Hi) pour obtenir une horloge de travail effective globale (Hg) qui possède un nombre fixe de fronts montants entre 2 instants d'échantillonnage (deux fronts montants de l'horloge d'échantillonnage, He). La mise en œuvre de ce processus est représentée par la figure IV.5.

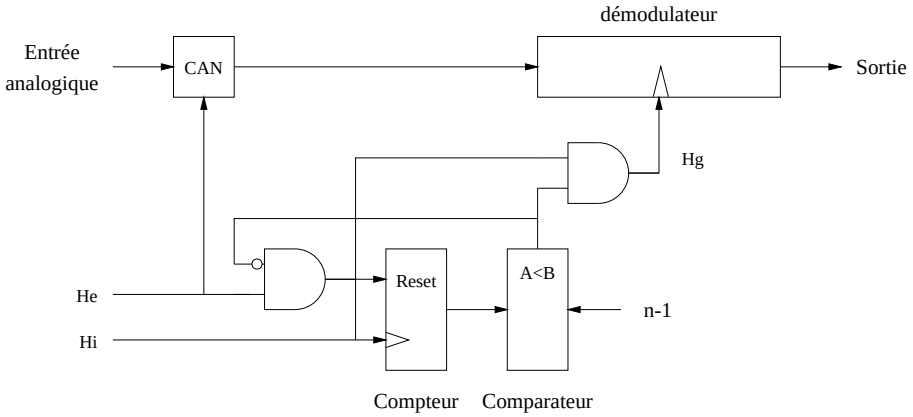


FIGURE IV.5 : *Mise en œuvre de la génération de l'horloge*

Elle utilise un compteur avec un reset synchrone et un comparateur pour décider à partir de quel moment l'horloge de travail initiale doit être inhibée. Le reset du compteur s'effectue lorsque l'horloge d'échantillonnage passe à l'état haut pendant que le comparateur inhibe l'horloge initiale, ce qui veut dire que le système est en attente d'un nouvel échantillon. Ensuite le compteur indique au comparateur le nombre de

cycles de l'horloge de travail initiale qui se sont déroulés depuis le dernier instant d'échantillonnage. Une fois que le nombre de cycles voulus, n, est atteint, l'horloge initiale est inhibée et un nouvel instant d'échantillonnage est attendu.

Ainsi, il est possible de dire que les échantillons arrivent de manière périodique dans l'architecture ce qui perment d'envisager plusieurs modèles de communications.

IV.2 Modélisation des communications

À partir des caractéristiques des applications étudiées, on peut définir quelques contraintes dans la conception de l'architecture et en particulier dans la manière dont les composants de l'architecture vont communiquer entre eux.

Il s'agit d'abord de choisir entre un modèle de communications dynamiques et un modèle de communications statiques. Le principal avantage du modèle statique est qu'il permet de valider facilement le sytème. En effet, l'ensemble des communications étant prévu à l'avance, on est sûr qu'aucun blocage ne risque d'arriver pendant le déroulement du programme. Mais, il faut d'abord savoir si ce modèle peut être utilisé pour la réalisation des applications cibles, parce qu'il n'est pas toujours applicable.

Tout d'abord, le flot de calcul est indépendant des données et prédictible. Par contre, il n'est pas vraiment périodique du fait de l'utilisation d'une fréquence symbole variable dans les applications cibles, cette fréquence pouvant prendre une plage importante de valeurs dans les applications cibles. Pour le rendre périodique (une donnée communiquée tous les n cycles d'horloge), il serait possible d'utiliser différentes fréquences de calcul, multiples de la fréquence d'échantillonnage ou de la fréquence symbole, selon que les calculs se font en rapport avec la fréquence d'échantillonnage ou avec la fréquence symbole. Mais dans ce cas, il faudrait vraiment séparer l'architecture en 2 parties, ce qui demanderait un surdimensionnement de l'architecture.

Une autre solution est de prévoir les communications pour le pire cas (la fréquence symbole la plus élevée, la puissance de calcul la plus forte) et de n'utiliser qu'une partie de ces communications dans les autres cas. Il suffit pour cela de rajouter un drapeau avec les données pour les valider, c'est-à-dire pour indiquer à un composant si la donnée qu'il reçoit périodiquement est utilisable ou non. Ce drapeau est appelé le drapeau "valide" puisqu'il sert à valider la communication.

Le but du drapeau valide est donc d'introduire un peu de dynamisme (juste ce qui est nécessaire pour gérer la fréquence symbole variable des démodulations monoporteuses étudiées).

IV.2.1 Mise en œuvre du drapeau "valide" [92]

L'intérêt du drapeau "valide" est présenté *figure IV.6*. Des instants périodiques de communications ($t0$, $t1$...) sont prévus pour transférer des données d'un composant à un autre. Mais, il arrive qu'aucun résultat ne soit disponible alors qu'une communication doit avoir lieu (à l'instant $t2$). Il faut en informer le composant suivant. Un drapeau est donc transmis à chaque instant de communication pour informer les composants attendant une donnée s'il y en a effectivement une pour eux. Ainsi, à l'instant $t2$, comme aucune donnée n'est transmise, ce drapeau a une valeur nulle.

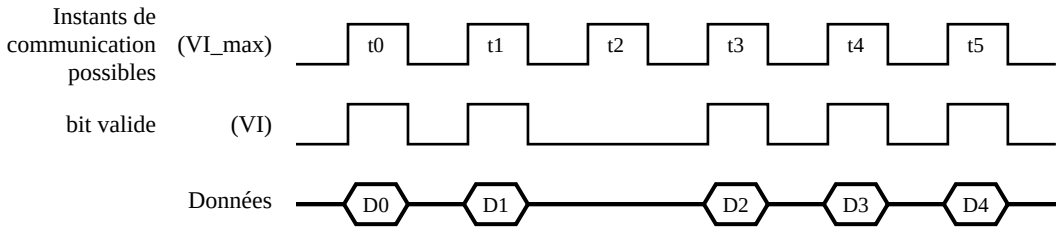


FIGURE IV.6 : *Exemple d'utilisation du drapeau "valide"*

Le drapeau valide permet donc de donner des informations supplémentaires en étant transmis en même temps que la donnée. Il faut maintenant voir comment sont traitées ces informations supplémentaires par les différents éléments de l'architecture, à savoir les multiplexeurs qui composent le réseau d'interconnexion et les unités de calculs.

IV.2.1.a Le drapeau valide et les multiplexeurs

Comme les communications entre les composants se font sans conflit, les multiplexeurs qui orientent les transferts de données d'un composant à un autre à travers le réseau d'interconnexion n'ont pas de conflit à gérer dynamiquement. Un programme cyclique peut-être déterminé à l'avance pour décider, à chaque instant, quelle entrée sera redirigée vers la sortie *(Figure IV.7)*.

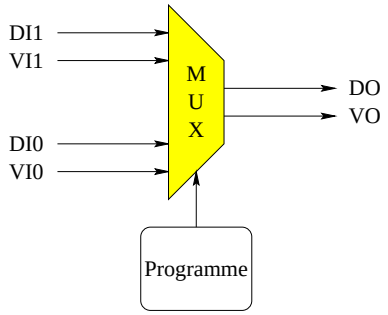


FIGURE IV.7 : *Multiplexeur pour les communications*

Le drapeau valide suit donc le même chemin que la donnée, le multiplexeur ne fait pas de différence entre une donnée valide et une donnée non valide. C'est exactement la même chose pour les registres qui constituent le reste du réseau d'interconnexion.

IV.2.1.b Le drapeau valide et les unités de calcul

Pour les blocs de calcul, un programme est nécessaire pour donner les instructions au bon moment, mais le "drapeau valide" peut également avoir un rôle dans le comportement du bloc *(Figure IV.8)*. C'est pour cela que, sur le schéma, le drapeau valide VI est associé à l'instruction venant du programme de contrôle.

Dans la majorité des cas, le drapeau valide va inhiber l'instruction, comme pour le filtrage. Si une donnée non valide arrive dans un CCP, celui-ci continue à faire les pré-calculs qu'il faisait, comme si aucune donnée n'arrivait.

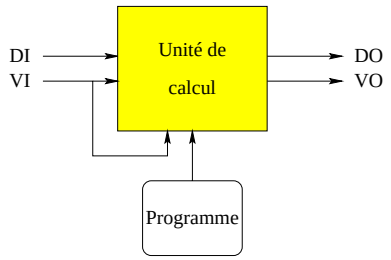


FIGURE IV.8 : *Mise en œuvre du drapeau valide dans les composants*

Dans d'autres cas, l'unité de calcul est un générateur de données, c'est-à-dire qu'elle n'a pas besoin de recevoir des données pour effectuer des calculs. Les oscillateurs contrôlés numériquement qui sont placés dans les boucles de contre-réaction sont de telles fonctions. Dans ce cas, les instructions et le comportement du bloc ne sont en aucun cas modifiés par une donnée valide ou non.

IV.2.1.c Le drapeau valide et la décimation

La réservation des créneaux de communications s'effectue en supposant que toutes les données transmises sont valides. Ainsi, lorsqu'il y a une décimation par n dans un composant, un résultat est obtenu chaque fois que n données ont été reçues. Il suffit donc de ne réserver qu'un seul créneau sur n pour les communications futures. Or, l'utilisation du drapeau valide rend aléatoire l'arrivée des $n^{ièmes}$ données et donc la génération d'un résultat. Les créneaux de communications théoriques n'ont alors aucune assurance d'être toujours à la bonne position comme le montre la *figure IV.9*.

En effet, VO_{max} qui représente l'ensemble des réservations des communications théoriques pour la sortie des résultats a été obtenu en réalisant une décimation par 2 sur les communications théoriques en entrée du bloc (VI_{max}). Mais il ne couvre pas l'ensemble des besoins revendiqués par les communications pratiques initiales représentées par VO_i qui découlent des résultats à partir des données reçues par le bloc et validées par VI. Il est donc nécessaire d'être capable de retarder certaines sorties d'un composant pour qu'elles puissent utiliser les créneaux de communications qui leur sont attribués comme le fait VO_f. Par exemple, pour une décimation par 2, si le nombre

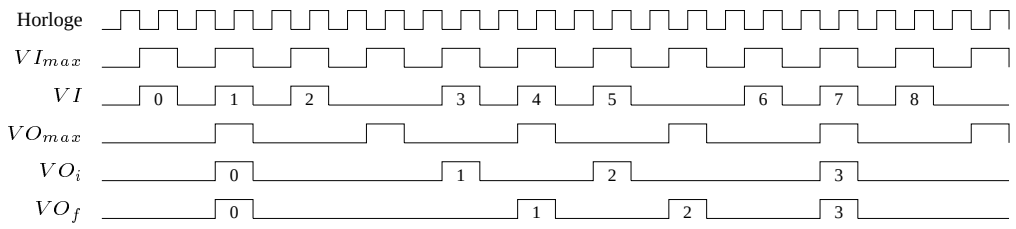


FIGURE IV.9 : *Influence des créneaux de communications et de la décimation sur la conception des composants*

de données non valides reçues est impair, il y a un décalage au niveau de la sortie et il faut alors retarder cette sortie pendant un temps correspondant à la période d'arrivée des données.

Le drapeau valide a été mis au point pour s'occuper des problèmes rencontrés à cause des fréquences symboles variables dans les démodulations monoporteuses. Dans les démodulations multiporteuses, par contre, la fréquence symbole est fixe, mais très petite, ce qui réduit la périodicité des calculs et augmente par conséquent la taille des programmes de contrôle. Une solution doit être trouvée à ce problème afin de conserver une périodicité des calculs élevée.

IV.2.2 Des communications à double vitesse [93]

Dans les applications de démodulation monoporteuse, la fréquence d'échantillonnage en entrée du circuit et la fréquence symbole en sortie du circuit sont du même ordre de grandeur. Il est donc possible de trouver une périodicité dans les communications.

Par contre, dans les applications de démodulation multiporteuse, un symbole correspond à une multitude d´échantillons (jusqu'à 8192 dans la démodulation terrestre). La périodicité de certaines applications est ainsi extrêmement faible, rendant inacceptable la périodicité globale de l'application puisque certains algorithmes fonctionnent à la fréquence symbole.

Pour résoudre ce problème, il est possible, soit de permettre d'utiliser deux vitesses différentes dans les communications en réservant des instants de communications aux différents types de fonctions, soit de séparer les fonctions s'effectuant à la fréquence symbole de celles s'effectuant à la fréquence d'échantillonnage, et d'utiliser une interface entre les deux parties.

La première solution utilise deux programmes, l'un pour les communications et les exécutions à haut débit à la fréquence d'échantillonnage et l'autre pour ceux à la

fréquence symbole, un créneau temporaire étant réservé à chaque programme pour les communications et le lancement des calculs *(Figure IV.10)*. Pour cela, il suffit d'utiliser une machine à états composée principalement d'un compteur modulo la périodicité des communications et qui activera l'instruction émise par l'un ou l'autre des programmes en fonction de la répartition des créneaux choisie sur la période.

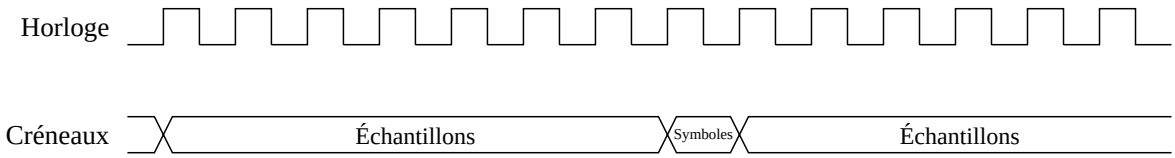


FIGURE IV.10 : *Répartition des créneaux de communications*

La deuxième solution consiste à séparer l'architecture en 2 parties distinctes. La première, constituée des composants et du réseau d'interconnexion définis pour les démodulations câble et satellite, s'occupe des calculs à haut débit. La seconde, constituée de la mémoire nécessaire aux algorithmes s'effectuant à la fréquence symbole de la démodulation terrestre, utilise un processeur programmable pour réaliser ces calculs. L'interface entre les deux parties est réalisée avec des FIFO *(Figure IV.11)*.

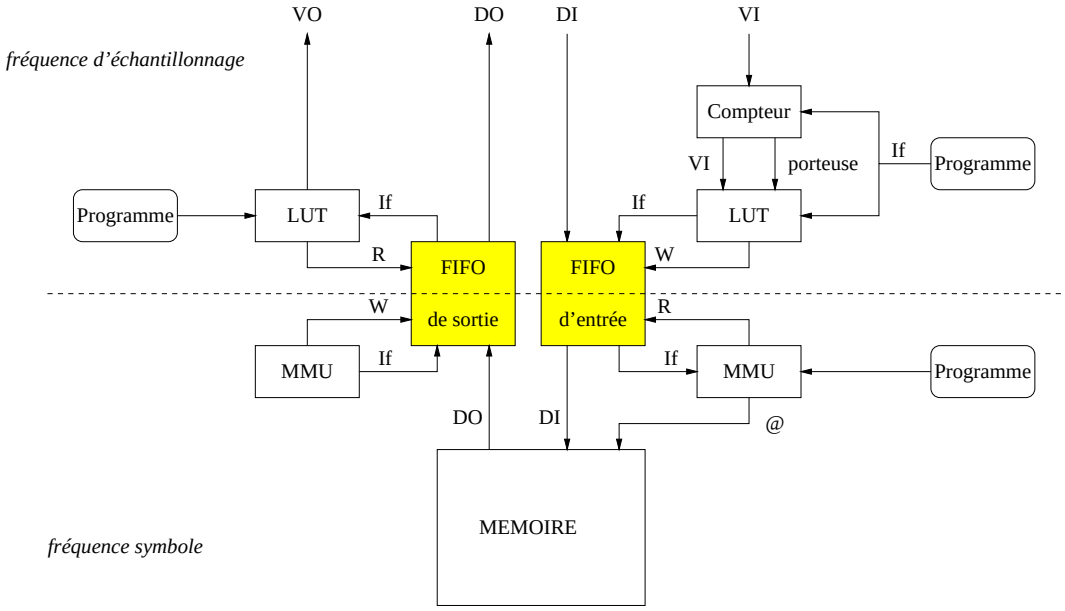


FIGURE IV.11 : *Interface haut débit/bas débit*

Lors du passage de données de la partie haut débit à la partie bas débit, la FIFO d'entrée est utilisée. Pour gérer l'écriture dans cette FIFO, un compteur et une "look up table" (LUT) effectuent le tri sur les données qui doivent effectivement traverser l'interface. En effet, les algorithmes fonctionnant à la fréquence symbole utilisent souvent une petite partie des données (les porteuses pilotes). Le compteur indique, en comptant le nombre de drapeaux valides, le numéro de la porteuse qui est arrivée et la LUT regarde dans une table pour savoir s'il s'agit d'une porteuse à transférer. L'identifieur If de la fonction ayant envoyé la donnée est également conservé en même temps que la

donnée puisqu'en fonction de cet identifieur, l'unité de gestion de la mémoire (MMU : Memory Management Unit) pourra calculer l'adresse @ où la donnée devra être stockée en mémoire.

Lors du passage inverse, de données de la partie bas débit à la partie haut débit, la FIFO de sortie est utilisée. La MMU (il n'y en a qu'une seule, même si elle apparaît deux fois sur le schéma) transfère une donnée de la mémoire vers la FIFO et lui ajoute un identifieur. Au moment de la lecture de la donnée, une LUT regarde si c'est bien une donnée avec cet identifieur qui doit être introduite dans l'architecture statique. Si ce n'est pas le cas, la donnée reste dans la FIFO jusqu'au moment propice de lecture.

La deuxième solution permet de ne pas transformer l'architecture de la partie principale travaillant à la fréquence d'échantillonnage. L'interface, la mémoire et le processeur programmable pouvant être considérés comme une simple unité de calcul.

Maintenant que l'ensemble des problèmes liés à la conception de l'architecture P-DCP a été résolu, il reste à estimer la complexité matérielle de cette architecture.

IV.3 Modélisation de la complexité matérielle

Jusqu'à présent, l'architecture P-DCP a été évoquée de manière générique, sans se préoccuper précisément de ses éléments constitutifs. Cette section va donc présenter un exemple concret de l'architecture générique visant à supporter l'ensemble des applications cibles, à savoir la démodulation câble, satellite et terrestre, définies par le consortium européen DVB.

En fait, le CCP, dans sa version actuelle, n'est pas encore capable de faire les FFT présentes dans la démodulation terrestre. Comme la réalisation de cette démodulation terrestre demande également l'utilisation d'un cœur de DSP qu'il faudra choisir au moment même de la réalisation matérielle de l'architecture, l'estimation suivante correspond plutôt à un démodulateur supportant le câble et le satellite.

IV.3.1 Estimation de la complexité matérielle des composants

Le CCP représentant l'essentiel de la complexité de l'architecture complète puisqu'il réalise l'essentiel des opérations, l'estimation de son coût matériel a été faite à partir d'une description VHDL. Pour les autres composants de l'architecture, on se contentera d'une estimation grossière en fonction du nombre d'opérateurs et de registres en utilisant les valeurs du *tableau IV.1*.

L'élément de base d'un registre est une bascule D qui est réalisée avec 26 transistors soit l'équivalent de 7 portes logiques. Ainsi, un registre de n bits a une complexité de

Opérateur	Complexité
Registre n bits	$7 \times n$
Multiplexeur n bits	$3 \times n$
Additionneur n bits	$7 \times n$
Accumulateur n bits	$14 \times n$
Compteur n bits	$11 \times n$
Comparateur n bits	$7 \times n$
Multiplieur $n \times n$ bits	$9 \times n^2$

TABLEAU IV.1: *Complexité des opérateurs de base en nombre de portes NAND*

$7 \times n$ portes équivalentes.

L'élément de base d'un additionneur est le "Full Adder" qui effectue une addition de 1 bit avec la retenue venant du "Full Adder" précédent. Un "Full Adder" est réalisé avec 28 transistors soit l'équivalent de 7 portes logiques. On supposera dans les calculs que chaque additionneur est suivi d'un registre qui sert de niveau de pipeline avant la suite des calculs. Ainsi, un additionneur de n bits a une complexité de $14 \times n$ portes équivalentes.

Un compteur est un additionneur avec une opérande fixe (=1) et un registre pour conserver le résultat. Il est donc possible d'utiliser un "Half Adder" (4 portes) plutôt que des "Full Adder" si bien que la complexité est de $11 \times n$ portes équivalentes.

Un accumulateur est un additionneur dont la sortie reboucle sur une entrée de l'additionneur en passant par un registre. La complexité est donc la même que pour un compteur.

Le comparateur réalise une soustraction (une addition) et analyse le résultat obtenu. Il a donc la complexité d'un additionneur tout seul, soit $7 \times n$ portes équivalentes.

Un multiplieur $n \times n$ bits calcule n additions n bits. Il utilise donc n^2 Full Adder. Il faut également n^2 portes "ET" pour générer les produits partiels. Si l'on ajoute les niveaux de pipelines ainsi que les structures de multiplieurs rapides qu'il faudra utiliser pour atteindre la vitesse requise, il est possible de prendre une complexité approximative de $9 \times n^2$ portes équivalentes.

Avec la complexité des différents blocs de base utilisés pour faire les calculs, il est possible de déterminer une estimation de la complexité des différents blocs de calcul qui composent l'architecture P-DCP. On supposera que toutes les données d'entrées passent par une couche de registres avant d'être utilisées.

IV.3.1.a Le contrôle automatique de gain

Cette boucle est en partie analogique et en partie numérique. La partie numérique se contente de calculer la puissance du signal en effectuant une sommation de la valeur absolue du signal. Ce bloc de calcul se compose donc d'un compteur 8 bits et d'un accumulateur 24 bits en plus d'un registre en entrée soit une complexité de 536 portes équivalentes *(Tableau IV.2)*.

Opérateur	Nombre	Coût unitaire	Total
Registre 16 bits	1	112	112
Accumulateur 24 bits	1	336	336
Compteur 8 bits	1	88	88
Total			536

TABLEAU IV.2: *Complexité du bloc de contrôle automatique de gain*

IV.3.1.b La récupération d'horloge

Ce bloc de calcul se compose d'un détecteur de Gardner d'un filtre de boucle, d'un NCO et du mélangeur de l'interpolateur qui sont définis au *chapitre I*.

Le détecteur de Gardner

Ce bloc réalise l'*équation I.3 page 15*. Il faut donc prévoir 2 multiplieurs, 3 additionneurs et 6 registres pour les données d'entrée, soit une complexité de 6 milliers de portes *(Tableau IV.3)*.

Opérateur	Nombre	Coût unitaire	Total
Registre 16 bits	6	112	672
Multiplieur 16 bits	2	2304	4608
Accumulateur 16 bits	3	224	672
Total			5952

TABLEAU IV.3: *Complexité du détecteur de Gardner*

Le filtre de boucle

Il s'agit d'un filtre récursif du premier ordre. Il faut utiliser l'*équation I.4 page 15* avec comme valeur $M = 1$ et $N = 2$. En utilisant la technique des pré-calculs définie lors de la conception du CCP, il est possible de ne prévoir qu'un seul multiplieur, un accumulateur et 2 registres, soit une complexité de 2,8 milliers de portes *(Tableau IV.4)*.

Opérateur	Nombre	Coût unitaire	Total
Registre 16 bits	2	112	224
Multiplieur 16 bits	1	2304	2304
Accumulateur 16 bits	1	224	224
Total			2752

TABLEAU IV.4: *Complexité du filtre de boucle*

L'oscillateur contrôlé numériquement (NCO)

Ce bloc se compose d'un compteur et d'un comparateur en plus d'un registre pour la donnée d'entrée, soit une complexité de 400 portes équivalentes *(Tableau IV.5)*.

Opérateur	Nombre	Coût unitaire	Total
Registre 16 bits	1	112	112
Compteur 16 bits	1	176	176
Comparateur 16 bits	1	112	112
Total			400

TABLEAU IV.5: *Complexité du NCO*

Le mélangeur de l'interpolateur

Ce bloc met en œuvre le mélangeur de l'interpolateur qui est représenté *figure I.6 page 11* avec 2 multiplications et 4 additions sur des données complexes. Comme les calculs doivent être consécutifs (les résultats d'une opération étant les opérandes des autres opérations), il est possible d'itérer les 2 multiplications et additions finales sur un seul multiplieur et additionneur. En supposant en plus des registres sur toutes les données d'entrées, on obtient une complexité de 6,8 milliers de portes équivalentes *(Tableau IV.6)*.

Opérateur	Nombre	Coût unitaire	Total
Accumulateur 16 bits	6	224	1344
Registre 16 bits	7	112	784
Multiplieur 16 bits	2	2304	4608
Total			6736

TABLEAU IV.6: *Complexité du mélangeur de l'interpolateur*

Le bloc de calcul mettant en œuvre la récupération d'horloge a donc une complexité de 15,8 milliers de portes équivalentes.

IV.3.1.c La récupération de porteuse

Ce bloc de calcul réalise 2 boucles à verrouillage de phase numérique. Il utilise donc 2 détecteurs différents (des tables analysant la position des symboles dans une constellation), 2 filtres de boucles, 2 NCO et 2 tables de sinus/cosinus qui transforment la phase du NCO en un nombre complexe.

Ce bloc de calcul a donc une complexité de 6,3 milliers de portes équivalentes plus les tables pour les détecteurs et la génération des sinus/cosinus.

IV.3.1.d L'égaliseur

Ce bloc de calcul réalise les opérations qui ne sont pas du filtrage dans le calcul de l'égaliseur récursif avec décision dans la boucle *(Figure I.7 page 12)*. Il est donc composé de deux additionneurs complexes et d'un bloc de décision qui consiste juste à prendre les bits de poids fort de la donnée. Il y a également 4 registres pour les données d'entrée complexes et 6 autres registres qui suivent les additionneurs et le bloc de décision soit un total de 1,6 milliers de portes équivalentes *(Tableau IV.7)*.

Opérateur	Nombre	Coût unitaire	Total
Additionneur 16 bits	4	112	448
Registre 16 bits	10	112	1120
Total			1568

TABLEAU IV.7: *Complexité du bloc de calcul d'aide pour l'égaliseur*

IV.3.2 Le CCP

En premier lieu, il faut définir le nombre de CCP qui seront nécessaires pour effectuer l'ensemble des opérations de filtrage pour les différentes applications. Ensuite, en fonction de la répartition des filtres sur les différents CCP, il sera possible de déterminer les capacités de mémorisations qui seront nécessaires et donc la complexité des CCP.

IV.3.2.a Calcul du nombre de CCP nécessaires

Pour calculer le nombre de CCP qui devront être intégrés dans l'architecture, il ne suffit pas de diviser la puissance de calcul requise par le filtrage par la puissance de calcul que peut développer un CCP. En effet, le parallélisme des calculs entraîne des effets de bords qui rendent le CCP un peu moins performant que son maximum. Pour trouver le nombre de CCP requis, il faut calculer le nombre de cycles nécessaires pour effectuer les différents filtres et connaître le nombre de cycles de calcul dont dispose le CCP.

Fonctions	**Nombre de cycles**		**Fréquence des calculs**	
	Câble	Satellite	Câble	Satellite
Démodulation IQ	1		Fe	$\frac{Fe}{2}$
Filtrage anti-repliement	5		$\frac{F_e}{2^k}$	$\frac{F_e}{2^{k+1}}$
Filtre de l'interpolateur	3+1	2+1	$\frac{F_e}{2^k}$	$\frac{F_e}{2^{k+1}}$
Filtre de Nyquist	9	4	$2 \times F_s$	F_s
Multiplication complexe	1		F_s	$\frac{F_s}{2}$
Égaliseur	8+4	-	F_s	$\frac{F_s}{2}$

TABLEAU IV.8: *Puissance de calcul requise pour les algorithmes de filtrage*

Le *tableau IV.8* donne le nombre de cycles requis pour le calcul de chaque filtre, ainsi que la périodicité de ces calculs. Comme pour le satellite, une précision de 8 bits est utilisée, il est possible d'utiliser le parallélisme de sous-mots dans les CCP. Cela permet de diviser par 2 la fréquence des calculs puisque les données arrivent deux par deux.

Comme la puissance de calcul requise n'est pas une fonction totalement linéaire de la fréquence symbole, les pires cas ne correspondent pas forcément à la fréquence symbole la plus élevée.

Pour la démodulation câble, le pire cas apparaît lorsque $F_s \lesssim \frac{F_e}{2(3-\alpha)}$ où α est le roll-off du filtre de Nyquist (α a pour valeur 0,35 ou 0,5). Dans ce cas, deux filtres anti-repliement à la fréquence de $\frac{F_e}{2}$ sont nécessaires, et la fréquence symbole pour les estimations doit être $\frac{F_e}{4}$ qui correspond à un facteur de roll-off de 0. Il en résulte que les CCP doivent pouvoir exécuter N_{iter} itérations à une fréquence de $\frac{F_e}{4}$ avec :

$$N_{iter} = 4 \times 1 + (2+2) \times 5 + 2 \times (3+1) + 2 \times 9 + 1 \times 1 + (8+4) \times 1 = 63$$

Le nombre de CCP nécessaires est donc $n_{CCP} = \left\lceil 63 \times \frac{F_e}{4} \times \frac{1}{F_T} \right\rceil$ où F_T est la fréquence de travail. Il a été expliqué à la *section IV.1.4* que la fréquence de travail devait être un multiple de la fréquence d'échantillonnage. Comme $F_e = 28,92$ MHz et que F_T ne doit pas dépasser 130 MHz, il en résulte que $F_T = 4 \times F_e$ et $n_{CCP} = 4$ pour la démodulation câble.

Pour la démodulation satellite, le pire cas apparaît lorsque $F_s = 24$ MHz. Dans ce cas, un filtre anti-repliement à $\frac{F_e}{2}$ est nécessaire. Il en résulte que

$$n_{CCP} = \left\lceil \frac{F_e}{2F_T} + 5 \times \frac{F_e}{2F_T} + (2+1) \times \frac{F_e}{2F_T} + 4 \times \frac{F_s}{F_T} + \frac{F_s}{2F_T} \right\rceil \quad \text{(IV.1)}$$

$$= \left\lceil \frac{9F_e}{2F_T} + \frac{9F_s}{2F_T} \right\rceil = \left\lceil \frac{27}{8} \right\rceil = 4 \quad \text{(IV.2)}$$

avec $F_T = 2 \times F_E$ et $F_T = 4 \times F_s$. Soit, de nouveau, il faut 4 CCP pour réaliser les calculs.

IV.3.2.b Complexité du CCP

Lors de l'estimation de la complexité du CCP réalisée à la *section III.4*, il est apparu que cette complexité dépendait fortement de la taille des mémoires intégrées dans celui-ci. En fonction de la faible taille des filtres à réaliser (il suffit de voir le nombre de cycles d'utilisation du CCP dans le *tableau IV.8*), une taille de mémoire de 20 coefficients est suffisante. Avec cette valeur on obtient une estimation de 40000 portes équivalentes pour un CCP *(Figure III.25)*.

IV.3.2.c Réalisation de la FFT

Il est prévu de réaliser les FFT sur les CCP qui sont capables de réaliser un papillon de radix-2 à chaque cycle *(section III.2.2.c page 47)*.

Pour réaliser la démodulation par voie hertzienne, il est nécessaire de pouvoir réaliser une FFT de 8192 points tous les symboles ainsi qu'une FFT inverse de 4096 points pour la synchronisation.

Pour réaliser une FFT sur 2^n points, il faut effectuer $n \times 2^{n-1}$ papillons. Ainsi pour la FFT de 8192 points, il faut effectuer 13×4096 papillons pendant la réception de 8192 échantillons à 9,14 MHz ce qui correspond au calcul de 6,5 papillons pour chaque échantillon reçu. Or, avec une fréquence de travail de 130 MHz, on dispose de 14 cycles entre deux échantillons, ce qui est largement suffisant pour faire l'ensemble des calculs de FFT sur un seul CCP.

IV.3.3 Estimation de la complexité du réseau d'interconnexion

Une fois que l'on a l'ensemble des composants de l'architecture, on peut ajouter le réseau d'interconnexion final qui permet de relier les différents blocs. On obtient ainsi une architecture découlant de l'architecture générique P-DCP qui doit être en mesure de supporter la démodulation DVB *(Figure IV.12)*. Il est alors possible de compter les registres et les multiplexeurs du réseau d'interconnexion.

Dans la *figure IV.12*, le bloc de calcul de récupération d'horloge est découpé en deux parties pour que les communications entre les CCP et les boucles soient les plus rapides possibles. En effet, il est prévu de réaliser les filtres de l'interpolateur dans le CCP 1 et le filtre de Nyquist dans le CCP 2. Par contre plusieurs blocs de calcul sont directement reliés au dernier CCP, à savoir le bloc de calcul effectuant la récupération

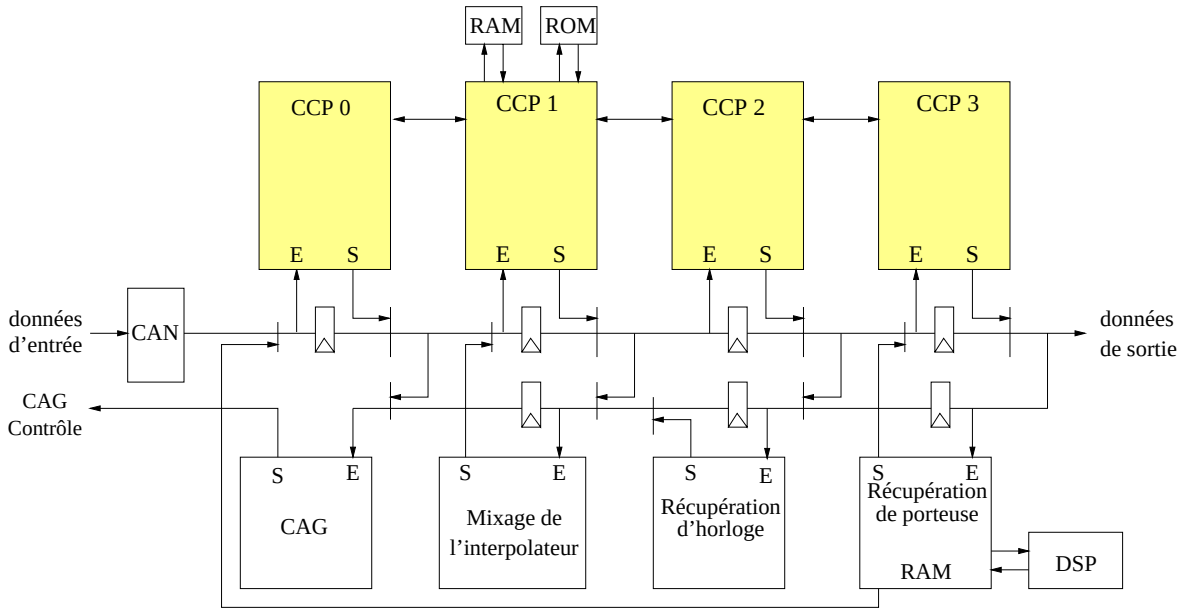


FIGURE IV.12 : *Architecture P-DCP pour la démodulation DVB*

de porteuse ainsi que le bloc de calcul complémentaire de l'égaliseur puisque celui-ci doit être exécuté dans ce dernier bloc de calcul. L'unité de calcul composée de l'interface haut débit/bas débit, d'une mémoire et d'un processeur général (DSP) sont aussi reliés à ce dernier CCP.

D'après la figure, on peut compter 11 multiplexeurs et 7 registres complexes ce qui donne une complexité de 2,6 milliers de portes *(Tableau IV.9)*.

Opérateur	Nombre	Coût unitaire	Total
Multiplexeur 16 bits	22	48	1056
Registre 16 bits	14	112	1568
Total			2624

TABLEAU IV.9: *Complexité du réseau d'interconnexion*

IV.3.4 Taille du programme de contrôle

Il faut d'abord déterminer le nombre de bits qu'il faut pour contrôler chaque composant de l'architecture (multiplexeurs et blocs de calcul) et ensuite il faut définir la périodicité des calculs.

Pour les multiplexeurs, 1 bit de contrôle suffit. Pour les CCP, cela dépend du nombre de filtres que l'on veut pouvoir multiplexer. Un maximum de 7 filtres (3 bits) est suffisant. Pour les 6 autres composants, 1 ou 2 bits de contrôle sont suffisants. Cela donne un mot de commande de 35 bits.

Autant pour la puissance de calcul, les pires cas correspondaient à des fréquences symboles élevées, autant pour déterminer la périodicité des calculs, les pires cas sont les fréquences symboles basses. En effet, plus la fréquence symbole est faible, plus il y a de cycles entre 2 symboles transmis. Comme pour la démodulation câble, la périodicité s'apparente à la fréquence symbole, la fonction la plus lente s'exécutant à cette fréquence, on obtient une périodicité de 64 cycles au maximum. En effet, pour une fréquence de travail autour de 130 MHz, il y a un nouvel échantillon tous les 4 cycles avec au plus 3 décimations par deux dans le filtrage anti-repliement et une après le Nyquist, ce qui donne effectivement une périodicité de 64 cycles.

On obtient un programme d'une taille de 280 octets seulement (64 mots de 35 bits).

IV.3.5 Complexité totale

Pour obtenir la complexité totale de l'architecture, il suffit de regrouper ensemble les différentes estimations calculées dans ce chapitre. On obtient ainsi une complexité de 187k portes équivalentes *(Tableau IV.10)*. À cette estimation, il faut aussi ajouter le programme de contrôle de 280 octets.

Bloc de calcul	Nombre	Coût unitaire	Total
CCP	4	40,0k	160,0k
CAG	1	0,6k	0,6k
Récupération d'horloge	1	15,8k	15,8k
Récupération de porteuse	1	6,3k	6,3k
Egaliseur	1	1,6k	1,6k
Réseau d'interconnexion	1	2,6k	2,6k
Total			186,9k

TABLEAU IV.10: *Complexité du P-DCP*

Cette estimation ne tient pas compte des tables utilisées dans les boucles de contre-réaction, ni des besoins supplémentaires de la démodulation terrestre, à savoir 1,5 Mo de mémoire et un processeur général développant aux alentours de 60 Mops.

Cette estimation ne tient pas non plus compte de la logique qui sera nécessaire pour configurer les différents blocs de calcul en particulier les CCP, ni de la mémoire qui contiendra les différents programmes de contrôle en fonction de l'application à réaliser et du mode de fonctionnement de l'application choisie.

Mais le résultat de cette estimation est encourageant. En effet le gain réalisé en ce qui concerne la mise en œuvre de l'égaliseur dans les CCP en réutilisant les multiplieurs, permet de compenser le surcoût dû à la flexibilité, si bien que le démodulateur obtenu est comparable en complexité avec d'autres démodulateurs [25, 33].

IV.4 Conclusion

Une architecture à base de coprocesseurs a été définie pour prendre en compte les importantes puissances de calcul requises par la démodulation numérique.

Afin de profiter des avantages de validation offerts pas l'utilisation de communications statiques dans une architecture, un protocole spécifique utilisant un drapeau valide a été créé pour supporter les fréquences symboles variables des démodulations câble et satellite.

Un réseau d'interconnexion spécialement adapté à la circulation des données a également été défini afin de réduire au minimum le coût des communications, puisque la latence est une contrainte forte dans les applications étudiées.

Finalement, une architecture estimée à moins de 200k portes équivalentes devrait être à même de réaliser une démodulation par câble ou satellite. Pour pouvoir s'occuper de la démodulation terrestre, un puissant cœur de DSP et une mémoire de 150 Ko doivent être prévus.

Maintenant que l'architecture est définie, il faut prouver qu'elle est en effet capable de réaliser ce que l'on attend d'elle.

Chapitre V

Validation et résultats

Après avoir analysé les applications de "décodage canal" à mettre en œuvre dans notre architecture *(Chapitre I)*, défini les caractéristiques d'un coprocesseur de filtrage *(Chapitre III)* et décrit et dimensionné l'architecture globale autour de ce coprocesseur *(Chapitre IV)*, il nous reste maintenant à valider globalement l'architecture résultante. L'objectif de ce chapitre est donc de prouver la validité de l'architecture. Une première étapes est donc de supporter les applications câble et satellite qui sont similaires. L'étape suivante consistera à mettre en œuvre la démodulation terrestre qui requiert l'utilisation d'une mémoire et d'un processeur général.

Cette validation est nécessaire puisque l'approche utilisée dans les chapitres précédents a consisté à définir l'architecture en résolvant localement les problèmes un à un ce qui ne garantit pas le fonctionnement de l'ensemble. Le CCP a été conçu pour réaliser le filtrage au chapitre III. Le "bit valide" a été ajouté pour prendre en compte les problèmes de fréquence symbole variable dans la *section IV.2.1*. Le comportement des composants a été modifié à cause de l'utilisation simultanée de la décimation et du "bit valide" dans la *section IV.2.1.c*.

Il faut maintenant vérifier, non seulement que l'application peut effectivement fonctionner sur l'architecture telle qu'elle est définie, mais aussi que la taille des mémoires tampons est raisonnable (2 ou 3 registres par composant) ainsi que la taille des programmes (une dizaine d'instructions par composant). Alors, on aura prouvé que le système défini est opérationnel.

Après avoir présenté la méthodologie utilisée pour valider notre travail, nous présenterons les outils existant pour la synthèse d'architecture. Après un constat d'échec sur l'usage de ces outils, nous détaillerons les outils qu'il a fallu développer pour mettre en œuvre la méthodologie choisie, puis les résultats obtenus.

V.1 Méthodologie de validation de l'architecture

La validation de l'architecture consiste à réaliser une simulation comportementale des différentes applications sur l'architecture. Cela permettra d'étudier les performances de l'architecture (efficacité du réseau d'interconnexion et de la synchronisation (contraintes de faible latence)) et de vérifier que tous les algorithmes des applications ont bien été pris en compte dans la conception de l'architecture.

La méthodologie retenue pour valider l'architecture s'effectue en 3 étapes : une étape de description des applications, de l'architecture et de leur interaction, une étape d'ordonnancement des calculs et une étape de simulation comportementale de l'application sur l'architecture.

La première étape consiste à créer une représentation nécessaire et suffisante des informations dont l'on dispose sur les éléments constitutifs des applications et de l'architecture. Il s'agit de déterminer les caractéristiques qui permettront ensuite d'effectuer un ordonnancement des calculs sur l'architecture décrite. Cette étape est étudiée en détail dans la *section V.3*.

La deuxième étape, l'ordonnancement des calculs, sert à déterminer les conflits possibles entre les ressources pour estimer les pertes de latence dues à ces conflits et connaître précisément les instants d'exécution de tous les algorithmes. Il est alors possible de se faire une idée sur les performances de l'architecture, sans pour autant être assuré de l'efficacité de la démodulation. Cette étape permet également de générer les programmes de chacun des composants. Elle est étudiée en détail dans la *section V.4*.

La troisième étape, la simulation comportementale, sert à vérifier que l'ensemble des algorithmes de l'application a bien été pris en compte dans les étapes précédentes et que l'architecture réalise bien ce qui lui était demandé. Il est alors possible de comparer les performances d'une démodulation effectuée avec cette architecture par rapport à celles effectuées par des solutions dédiées. Elle est étudiée en détail dans la *section V.5*.

Cette méthodologie ressemble à celle utilisée dans les outils de synthèse d'architecture si ce n'est, que ces outils comportent une étape supplémentaire qui consiste à concevoir également l'architecture avant de la valider. Par conséquent, il est important de s'intéresser à ces outils pour savoir si leur utilisation pourrait nous être utile.

V.2 La synthèse d'architecture

Le but des outils de synthèse d'architecture est de concevoir une architecture à partir d'un algorithme. Les outils sont généralement limités par le choix d'un type d'architecture cible prédéfini afin de restreindre l'espace de recherche. Pour réaliser la synthèse, les outils effectuent 4 étapes principales qui sont la sélection et l'allocation des

ressources, l'ordonnancement des calculs et l'assignation des calculs aux ressources [94].

- La sélection des ressources consiste à choisir dans une bibliothèque de composants préexistants ceux qui sont appropriés pour l'application considérée, en tenant compte des contraintes éventuelles de temps et de coût.
- L'allocation des ressources consiste à déterminer, pour chaque ressource sélectionnée, le nombre d'instances de cette ressource qu'il faudra mettre en œuvre en fonction du parallélisme de calcul et des contraintes de temps.
- L'assignation des calculs aux ressources consiste à affecter l'ensemble des calculs aux ressources utilisées afin de réduire au minimum les besoins de communications entre les ressources ainsi que les ressources mémoires à utiliser.
- L'ordonnancement des calculs a pour fonction de déterminer les instants précis où se déroule l'ensemble des opérations de l'application en fonction du parallélisme utilisé. Il est ainsi possible dans cette étape de modifier la représentation de l'application pour profiter au mieux de ce parallélisme et pour satisfaire aux contraintes de temps.

L'agencement de ces différentes étapes diffère selon les outils en fonction des techniques d'optimisation choisies. Le résultat cette synthèse est une architecture décrite dans un language de description matériel (type VHDL) qui permet de simuler le résultat obtenu.

Les trois premières étapes de la synthèse (sélection, allocation, assignation) servent à contruire l'architecture en fonction d'un ensemble de composants. Dans notre cas, ces différentes étapes ont été réalisées lors de la conception de notre architecture.

Les différents outils de synthèse d'architecture peuvent être séparés en 2 groupes.

Un premier groupe comprend les outils de synthèse d'ASIP (Application Specific Integrated Processor). C'est la voie la plus étudiée pour l'instant avec des logiciels comme AMICAL [95], CADDY [96], GAUT [97], HYPER [98], MARS [99], MISTRAL [100], OSCAR [101]... La brique de base de ces outils est le multiplieur, l'additionneur ou le registre. Ces outils peuvent être utilisés pour la conception d'unités de calcul spécialisées, mais ne peuvent pas être utilisés pour modéliser le coprocesseur de filtrage qui est trop complexe puisqu'il doit supporter un nombre important de fonctionnalités.

Un second groupe comprend les outils de synthèse de systèmes complets (un ensemble de fonctions) comme CASCH [102], PHIDEO [103], SYNDEX [104]. La brique de base de ces outils est le DSP, la mémoire ou l'ASIC. Ces outils sont assez proches de ce que l'on désire comme fonctionnalités, mais la modélisation de l'application, à cause de la décimation, est rendue trop complexe. De plus, comme on se trouve au niveau système, la différence entre latence (nombre de cycles entre l'arrivée d'une donnée et la sortie d'un résultat) et débit (nombre de cycles minimaux entre l'utilisation de

2 données successives) n'est pas faite, parce qu'elle est négligeable. Or pour notre architecture où la latence est critique, il n'est pas envisageable de perdre le moindre cycle en confondant la latence et le débit, c'est-à-dire en attendant qu'un résultat soit obtenu avant de pouvoir commencer un autre calcul comme s'il n'y avait aucun étage de pipeline.

En fait, le principal obstacle dans l'utilisation de tels outils provient de la complexité du CCP qui associe un comportement statique, sous certaines conditions, vu de l'extérieur avec un comportement dynamique en interne pour effectuer les calculs.

Aucun outil actuel ne semble donc correspondre à nos attentes (prise en compte de la complexité du comportement du CCP et de la multiplicité des applications que l'architecture doit supporter). C'est pourquoi un outil a été développé, afin de pouvoir valider rapidement le concept de l'architecture P-DCP. Cette solution permet aussi d'avoir un meilleur contrôle sur le développement de l'architecture (pas de contraintes imposées par les limitations inhérentes à ce type d'outils).

V.3 Modélisation conjointe de l'architecture et de l'application

Comme la validation de l'architecture ne peut pas utiliser les outils de synthèse d'architecture existants, il est nécessaire de réaliser un outil qui va permettre de valider notre architecture basée sur des communications statiques en ordonnançant les fonctions des applications sur les différents coprocesseurs de l'architecture. Il sera alors possible de générer le programme de contrôle qui gèrera tous les composants de l'architecture (unités de calcul, mémoires tampons, multiplexeurs de routage).

L'objectif de cette section est de présenter la modélisation utilisée pour décrire l'architecture d'une part, l'application d'autre part et comment les relier ensemble, cette modélisation devant servir ensuite de base pour effectuer l'ordonnancement des calculs. Les fichiers de descriptions mettant en pratique cette modélisation sont présentés dans l'*annexe A*.

Pour cela il est nécessaire d'extraire de l'architecture et de l'application les paramètres qui nous permettront de les décrire de manière simple, mais néanmoins suffisante, pour gérer les instants de communications.

Pour mettre en pratique les explications concernant les descriptions requises au niveau architecture et au niveau algorithme, une partie de l'algorithme *(Figure V.1)* et de l'architecture *(Figure V.2)* sera utilisée comme exemple. Il s'agit de la partie réalisant la récupération de l'horloge dans les démodulations câble ou satellite. Cet exemple contient un résumé de tous les types d'algorithmes utilisés dans la démodulation. Il constituera donc un exemple tout à fait satisfaisant. E et S modélisent les fonctions

d'entrées sorties de l'application, $I1$, $I2$ et $MIXER$ modélisent les différentes parties de l'interpolateur (filtres et mélangeur).

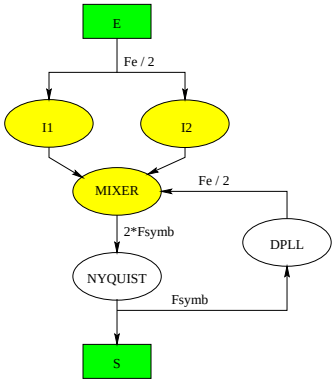


FIGURE V.1 : *Exemple d'algorithme*

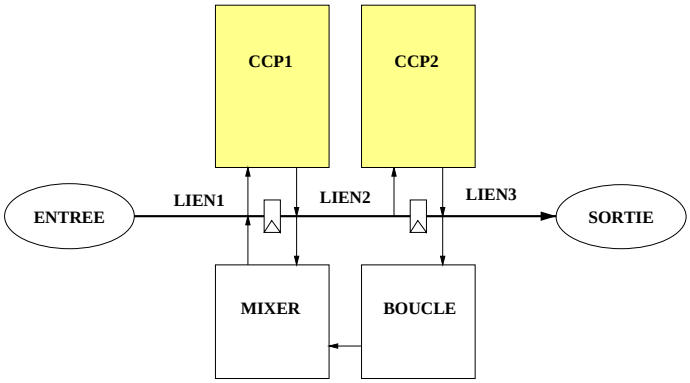


FIGURE V.2 : *Exemple d'architecture*

La *figure V.2* décrit un exemple d'architecture. Les multiplexeurs utilisés pour les communications ne sont pas dessinés mais ils se trouvent implicitement placés chaque fois qu'une connexion représentée par une flèche (les sorties des ressources) arrive en contact avec un des liens d'interconnexion (les traits épais).

V.3.1 Extraction des caractéristiques de l'architecture

Pour modéliser l'architecture, il faut commencer par décrire l'ensemble des connexions entre les composants. Notre architecture est définie par cinq types de composants : les unités de calcul, les mémoires tampons associées, les liens unidirectionnels, les registres du réseau d'interconnexion et les multiplexeurs pour le routage dans le réseau d'interconnexion. Les mémoires tampons sont implicitement toutes définies avant et après chaque unité de calcul alors que les multiplexeurs sont implicitement définis pour multiplexer l'ensemble des entrées arrivant sur un lien unidirectionnel.

En plus des connexions, les composants d'une architecture sont caractérisés par leur latence intrinsèque, c'est-à-dire le nombre d'étages de pipeline ou le nombre de cycles qu'il faudra à un calcul pour donner un résultat en sortie du composant à partir de l'instant où il a toutes les données requises.

Ainsi, l'architecture est complètement décrite par une représentation comme celle de la *figure V.3* qui se compose de boîtes (les ressources) et de flèches (les connexions unidirectionnelles entre les ressources). La latence de chaque ressource est indiquée entre parenthèses à côté du nom de la ressource. Les multiplexeurs, implicitement définis dès que l'entrée d'un composant reçoit des données venant de différentes sources, sont indiqués ici par des petits cercles.

Une telle description ne différencie pas les ressources utilisées (ressource de calcul comme les CCP ou ressource de communications comme les liens). C'est l'assignation

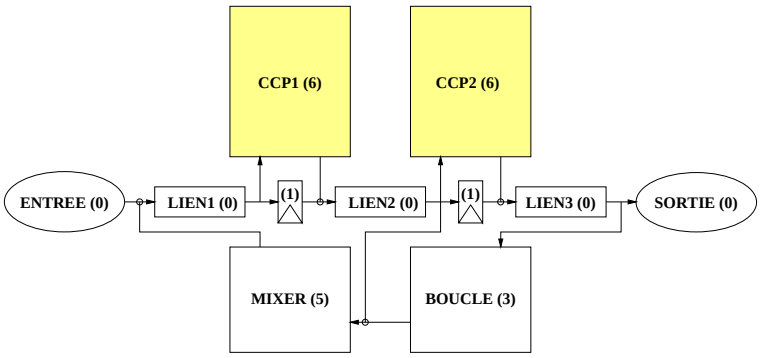


FIGURE V.3 : *Modélisation des composants de l'architecture*

des ressources de calcul à l'exécution d'une fonction qui différenciera ces deux types de ressources.

V.3.2 Extraction des caractéristiques de l'application

Comme pour la description de l'architecture, la description de l'application utilise des connexions pour relier les différents algorithmes. Par contre, il faut également définir la fréquence de fonctionnement de chaque algorithme ainsi que les données qui déclencheront le début des calculs dans un composant précis.

Dans le cas général simple, un algorithme va recevoir une donnée à une certaine fréquence et calculer grâce à cette donnée un résultat à la même fréquence. Mais il est aussi possible qu'une décimation se produise, qu'aucune donnée ne soit nécessaire ou que plusieurs données soient nécessaires.

V.3.2.a Lancement des calculs et décimation

Prenons l'exemple du mélangeur de l'interpolateur ($MIXER$) *(Figure V.1)*. Celui-ci reçoit des données venant de 2 filtres ($I1$ et $I2$) et d'une boucle de contre-réaction ($DPLL$).

Toutes ces fonctions doivent envoyer des données au $MIXER$ à la même fréquence $\frac{F_e}{2}$. Mais il faut pouvoir déterminer laquelle de ces données arrive toujours en dernier puisque c'est elle qui va commander l'exécution des calculs et donc la fréquence des calculs de la fonction $MIXER$. Il faut donc pouvoir décrire, d'une part les connexions entre fonctions qui n'ont pour rôle que d'apporter une certaine information à la fonction à réaliser, et d'autre part, les connexions qui ont en plus pour rôle de donner le signal de lancement des calculs.

L'algorithme peut ainsi être représenté par la *figure V.4* qui se compose de boîtes (les fonctions) et de flèches (les connexions entre les fonctions). Certaines flèches sont

associées à une valeur V. Cela indique qu'il faut attendre que V données aient transité par le lien représenté par la flèche avant de lancer un nouveau calcul. Parmi toutes les flèches qui arrivent sur une boîte, au plus une peut être associée à un nombre.

Il apparaît que c'est le résultat de l'un des filtres de l'interpolateur ($I1$) qui va commander l'exécution du mélangeur de l'interpolateur. Les autres flèches arrivant au $MIXER$ servent à indiquer que les résultats de $I2$ et $DPLL$ doivent être transmis à la fonction $MIXER$, même s'ils n'ont aucune incidence sur le déclenchement des calculs.

La flèche arrivant à la boîte $NYQUIST$ est associée à la valeur 2, ce qui indique que 2 données doivent être reçues par la fonction $NYQUIST$ pour commencer un nouveau calcul. Cela signifie qu'une décimation par 2 est effectuée dans cette fonction.

La description des connexions entre les algorithmes avec les indications de dépendance des exécutions et les décimations mises en jeu permettent de décrire correctement le comportement extérieur de la plupart des algorithmes des applications étudiées. Seules quelques fonctions ont un comportement indépendant du flot de données entrant. Ces fonctions peuvent être assimilées à des générateurs de données. C'est le cas des fonctions qui ne reçoivent pas de données d'autres fonctions (E) ou des fonctions pour lesquelles aucune flèche pointant sur elles ne possède de valeur associée ($DPLL$).

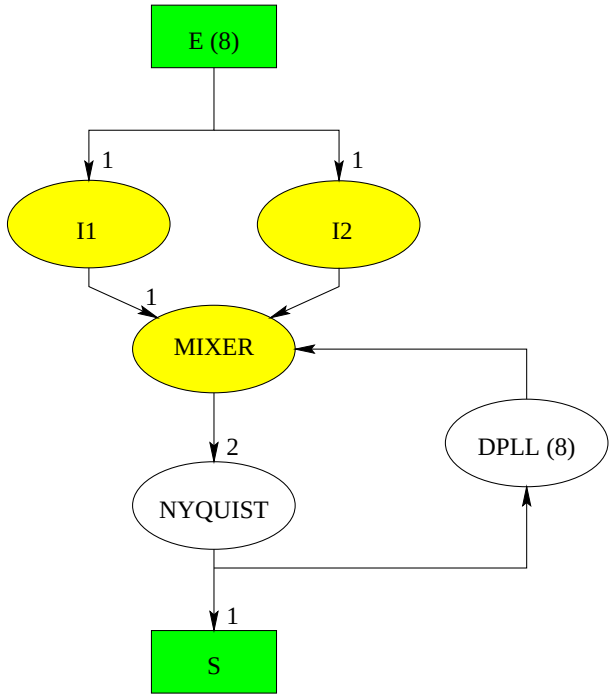


FIGURE V.4 : *Modélisation des fonctions de l'application*

V.3.2.b Générateur

Il s'agit de la fonction qui modélise l'entrée des données ainsi que des NCO qui génèrent un signal à une fréquence donnée, indépendamment de l'arrivée ou non de nouvelles données.

Pour de telles fonctions, il est donc nécessaire de donner directement la fréquence des calculs. C'est la valeur qui est indiquée entre parenthèses à l'intérieur des boîtes concernées dans la *figure V.4*. Cette valeur est donnée en nombre de cycles de travail.

Une fois que cette description est réalisée et que l'on a d'un côté la description de l'architecture, de l'autre la description de l'application, il faut relier les deux en indiquant quelles fonctions sont réalisées sur quels composants. Cela s'appelle l'assignation.

V.3.3 Assignation des composants

Il s'agit de relier les ressources et les fonctions. Toutefois, il est à noter qu'une fonction ne se réalise pas toujours de la même manière sur deux ressources conçues différemment. Il est possible d'avoir autant d'opérateurs que d'opérations à effectuer dans une fonction pour pouvoir recevoir une donnée à chaque cycle. Mais, il est également possible d'avoir quelques opérateurs qui vont être utilisés plusieurs fois consécutivement (k fois par exemple) pour réaliser la fonction. Dans ce cas, la ressource ne pourra pas accepter de recevoir une donnée à chaque cycle, mais tous les k cycles. Le débit maximal possible des données en entrée dépend donc de k.

Pour décrire l'assignation, on peut donc reprendre la *figure V.1* en indiquant pour chaque fonction (mot en gras dans la *figure V.5*), la ressource sur laquelle elle sera exécutée (mot entre parenthèses et en italique) ainsi que le débit maximal possible des données en entrée en nombre de cycles de travail (nombre entre parenthèses et en gras).

Une fois que l'architecture est décrite, de même que l'application et son assignation sur l'architecture, il est possible de vérifier qu'il y a bien adéquation entre l'architecture et l'algorithme, c'est-à-dire que les possibilités de l'architecture n'ont pas été surestimées. Pour cela, on va réaliser l'ordonnancement des calculs pour savoir précisément ce que fait chaque composant à chaque instant.

V.4 Phase 1 : Ordonnancement des calculs et des communications

L'objectif de cette section est de présenter la méthode utilisée pour réaliser l'ordonnancement des calculs et des communications sur l'architecture cible. Il existe un nom-

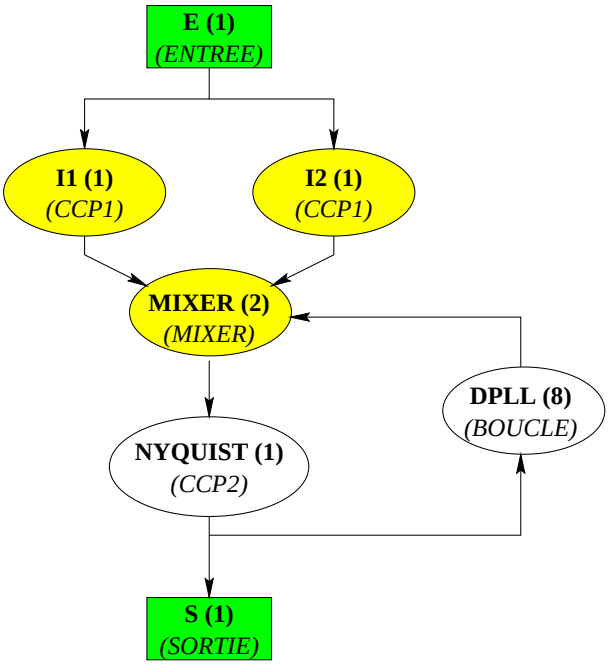


FIGURE V.5 : *Assignation des composants*

bre important de techniques d'ordonnancement : l'ordonnancement au plus tôt (ASAP : As Soon As Possible), au plus tard ((ALAP : As Last As Possible), l'ordonnancement par liste ou par poids... [97, 105]. Nous avons choisi d'utiliser la technique au plus tôt pour sa simplicité de mise en œuvre.

Le principe du programme d'ordonnancement est assez simple. Il consiste à essayer d'organiser les transferts de données entre les composants en propageant les réservations d'instants de communications en tenant compte chaque fois des latences. Cela entraîne des conflits qui sont résolus en retardant les instants de communication de certaines données par l'utilisation de mémoires tampons situées après chaque composant. En tenant compte de ces modifications, une nouvelle itération du processus d'ordonnancement est effectuée jusqu'à ce qu'aucun conflit ne subsiste *(Figure V.6)*. Il se peut que le programme ne puisse pas obtenir une solution sans conflit parce que l'architecture est sous-dimensionnée si bien qu'une ressource est surchargée. Dans ce cas, il faut changer l'architecture pour en augmenter les capacités.

À chaque cycle de la simulation, on cherche à savoir quelles sont les fonctions f qui sont prêtes à commencer des calculs ($CALCUL\ de\ f$?).

Pour chacune de ces fonctions, une réservation temporelle des ressources qui devront être utilisées par leurs résultats est effectuée *(section V.4.1)*.

Lorsque qu'une fonction cherche à réserver une ressource déjà utilisée pour réaliser une autre action, il y a conflit. Pour résoudre de tels conflits, certaines communications sont retardées en conservant les résultats dans les mémoires tampons situées en sortie de chaque unité de calcul *(section V.4.2)*. Il est alors nécessaire de recommencer

l'ordonnancement de l'ensemble des communications en intégrant ces nouveaux retards dans les tableaux décrivant l'architecture. Cette solution est très gourmande en temps de simulation pour obtenir le résultat. Il faudrait utiliser des heuristiques pour éviter cette boucle extérieure qui retourne sur l'initialisation.

Une fois que l'ensemble de la simulation a pu se réaliser complètement sans qu'il y ait eu de conflit, l'ensemble précis des instants de communication et d'exécution des calculs est disponible. Cela permet, d'une part de vérifier que l'architecture est suffisamment dimensionnée et d'autre part de générer le programme de contrôle de l'architecture ($PROGRAMME$).

La *figure V.6* donne une représentation de cet algorithme. Trois sortes de boîtes

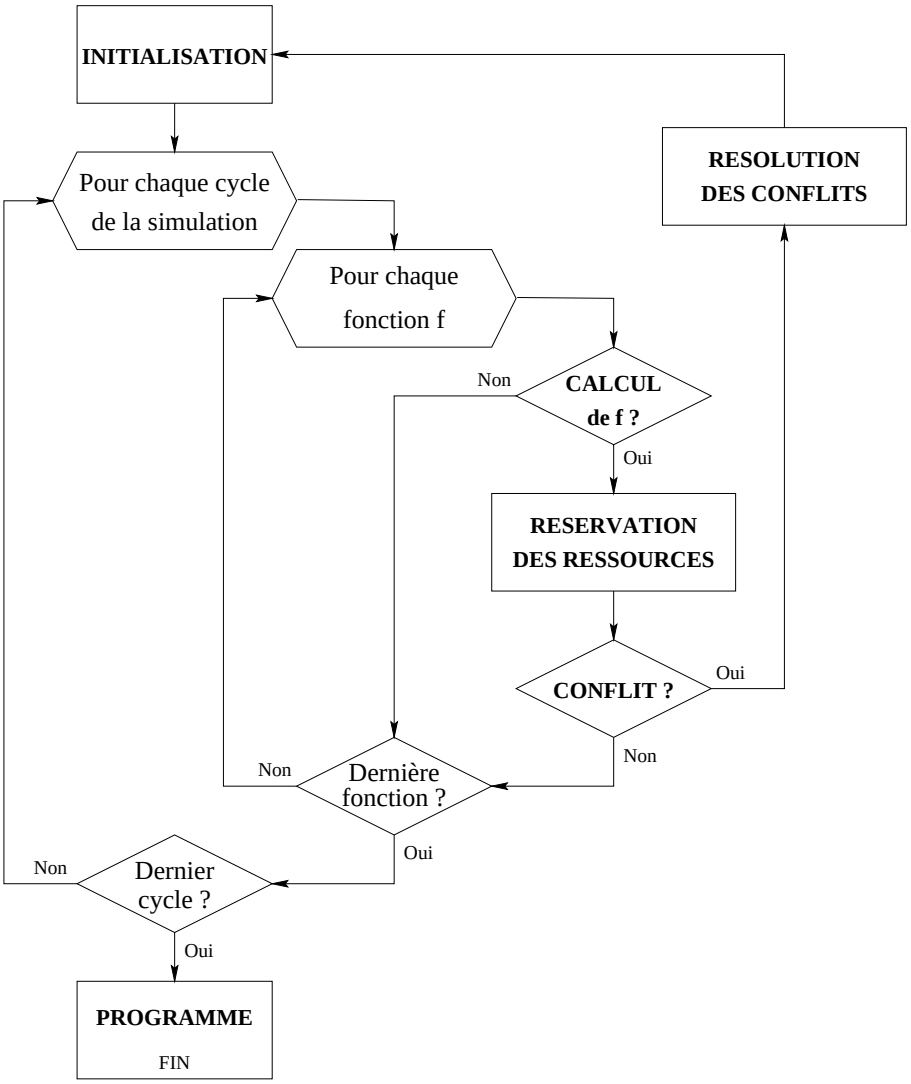


FIGURE V.6 : *Algorithme du programme d'ordonnancement*

sont représentées sur le schéma : les boîtes hexagonales qui représentent l'utilisation de boucles itératives de calcul, les losanges qui correspondent à des tests et signalent la

fin des itérations dans une boucle, et les rectangles qui réalisent les traitements.

Si le programme ne s'arrête pas en raison de conflits qui n'arrivent pas à être résolus, il faut étudier la suite des conflits pour voir comment modifier l'architecture afin de faire disparaître les conflits.

V.4.1 Réservation des ressources

La réservation des ressources consiste à prendre une option sur l'utilisation, dans le futur, des ressources de communications qui serviront à transmettre les résultats de f vers l'ensemble des ressources de calcul qui ont besoin de ces résultats. Ces réservations sont effectuées en fonction de la latence des calculs de la fonction f dans la ressource exécutant cette fonction,

Pour cela, il suffit d'utiliser un tableau à deux dimensions. Une dimension correspond au temps et l'autre dimension aux nombres de ressources de l'architecture. Ainsi, il est possible de réserver une case du tableau pour indiquer qu'il faudra utiliser une ressource à un instant donné. Si la case du tableau n'est pas réservée, elle le devient. Si elle est déjà réservée pour réaliser une autre action, il y a conflit.

Il faut noter que la case peut déjà être réservée sans qu'il y ait conflit. Cela est le cas lorsqu'une ressource de communications (ressource n'ayant aucune fonction à réaliser) est réservée deux fois pour la transmission de la même donnée vers 2 fonctions qui, elles, sont différentes.

V.4.2 Résolution des conflits

Il existe plusieurs types de conflits que l'on peut séparer mais qui peuvent toujours être résolus en utilisant les mémoires tampons implicitement définies avant et après chaque unité de calcul. En effet, ces mémoires tampons résolvent les conflits en retardant d'un cycle les données qui sont à la source des conflits :

1. Une ressource reçoit deux données différentes provenant de deux ressources différentes.

2. Une ressource de calcul reçoit deux fois la même donnée au même moment mais pour réaliser deux fonctions différentes.

3. Une ressource de calcul reçoit une donnée alors qu'elle est dans l'incapacité de l'utiliser (si plusieurs itérations étaient nécessaires sur une donnée reçue juste auparavant).

Dans le premier cas, il y a un conflit au niveau des communications, plusieurs données ayant la même destination étant émises en même temps. Pour remédier à cela, la seule solution est de retarder l'émission d'une des données. Ceci peut être réalisé, soit en retardant l'exécution du calcul générant cette donnée par l'utilisation du tampon d'entrée, soit en retardant uniquement l'émission du résultat par l'utilisation du tampon de sortie.

Dans les deux derniers cas par contre, il n'y a pas de conflit au niveau des communications puisque c'est au niveau de la ressource de calcul que les données se retrouvent bloquées. Une solution est d'utiliser la mémoire tampon à l'entrée de la ressource de calcul pour mémoriser la donnée qui ne pourra pas être utilisée. Une autre solution est de conserver la donnée dans le tampon de sortie du composant précédent jusqu'à ce que la ressource destinataire soit libre.

Finalement, dans tous les cas, une résolution n'utilisant que les tampons à l'entrée ou que les tampons à la sortie est possible. Ainsi, afin de réduire le nombre de mémoires tampons à prévoir dans le système final, il est possible de se contenter d'utiliser une seule catégorie de mémoires tampons. C'est l'utilisation des mémoires tampons en sortie des ressources de calcul qui a été mise en œuvre dans le programme d'ordonnancement.

De toute façon, il faut trouver un moyen de résoudre les conflits. Pour cela, il serait possible de choisir aléatoirement la donnée à retarder ou bien de parcourir l'ensemble des possibilités afin d'obtenir la meilleure latence pour le circuit ou la meilleure latence pour certaines boucles de contre-réaction. Dans un souci de compromis entre facilité de décision et complexité de l'algorithme, il a été décidé d'aider la décision en donnant des priorités aux fonctions.

Ainsi, lorsqu'il y a un conflit de ressources, on retarde l'émission de la donnée venant de la fonction la moins prioritaire. Si les 2 fonctions ont la même priorité, autant retarder la donnée qui a provoqué le conflit (celle qui a voulu réserver une donnée déjà réservée). L'avantage de cette solution est que si les résultats obtenus par l'ordonnancement ne paraissent pas bons (trop de délais ajoutés), il est possible de changer les priorités des fonctions pour essayer d'améliorer cet ordonnancement. De plus, de cette manière, il est possible de transmettre dans une certaine mesure la connaissance implicite de l'application et des fonctions critiques au simulateur.

V.4.3 Évolution possible de la notion de conflit

Pour l'instant, la notion de conflit s'applique à chaque ressource indépendamment des autres et sans aucune contrainte, ce qui ne donne pas beaucoup de contrôle sur certaines parties de l'architecture comme les mémoires tampons et la taille du programme.

En effet, aucune limite n'est prévue pour la taille des mémoires tampons. Or, une fois que le circuit sera réalisé, cette taille sera fixée et les simulations devront en tenir compte. Il pourrait même s'agir d'un paramètre propre à chaque ressource. Il serait

donc intéressant d'intégrer dès à présent cette contrainte dans le programme chargé d'effectuer l'ordonnancement et de transformer l'algorithme gérant les conflits pour que, lors d'un conflit, la donnée retardée soit celle dont la mémoire tampon est la moins utilisée.

De même, la taille programme n'a aucune contrainte. Il est prévu que chaque ressource puisse être employée (recevoir une instruction) à chaque cycle de la simulation. Mais ce n'est jamais le cas. Les CCP, en particulier, sont étudiés pour réaliser des calculs sur plusieurs cycles de manière autonome, et le réseau d'interconnexion est dimensionné pour réduire au minimum les latences dues aux communications. Toutes ces considérations font que, en pratique, seules quelques ressources reçoivent des instructions à chaque cycle. Cela pourrait être mis à profit pour réduire la taille programme en ne permettant qu'un nombre limité d'instructions à chaque cycle, un peu comme dans les processeurs où seulement une partie des ressources peut être utilisée simultanément. Bien entendu, il faut veiller à ne pas trop dégrader les performances de l'architecture.

Une fois l'ordonnancement effectué, la validation de l'architecture pour les applications considérées est pratiquement achevée. Il reste juste à vérifier que l'application a été correctement modélisée sans oublier aucune fonction. Pour cela, il suffit de simuler le comportement exact de l'application sur l'architecture en fournissant un signal d'entrée à démoduler et en utilisant une description comportementale de l'ensemble des blocs de l'architecture. Un tel simulateur est décrit maintenant.

V.5 Phase 2 : Simulation comportementale

Ce simulateur a pour objet de réaliser une chaîne de démodulation en utilisant comme plateforme l'architecture P-DCP. Pour cela, il est nécessaire de décrire le comportement fonctionnel de chaque bloc de l'architecture, de définir une interface pour les communications entre ces blocs en utilisant les informations générées par le programme, décrit précédemment, qui ordonnance justement les communications et les calculs.

V.5.1 Présentation du fonctionnement général du programme de validation

Ce programme effectue une validation des algorithmes utilisés avec une simulation au cycle près en ce qui concerne les transferts de données. Pour cela, après une phase d'initialisation qui configure l'ensemble des ressources de l'architecture, le cœur de l'algorithme s'effectue en trois étapes :

– La première étape consiste à récupérer les résultats que sont prêts à fournir les

différents blocs de l'architecture et à les stocker dans un tableau qui va simuler la latence des communications dans le réseau d'interconnexion.

- Ensuite, en fonction des informations fournies par le programme d'ordonnancement, les blocs qui peuvent recevoir une donnée la reçoivent.
- Finalement, les blocs exécutent les calculs qu'ils doivent faire.

La *figure V.7* donne une représentation de cet algorithme.

Pour que cet algorithme fonctionne, il faut que la latence des blocs ne soit pas nulle, ce qui n'est pas le cas si l'on considère les liens de communications comme des blocs. Ainsi, dans un souci de simplicité, les liens de communications, qui ne réalisent d'ailleurs aucune transformation sur les données, ne sont pas pris en compte dans cette simulation. Ceci ne pose pas de problème pour la validation puisque l'ordonnancement a déjà permis de vérifier cette partie de l'architecture.

V.5.2 Récupération des résultats

Il s'agit de voir, à chaque cycle, les unités de calcul qui ont des résultats à transmettre sur le réseau d'interconnexion après avoir pris en compte la latence du bloc et le délai supplémentaire éventuel requis pour éviter les conflits. On récupère ici tous les résultats, sans se préoccuper des informations apportées par l'ordonnancement des communications. Le test pour vérifier qu'il n'y a pas eu d'erreur à ce niveau s'effectue plus tard.

V.5.3 Mémorisation des résultats

Il ne s'agit pas d'une mémorisation au niveau matériel des résultats, mais plutôt d'un tableau de variables locales contenant les résultats en transit sur le réseau d'interconnexion. Ce tableau peut conserver les résultats pendant plusieurs cycles pour prendre en compte les registres placés dans le réseau.

V.5.4 Transfert des données

À partir des informations générées par le calcul de l'ordonnancement, il est possible de connaître les instants exacts où les différents blocs doivent recevoir une donnée. En fonction de ces informations, les données stockées dans le tableau de résultats sont transférées vers les différents blocs concernés.

Lorsque tous les transferts possibles ont été réalisés, un test est fait sur les éléments du tableau de résultats pour vérifier s'il reste des données qui n'auraient pas dû se

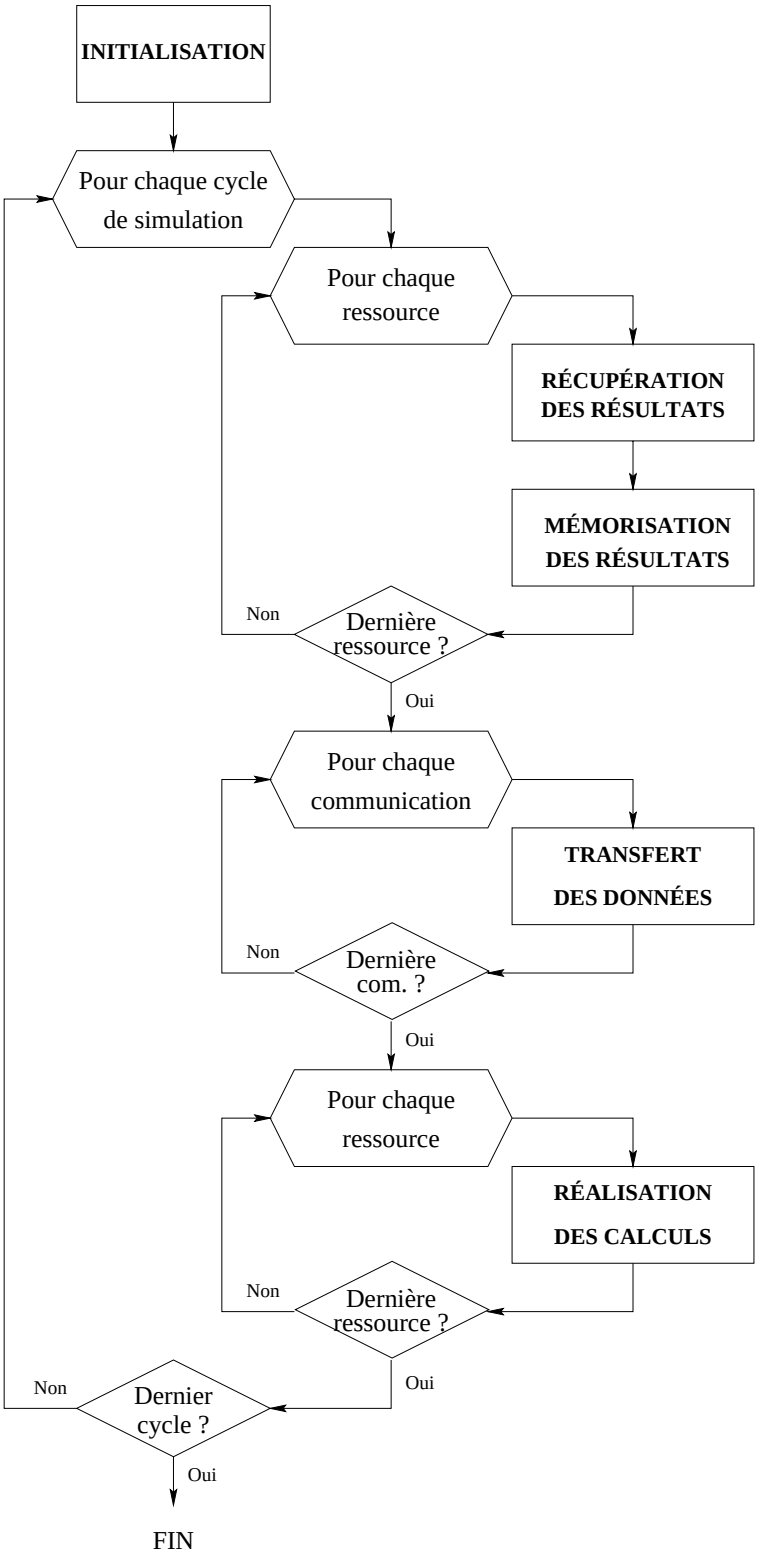


FIGURE V.7 : *Algorithme du programme de simulation comportemental*

trouver là. Ce test permet donc de vérifier la cohérence (latence, lancement des cal-

culs, décimation) entre les descriptions littérales de l'architecture et de l'application en tant que boîtes noires avec l'implémentation qui en est réalisée. Cette étape de transfert est en particulier importante pour vérifier la bonne intégration du programme de simulation du "CCP" avec le reste de l'architecture.

V.5.5 Réalisation des calculs

La réalisation des calculs est la partie spécifique à chaque ressource de calcul qui va simuler le comportement de celle-ci. Il ne s'agit évidemment pas ici d'obtenir une description comportementale cycle à cycle de l'intérieur de chaque bloc, mais d'obtenir un comportement cycle à cycle de l'interface de chaque bloc. Ainsi, pour une séquence de données reçues, il faut que le résultat soit correct et renvoyé à l'extérieur au bon moment.

Pour correspondre de la manière la plus fidèle possible aux descriptions de l'architecture et de l'application utilisées pour prévoir l'ordonnancement des calculs, cette étape de la simulation est subdivisée en 3 sous-étapes :

La première sous-étape consiste à décider si un calcul doit être lancé. Cette décision peut être obtenue de façon automatique grâce aux informations données dans la description de l'application (fonctions génératrices, provenance des données qui commandent le lancement des calculs, décimation).

Si la première sous-étape indique qu'un calcul doit être lancé, ce calcul est exécuté de manière instantanée en utilisant l'algorithme approprié.

Ensuite, les latences correspondant au calcul et au délai dans les tampons mémoires sont ajoutées. Ainsi, le résultat attendu sera disponible à la sortie du bloc au bon moment.

En utilisant les outils qui viennent d'être présenté, on obtient quelques résultats qui servent à programmer et à valider l'architecture.

V.6 Résultats

Nous présentons ici les résultats obtenus en utilisant les outils définis dans ce chapitre pour réaliser l'ordonnancement des calculs et des communications et pour simuler le comportement de l'architecture. Ces résultats ne concernent que la démodulation câble et la démodulation satellite et utilisent l'architecture correspondant à la *figure IV.12*.

V.6.1 Phase 1 : Ordonnancement des calculs et des communications

L'utilisation du programme d'ordonnancement va permettre d'obtenir les informations sur la latence des boucles de contre-réaction, la taille des mémoires tampons et leur utilisation ainsi que sur la taille des programmes et leur utilisation.

Ce programme requiert une description de l'application, de l'architecture et de l'assignation des fonctions de l'application sur les ressources de l'architecture.

Pour le câble, il y a 6 modes de fonctionnement possibles, chacun correspondant à une plage de fréquences symboles et à une structure de filtre anti-repliement particulière. Nous allons donc tester ces 6 configurations possibles qui correspondent à 6 applications, 6 assignations mais une seule architecture. En fait, l'assignation des différentes fonctions de filtrage (les seules fonctions différentes à assigner en fait) est toujours la même et représentée par la *figure V.8*.

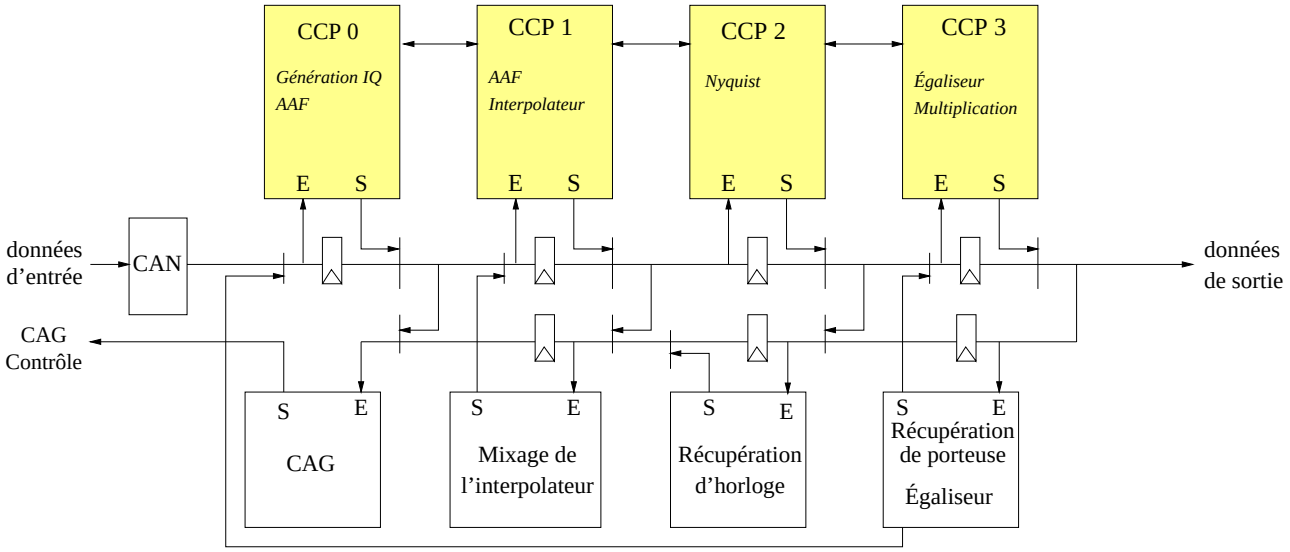


FIGURE V.8 : *Assignation des fonctions pour la démodulation câble*

Dans cette figure, le terme *Interpolateur* dans le CCP 1 fait référence aux filtres de l'interpolateur, qui seront réalisés dans ce coprocesseur. Le filtre anti-repliement *AAF* étant composé de plusieurs filtres successifs, une partie de ces filtres est réalisée par le CCP 0, une autre par le CCP 1. L'égaliseur, comme l'interpolateur, est en partie exécuté sur le CCP 3 et en partie sur un coprocesseur complémentaire, comme expliqué à la *section IV.3.1.d*.

Pour le satellite, la puissance de calcul requise par les filtres interpolateurs est plus importante que pour le câble, si bien qu'ils sont en partie réalisés sur le CCP 2, ce qui décale également le calcul du filtre de Nyquist *(Figure V.9)*.

Les descriptions nécessaires pour réaliser l'ordonnancement étant faites, il est main-

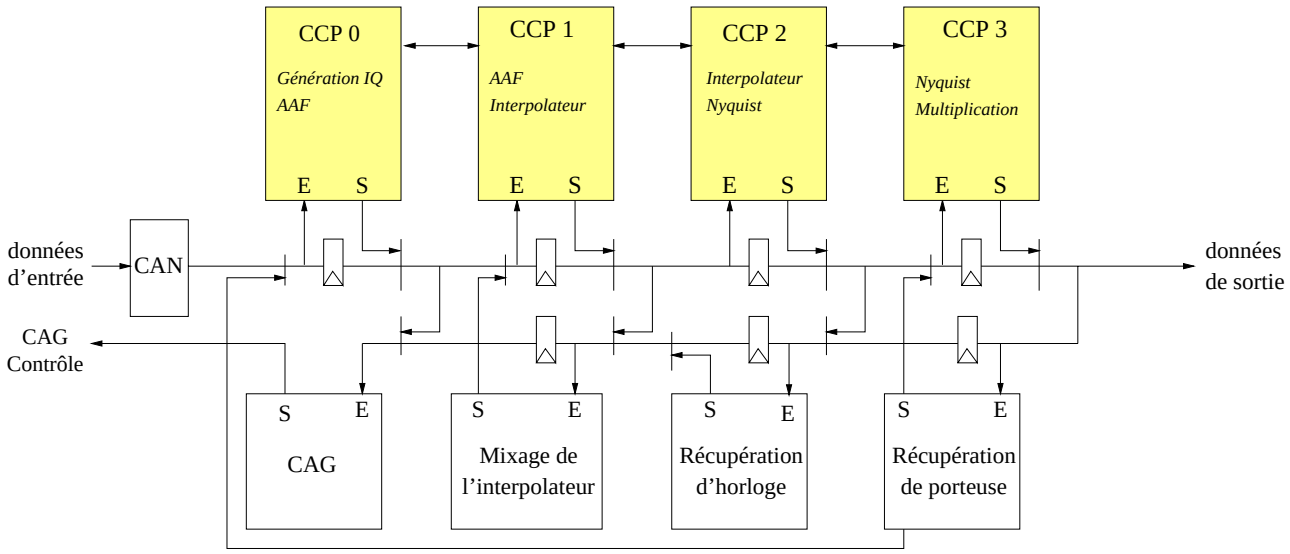


FIGURE V.9 : *Assignation des fonctions pour la démodulation satellite*

tenant possible de voir les résultats obtenus pour les différentes plages de fréquences symboles.

V.6.1.a Résolution des conflits

Les conflits qui doivent être résolus sont peu nombreux (3 ou 4 au maximum). En fait, comme l'architecture du réseau d'interconnexion est conçue pour correspondre le plus possible à la circulation des données entre les algorithmes, cela est tout à fait normal.

Pour le câble, deux types de conflits apparaissent.

Il y a d'abord un conflit entre les deux filtres interpolateurs qui voudraient être réalisés en même temps sur le CCP 1. Comme cela n'est pas possible, l'un des deux est retardé ce qui ne cause aucun délai supplémentaire dans les calculs ultérieurs. En effet, en regardant le schéma de l'interpolateur *(Figure I.6 page 11)*, on s'aperçoit qu'il doit en effet y avoir un retard entre leur exécution.

Ensuite, il y a des conflits pour accéder au CCP 3. Cela est dû au nombre de données différentes que ce bloc doit recevoir pour effectuer ses calculs (deux données pour la multiplication et pour chacun des filtres de l'égaliseur). Un réordonnancement des transferts allant vers ce bloc est donc nécessaire.

Ce type de conflit apparaît aussi dans la boucle de récupération de l'horloge du satellite lorsque le débit des données est important. En fait pour les boucles de contre-réaction, les conflits sont dus au fait que les générateurs (les oscillateurs des boucles) génèrent des données sans s'occuper de leur voisinage. Il suffit donc de les paramétrer pour qu'ils génèrent leur données au bon moment sans risquer de nouveaux conflits.

Par conséquent, un simple registre peut suffire pour servir de mémoire tampon puisque les conflits sont rares si bien qu'il n'y a jamais plusieurs données qui se retrouvent bloquées à la sortie d'un même coprocesseur.

V.6.1.b Latence des boucles

La faible latence des calculs et des communications était une contrainte requise pour la synchronisation des démodulations à l'aide des boucles de contre-réaction. Il est donc nécessaire d'étudier les résultats concernant les latences de ces boucles, à savoir la récupération de l'horloge et la récupération de la porteuse. Cette dernière réagit sur 2 parties de l'application. La réaction sur la génération IQ est une boucle lente alors que la réaction sur la multiplication complexe est une boucle rapide.

Le *tableau V.1* présente les latences en fonction des plages de fréquences symboles. Pour chaque plage, 2 configurations de filtre anti-repliement sont utilisées en fonction de la valeur exacte de la fréquence symbole. Les nombres entre parenthèses indiquent le nombre de cycles pour que la réaction correspondant à l'arrivée d'une donnée à l'entrée de la boucle soit effective. Les autres nombres correspondent aux nombres de symboles qui ont été traitées pendant cette période là. Ainsi le chiffre 1 dans la dernière colonne de la dernière ligne indique que la réaction arrive avant qu'une nouvelle donnée arrive, ce qui est parfait.

Plage de fréquences symboles	Récupération de l'horloge	Récupération de porteuse lente	Récupération de porteuse rapide
[7 MHz; 3,5 MHz[	2 (26) 2 (26)	5 (71) 5 (75)	3 (34) 3 (44)
[3,5 MHz; 1,75 MHz[	1 (28) 2 (38)	3 (95) 4 (99)	2 (52) 2 (46)
[1,75 MHz; 0,87 MHz[	1 (38) 1 (34)	2 (127) 3 (135)	1 (62) 1 (57)

TABLEAU V.1: *Latence des boucles de contre-réaction dans le câble*

L'intérêt de la solution programmable avec une fréquence de travail importante apparaît pour les fréquences symboles basses puisque l'on arrive alors à avoir certaines contre-réactions pratiquement instantanées. Par contre la latence de la boucle lente augmente avec la réduction de la fréquence symbole. La raison en est la décimation qui ralentit de plus en plus la boucle alors que les échantillons arrivent toujours à la même fréquence. Ce n'est donc pas l'architecture qui peut être remise en cause pour ces latences importantes.

Pour le satellite, les performances des boucles sont moins bonnes *(Tableau V.2)*. Cela s'explique pour plusieurs raisons. D'abord, la fréquence d'échantillonnage et les fréquences symboles sont plus importantes ce qui laisse moins de cycles entre l'arrivée

de deux données. De plus, le parallélisme de sous-mots est utilisé, ce qui fait que les données arrivent 2 par 2.

Plage de fréquences symboles	Récupération de l'horloge	Récupération de porteuse lente	Récupération de porteuse rapide
[32 MHz; 24 MHz[	12 (22)	26 (51)	8 (29)
[24 MHz; 16 MHz[	14 (25)	28 (55)	8 (29)
[16 MHz; 8 MHz[	8 (28)	34 (68)	4 (28)
[8 MHz; 4 MHz[	6 (38)	46 (92)	4 (45)

TABLEAU V.2: *Latence des boucles de contre-réaction dans le satellite*

L'ordonnancement nous a donc permis d'obtenir quelques informations sur les performances de l'architecture, mais pour savoir si les latences des boucles de contre-réaction sont correctes, il faut simuler une communication numérique en utilisant notre architecture grâce au simulateur décrit dans ce chapitre.

V.6.2 Phase 2 : Simulation comportementale

Dans cette section, il s'agit d'intégrer les différents algorithmes des applications de manière à simuler une transmission effective de données pour vérifier que l'application décrite réalise effectivement la démodulation souhaitée. Cela permettra de vérifier que les descriptions de l'architecture et de l'application qui ont servi pour réaliser l'ordonnancement étaient correctes et que l'architecture est elle aussi bien conçue.

L'intégration des algorithmes se fait en plusieurs étapes qui permettent de tester les différentes parties de l'algorithme séparément.

La première étape consiste à contruire la chaîne directe de démodulation (sans les boucles) et à n'ajouter aucun bruit dans le canal de transmission. Cette étape permet de valider l'implantation des différents NCO, de la transposition en bande de base (génération IQ), de l'interpolateur ainsi que les filtres. On doit obtenir une constellation de points bien distincts.

La deuxième étape consiste simplement à ajouter du bruit dans le canal de transmission pour vérifier que les filtres effectuent bien leur rôle de réducteur de bruits. En fonction de la puissance du bruit, l'aspect de la constellation obtenue change, les points symbolisant le symbole devenant des nuages de points plus ou moins larges.

La dernière étape consiste à ajouter les boucles de contre-réaction et à introduire une petite erreur dans les fréquences utilisées pour voir si le système arrive à se synchroniser.

Une fois que l'application complète est programmée, il est possible de tester ses performances par rapport aux autres démodulateurs, c'est-à-dire les conditions dans lesquelles elle arrive à démoduler un signal ainsi que la qualité du résultat (le taux d'erreurs par bit).

V.7 Conclusion

Un ensemble d'outils nécessaires à la validation de l'architecture P-DCP a été présenté, à commencer par un outil qui permet d'ordonnancer les calculs et les communications de l'architecture afin de profiter du caractère statique et périodique des communications. Cet outil permet de valider la partie communication de l'architecture, en précisant l'ensemble des communications qui s'effectuent à chaque cycle d'horloge.

Ensuite, un simulateur comportemental modulaire de l'architecture a été mis en place. Il doit permettre de vérifier que l'architecture P-DCP définie est en effet capable de réaliser des démodulations.

Pour l'instant, l'ordonnancement des démodulations câbles et satellite a été réalisé, et la simulation de ces mêmes démodulations est en bonne voie.

Il reste à ajouter des fonctionnalités aux outils pour pouvoir valider la démodulation terrestre qui utilise deux éléments complexes qui ne sont pas requis pour l'instant, à savoir une mémoire et la gestion de cette mémoire ainsi qu'un processeur général.

Dans le but d'une implantation matérielle de l'architecture, il faudrait aussi prévoir de pouvoir générer, à partir des informations fournies par l'ordonnancement des calculs et des communications, le programme des applications directement utilisable par cette implantation.

CONCLUSION ET PERSPECTIVES

L'immense marché des récepteurs de télévision numérique associé aux possibilités toujours croissantes des circuits intégrés en technologie CMOS permet de s'intéresser maintenant à la conception de démodulateurs de télévision numérique multi-standard. En effet, jusqu'à présent, les puissances de calcul requises en démodulation (10 Gops) associé aux différents algorithmes correspondant à chaque type de transmission (câble, satellite...) imposait la conception d'un ASIC par standard, ce qui ne représente pas une solution satisfaisante (marché morcelé, coûts de développement multipliés...)

On s'est intéressé ici à concevoir une architecture programmable, appelée P-DCP pour "Parallel Digital Communication Processor", adéquate pour un tel démodulateur. À partir de l'analyse des algorithmes de démodulation, il a été possible de déterminer les caractéristiques fondamentales de cette architecture qui s'appuie sur :

- l'utilisation de blocs de calcul qui réalisent les fonctions critiques et en particulier les filtres afin d'obtenir de la flexibilité tout en conservant un rendement suffisant pour offrir une alternative fiable face aux circuits dédiés.
- L'utilisation du parallélisme de sous-mots pour profiter au maximum du chemin de données disponible pour réaliser des applications ayant des besoins considérablement différents en matière de précision sur les calculs.
- Un réseau d'interconnexion statique correspondant aux contraintes de la démodulation numérique (faible latence, haut débit, localité des communications) et intégrant un mécanisme permettant de gérer les communications dynamiques requises par la fréquence variable des symboles.

À cette occasion un coprocesseur de filtrage efficace et original, appelé CCP pour "Communication Co-Processor" a été conçu. Il est capable de réaliser un ensemble de filtres avec une faible latence grâce à la technique des calculs anticipés ajoutée à une

commutation de contexte sans perte de cycle. Il met d'ailleurs en œuvre des mécanismes intéressants (unité de permutation entre la mémoire et la partie opérative) qui pourraient être avantageusement utilisé dans la conception des prochains processeurs de traitements du signal à architecture parrallèle.

Afin de valider cette architecture P-DCP, des outils d'aide à la programmation ont été développés. Il permettent, à partir d'une modélisation de l'architecture et de l'application, de déterminer si l'architecture est suffisamment puissante pour effectuer les calculs ou si des modifications doivent y être apportées. Une description comportementale des différents composants de l'architecture permet alors de simuler l'exécution de l'application sur l'architecture pour vérifier l'ensemble du système. Les simulations menées sur l'architecture ont prouvé son efficacité pour les démodulations câble et satellite. La nécessité de développer de tels outils montre qu'il reste encore quelques lacunes dans les outils de synthèse d'architecture qui ne peuvent pas encore prendre en compte de manière optimale des composants au comportement complexe. Un travail important sur la modélisation comportementale de composants complexes est encore à faire. Pour utiliser les outils de synthèse d'architecture, il aurait fallu choisir cette voie dès le début ce qui aurait amené à une solution totalement différente (meilleure ?), conditionnée par l'outil.

La conception matérielle de notre architecture pour les démodulations monoporteuses (câble et satellite) peut commencer. Pour la démodulation terrestre, il reste à ajouter les composants requis (mémoire et cœur de processeur) et à les valider avant de passer à leur réalisation matérielle. Alors, il sera possible de juger sérieusement l'efficacité de l'architecture (coût, qualité de la démodulation).

Quelques améliorations sont envisageables sur les outils développés pour leur ajouter la génération automatique d'un programme lorsque l'architecture sera effectivement réalisée. Ceci facilitera grandement l'utilisation du circuit pour mettre en œuvre de nouvelles applications.

Cette architecture a été spécialement étudiée afin d'être utilisée pour la démodulation de la télévision européenne (standard DVB). Toutefois, il a toujours été conservé à l'esprit que l'objectif à plus long terme était de tendre vers un démodulateur universel pour la télévision numérique en intégrant les standards américains et japonais. C'est pourquoi, l'architecture est assez générique, composée de briques de bases qui peuvent être assemblées à volonté selon les besoins, voire être utilisées dans d'autres architectures sous forme d'IP (Intellectual Property). C'est aussi pour cela que les outils de validation utilisent une modélisation de l'architecture. Il suffit de changer cette modélisation pour être à même de valider une autre architecture, ce qui est très important puisque l'étude algorithmique est basée sur un nombre trop restreint d'applications pour supposer que l'architecture obtenue puisse supporter l'ensemble des standards.

Durant toute l'étude, les choix d'architectures furent dictés par les contraintes des algorithmes pré-existants (et indépendants) qu'il fallait réaliser. Un axe de recherche prometteur consisterait à effectuer le choix des algorithmes et de l'architecture conjoin-

tement, de façon à :

- restreindre le choix des algorithmes pour les différentes applications en vue de simplifier le matériel (surtout pour les boucles de contre-réactions),
- une fois le matériel fixé, repenser les algorithmes afin de profiter au maximum de la puissance de calcul disponible et améliorer ainsi les performances (taille des filtres).

Fort de l'expérience acquise lors de la conception de cette architecture de démodulateur de télévision numérique multi-standard, il devient possible de réfléchir à l'intégration de l'interactivité sur une telle architecture, puisque après l'essor de la télévision numérique par simple diffusion, nous assistons à l'essor de la télévision numérique interactive. Pour cela, il faudra étudier la réalisation d'un modulateur numérique universel. Tout ceci ouvre la porte à la conception d'un modem numérique universel supportant les autres standards de communications hauts débits (xDSL (Digital Subscriber Line) pour les différentes versions du système utilisé sur le réseau téléphonique, la transmission sur une ligne électrique...). Ce modem pourrait aussi faire le lien entre le réseau numérique domestique (In Home Digital Network) et le reste du réseau mondial.

Bibliographie

[1] Semiconductor Industry Association. International technology roadmap for semiconductor, 1998. http://www.semichips.org.

[2] H. Benoit. *La Télévision numérique: MPEG-1, MPEG-2 et les principes du système européen DVB*. Dunod, 1996.

[3] L. Rysdale, P. De Bot, and S. Hulyakar. Digital Video Broadcasting: Satellite specification. *Philips J. Res.*, 50(1):91–104, 1996.

[4] M. Ghosh. Digital Video Broadcasting: Cable specification. *Philips J. Res.*, 50(1):79–90, 1996.

[5] P. De Bot and F. Daffara. Digital Terrestrial Television Broadcasting. *Philips J. Res.*, 50(1):61–77, 1996.

[6] O. Gay-Bellile and E. Dujardin. A programmable OFDM demodulator architecture for digital terrestrial television. Technical Report C 98-676, LEP, January 1998.

[7] A.A.M.L. Bruekers and A.W.M. v. d. Enden. Variable Phase Delay (VPD) filters. Technical Report 015/96, PHILIPS, 1996.

[8] J. G. Proakis. *Digital Communications*. Mc Graw-Hill, third edition, 1995.

[9] W. C. Lindsey. A Survey of Digital Phase-Locked Loops. *Proceedings of the IEEE*, 69(4):410–431, April 1981.

[10] F. M. Gardner. A BPSK/QPSK Timing-Error Detector for Sampled Receivers. *IEEE Trans. Communications*, COM-34(5):423–429, May 1986.

[11] J. M. Guillaud and G. Martinez. Algorithms for a 256-QAM cable demodulator. Technical Report C 94 584, LEP, December 1994.

[12] A. J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Trans. Information Theory*, IT-13:260–269, April 1967.

[13] M. Purser. *Introduction to Error-Correcting Codes*. Artech House, 1995.

[14] T. K. Truong, M.-T. Shih, I. S. Reed, and E. H. Satorius. A VLSI Design for a Trace-Back Viterbi Decoder. *IEEE Trans. Communications*, 40(3):616–624, March 1992.

[15] P. J. Black and T. H. Meng. A 140 Mb/s, 32-state, radix-4 Viterbi decoder. *IEEE J. Solid-State Circuits*, pages 1877–1885, December 1992.

[16] G. Fettweis and H. Meyr. High-Speed Viterbi processor: A systolic array solution. *IEEE J. Selected Areas in Communications*, pages 1520–1534, October 1990.

[17] E. Boutillon. *Architecture et implantation VLSI de techniques de modulations codées performantes adaptées au canal de Rayleigh*. PhD thesis, ENST, December 1995.

[18] G. Cohen, J.-L. Dornstetter, and P. Godlewski. *Codes correcteurs d'erreurs : une introduction au codage algébrique*. CNET-ENST. Masson, 1992.

[19] O. Gay-Bellile. Reed-Solomon Decoder. Technical Report 9509a, ENST, January 1995.

[20] Y. R. Shayan and T. Le-Ngoc. A cellular time-domain Reed-Solomon decoder. In *IEEE International Conference on Commmunication Systems*, pages 20.4.1–5, 1990.

[21] H. M. Shao, T. K. Truong, L. J. Deutsch, J. H. Yuen, and I. S. Reed. A VLSI Design of a Pipeline Reed-Solomon Decoder. *IEEE Trans. Computers*, 34(5):393–403, May 1985.

[22] R. E. Blahut. A Universal Reed-Solomon Decoder. *IBM J. Res. Develop.*, 28(2):150–158, March 1984.

[23] R. B. Lee. Subword Parallelism with MAX-2. *IEEE Micro*, pages 51–59, August 1996.

[24] B. Wong and H. Samueli. A 200-MHz All-Digital QAM Modulator and Demodulator in 1.2-μm CMOS for Digital Radio Applications. *IEEE J. Solid-State Circuits*, 26(12):1970–1979, December 1991.

[25] E. De Man, M. Schulz, R. Schmidmaier, M. Schöbinger, and T. Noll. Architecture and Circuit Design of a 6-GOPS Signal Processor for QAM Demodulator Applications. *IEEE J. Solid-State Circuits*, 30(3):219–227, March 1995.

[26] N. Benvenuto, L. Franks, and F. Hill Jr. On the Design of FIR Filters with Powers-of-Two Coefficients. *IEEE Trans. Communications*, COM-32(12):1299–1307, December 1984.

[27] H. Samueli. An Improved Search Algorithm for the Design of Multiplierless FIR Filters with Powers-of-Two Coefficients. *IEEE Trans. Circuits and Syst.*, 36(7):1044–1047, July 1989.

[28] H. Samueli, T.-J. Lin, R. Hawley, and S. Olafson. VLSI Architectures for a High-Speed tunable digital Modulator/Demodulator/Bandpass-Filter chip set. In *Proc. ISCAS*, pages 1065–1068, 1992.

[29] R. van der Wal and Montreuil Leo. QPSK and BPSK demodulator chip-set for satellite applications. *IEEE Trans. Consumer Electronics*, 41(1):30–41, February 1995.

[30] F. Lane, C. Scarpa, J. Koslov, S. Vinekar, and Y. Juan. A Single Chip Demodulator for 64/256 QAM. *IEEE Trans. Consumer Electronics*, 42(4):1003–1009, November 1996.

[31] P. McGoldrick. Demodulator with FEC decoder has on-chip timing recovery. *Electronic design*, pages 55–64, September 1996.

[32] R. Joshi and H. Samueli. A 100 MHz, 5 Mbaud QAM Decision-Feedback Equalizer for Digital Television Applications. In *ISSCC Dig. Tech. Papers*, pages 68–69, February 1994.

[33] F. Lu and H. Samueli. A 60-Mbaud Adaptive Transversal Equalizer in 1.0-μm CMOS for QAM Digital Modems. In *Proc. CICC*, pages 16.6.1–16.6.4, 1993.

[34] R. Makowitz, A. Buttar, S. Anikhindi, J. Gledhill, and C. Patzelt. DVB-T decoder ICs. *IEEE Trans. Consumer Electronics*, 43(3):438–442, August 1997.

[35] A. Deelaruelle, J. Huisken, J. van Loon, and F. Welten. A chip set for a Digital Audio Broadcasting Channel Decoder. In *Proc. CICC*, pages 293–296, 1995.

[36] J. P. Sweeney. Leverage Superscalar DSPs in Digital Communications Systems. *Electronic design*, pages 49–62, November 1998.

[37] L. Geppert. High-flying DSP architectures. *IEEE SPECTRUM*, pages 53–56, November 1998.

[38] J. Stevens. DSPs in communications. *IEEE SPECTRUM*, pages 39–46, September 1998.

[39] Texas Instruments. *TMS320C62xx Technical Brief*, January 1997.

[40] OREN Semiconductor. OR51220 DTV US Multi-Standard Demodulator with FEC, 1998. http://www.oren.com/.

[41] W. Gehrke, R. Hoffer, and P. Piersch. A Hierarchical Multiprocessor Architecture based on Heterogeneous Processors for Video Coding Applications. In *Proc. ICASSP*, volume 2, pages 413–416, 1994.

[42] T. Araki, M. Toyokura, T. Akiyama, H. Takeno, B. Wilson, and K. Aono. Video DSP Architecture for MPEG2 CODEC. In *Proc. ICASSP*, volume 2, pages 417–420, 1994.

[43] H. Trezeguet. Le PC multimédia ne peut ignorer les média-processeurs. *Electronique*, pages 36–39, April 1997.

[44] C. Hansen. MicroUnity's MediaProcessor Architecture. *IEEE Micro*, pages 34–41, August 1996.

[45] E. Grayver and B. Daneshrad. Reconfigurable signal processing ASIC architecture for high speed data communications. In *Proc. ISCAS*, pages IV.389–392. UCLA, June 1998.

[46] O. Chen, W. Liu, H. Hsieh, and J.-Y. Wang. A highly-scaleable FIR using the radix-4 Booth algorithm. In *Proc. ICASSP*, pages 1765–1768. Taiwan, May 1998.

[47] M. Hatamian and S. K. Rao. A 100 MHz 40-Tap programmable FIR filter chip. In *Proc. ISCAS*, pages 3053–3056, New Orleans, USA, May 1990. AT&T Bell Laboratories.

[48] W. Kamp, R. Künemund, H. Söldner, and R. Hofer. Programmable 2D Linear Filter for Video Applications. *IEEE J. Solid-State Circuits*, 25(3):735–740, June 1990.

[49] P. A. Ruetz. The Architectures and Design of a 20 MHz Real-Time DSP Chip Set. *IEEE J. Solid-State Circuits*, 24(2):338–348, April 1989.

[50] C. C. Stearns, D. A. Luthi, P. A. Ruetz, and P. H. Ang. A reconfigurable 64-Tap transversal filter. In *Proc. CICC*, pages 8.8.1–8.8.4. LSI LOGIC, 1988.

[51] D. Reuver and H. Klar. A Configurable Convolution Chip with Programmable Coefficients. *IEEE J. Solid-State Circuits*, 27(7):1121–1123, July 1992.

[52] W. Haberecht, E. De Man, and M. Schulz. A programmable 32-Tap Digital Interpolation Filter in 1.5μm CMOS with 80 MHz Output Data Rate. In *Proc. CICC*, pages 13.1.1–13.1.4, Boston, USA, May 1990. Siemens.

[53] H. M. Chang and M. Sunwoo. An efficient programmable 2-D convolver chip. In *Proc. ISCAS*, pages II.429–432, June 1998.

[54] T. G. Noll and S. Meier. A 40 MHz Programmable Semi-Systolic Transversal Filter. In *ISSCC Dig. Tech. Papers*, pages 180–181, February 1987.

[55] T. G. Noll. Semi-Systolic maximum rate transversal filters with programmable coefficients. In *Systolic Arrays*, pages 103–112. W. Moore, 1987.

[56] R. Hartley, P. Corbett, P. Jacob, and S. Karr. A High-Speed FIR Filter designed by Compiler. In *Proc. CICC*, pages 20.2.1–20.2.4, 1989.

[57] G. Casagrande, A. Chiari, C. Golla, and S. Micelli. VLSI Programmable Digital Filter for Video Signal Processing. *Journal of VLSI Signal Processing*, 6(3):219–231, December 1993.

[58] C. Golla, F. Nava, F. Cavallotti, A. Cremonesi, P. Piacentini, and G. Casagrande. A 30 MSamples/s Programmable Filter Processor. In *ISSCC Dig. Tech. Papers*, pages 116–117. SGS-Thomson, February 1990.

[59] F. Airoldi, F. Cavallotti, A. Cremonesi, and G. G. Rizzotto. General purpose video processor for High-Speed filtering applications. *IEEE Trans. Consumer Electronics*, 36(3):334–338, August 1990.

[60] C. Golla, F. Nava, F. Cavallotti, A. Cremonesi, and G. Casagrande. 30 MSamples/s Programmable Filter Processor. *IEEE J. Solid-State Circuits*, 25(6):1502–1509, December 1990.

[61] E. A. Vopni, M. I. Elmasry, and J. V. Hanson. A CMOS Time-multiplexed digital filter for vocoder applications. In *Canadian Communications and Power Conference*, pages 457–460, 1980.

[62] P. Bednarz, S. Lin, C. Modlin, and J. Cioffi. Design, Performance, and Extensions of the RAM-DFE Architecture. *IEEE Trans. Magnetics*, 31(2):1196–1201, March 1995.

[63] J. B. Evans, Y. C. Lim, and B. Liu. A High-Speed programmable digital FIR filter. In *Proc. ICASSP*, volume 2, pages 969–971, Albuquerque, USA, April 1990. IEEE Signal Processing Society.

[64] K.-Y. Khoo, A Kwentus, and A. N. Willson, Jr. A programmable FIR Digital Filter using CSD coefficients. *IEEE J. Solid-State Circuits*, 31(6):869–874, June 1996.

[65] W. J. Oh and Y. H. Lee. Implementation of Programmable Multiplierless FIR Filters with Powers-of-Two Coefficients. *IEEE Trans. Circuits and Syst. II*, 42(8):553–556, August 1995.

[66] K.-Y. Khoo, A Kwentus, and A. N. Willson, Jr. An Efficient 175MHz Programmable FIR Digital Filter. In *Proc. ISCAS*, pages 72–75, May 1993.

[67] H. Leich and J. Hancq. Implantation de filtres numériques sur composants logiques programmables. In *GRETSI*, pages 1001–1004, Juan-Les-Pins, September 1995.

[68] G. R. Goslin. Implement DSP functions in FPGAs to reduce cost and boost performance. *EDN*, pages 155–164, October 1996.

[69] T. C. Denk, C. J. Nicol, P. Larsson, and K. Azadet. Reconfigurable hardware for efficient implementation of Programmable FIR Filters. In *Proc. ICASSP*, pages 3005–3008. Lucent Technologies, May 1998.

[70] C. J. Nicol, P. Larsson, K. Azadet, and J. H. O'Neill. A Low-Power 128-Tap Digital Adaptive Equalizer for Broadband Modems. *IEEE J. Solid-State Circuits*, 32(11):1777–1789, November 1997.

[71] ATMEL. *CBIC Programmable FIR Filter AT76C001*, 1996.

[72] A. Y. Kwentus, M. J. Werter, and A. N. Willson, Jr. A Programmable Digital Filter IC Employing Multiple Processors on a Single Chip. *IEEE Trans. Circuits and Syst. for Video Technology*, 2(2):231–244, June 1992.

[73] M. M. Cai, D. A. Luthi, P. A. Ruetz, and P. H. Ang. A 40 MHz programmable and reconfigurable filter processor. In *Proc. CICC*, pages 13.2.1–13.2.4, Boston, USA, May 1990. LSI LOGIC.

[74] D. E. Borth, I. A. Gerson, J. R. Haug, and C. D. Thompson. A Flexible Adaptive FIR Filter VLSI IC. *IEEE J. Selected Areas in Communications*, 6:494–503, April 1988.

[75] ATMEL. *Programmable FIR Filter AT76C002*, 1996.

[76] O. Gay-Bellile and E. Dujardin. Architecture of a Programmable Filter Co-Processor suitable for Digital Communications. Technical Report C 97-662, LEP, August 1997.

[77] O. Gay-Bellile and E. Dujardin. Architecture of a programmable FIR filter co-processor. In *Proc. ISCAS*, pages V.433–436, Monterey - USA, June 1998.

[78] E. Dujardin and O. Gay-Bellile. Software implementation of ADSL application with a convolution co-processor. In *Proc. ICASSP*, pages 3053–3056, Seattle - USA, May 1998.

[79] E. Dujardin and O. Gay-Bellile. Circuit programmable pour réaliser un filtre numérique, December 1997. Brevet 97 16306.

[80] I. Kuroda and T. Nishitani. Multimedia Processors. *Proceedings of the IEEE*, 86(6):1203–1221, June 1998.

[81] P. Quinton and Y. Robert. *Algorithmes et architectures systoliques.* Masson, 1989.

[82] L. R. Rabiner and B. Gold. *Theory and application of Digital Signal Processing.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[83] E. Dujardin and O. Gay-Bellile. Circuit processeur programmable muni d'une mémoire reconfigurable, pour réaliser un filtre numérique, March 1998. Brevet 98 02910.

[84] A. Capitanio, N. Dutt, and A. Nicolau. Architectural Tradeoff Analysis of Partitioned VLIWs. Technical Report 14, University of California, 1994.

[85] E. Dujardin and O. Gay-Bellile. Récepteur, circuit programmable et procédé de calcul de filtres numériques, March 1999. Brevet 99 02456.

[86] J. Zorko. The Communication Co Processor: Architecture definition and validation. Technical Report S 98 988, LEP, September 1998.

[87] X. Marchal. Cost estimation of Communication Co-Processor. Technical report, LEP, May 1999. To be published.

[88] P. Pirsch, N. Demassieux, and W. Gehrke. VLSI Architectures for Video Compression - A Survey. *Proceedings of the IEEE*, 83(2):220–246, February 1995.

[89] ACTS/DVBIRD/WP1. First Receiver Architecture - ICs specification, May 1996. Deliverable D01 - First Draft.

[90] O. Gay-Bellile and E. Dujardin. Système de transmission, récepteur et réseau d'interconnexion, May 1999. Brevet 99 06020.

[91] E. Dujardin and O. Gay-Bellile. Processeur ayant une fréquence de fonctionnement adaptable, May 1999. Brevet 99 06019.

[92] O. Gay-Bellile and E. Dujardin. Système de transmission, récepteur et dispositif de décimation, May 1999. Brevet 99 06021.

[93] E. Dujardin and O. Gay-Bellile. Démodulateur numérique programmable pour modulations ofdm, May 1999. Brevet 99 06018.

[94] A. Parker, M. Mc Farland, and R. Camposano. Tutorial on high level synthesis. In *Design Automation Conference (DAC)*, 1988.

[95] *A full synthesys session combining AMICAL and VHDL simulation and synthesis*, ?

[96] P. Gutberlet, J. Müller, H. Krämer, and W. Rosenstiel. Automatic Module Allocation in High Level Synthesis. In *Proc. European Design Automation Conference (EURO-DAC)*, pages 328–333, 1992.

[97] E. Martin, O. Sentieys, and J. L. Philippe. Synthèse architecturale de cœur de processeurs de traitement du signal. *revue TSI*, 1994.

[98] C.-M. Chu, M. Potkonjak, M. Thaler, and J. Rabaey. HYPER: An Interactive Synthesis Environment for High Performance Real Time Applications. In *IEEE International Conference on Computer Design*, pages 432–435, November 1989.

[99] C.-Y. Wang and K. K. Parhi. MARS: A High-Level DSP Synthesis Tool Integrated Within the Mentor Graphics Environment. In *Mentor Graphics User's Group Conference*, October 1995.

[100] J. Huisken and F. Weltenm. FADIC: Architectural Synthesis applied in IC Design. In *Design Automation Conference (DAC)*, pages 579–584, 1996.

[101] P. Marwedel, B. Landwehr, and R. Dömer. Built-in Chaining: Introducing Complex Components into Architectural Synthesis. Technical Report 611, University of Dortmund, Computer Science XII, April 1996.

[102] Y.-K. Kwok, I. Ahmad, M.-Y. Wu, and W. Shu. A Graphical Tool for Automatic Parallelization and Scheduling of Programs on Multiprocessors. In *EURO-PAR*, pages 294–301, Passau - Germany, 1997.

[103] P. Lippens, J. van Meerbergen, A. van der Werf, W. Verhaegh, B. McSweeney, J. Huisken, and O. McArdle. PHIDEO: A Silicon Compiler for High Speed Algorithms. In *Proc. European Conference on Design Automation (EDAC)*, pages 436–441, February 1991.

[104] Y. Sorel. Real-time embedded image processing applications using the A^3 methodology. In *IEEE International Conference on Image Processing*, pages II.145–148, November 1996.

[105] M. Mc Farland, A. Parker, and R. Camposano. The High-Level Synthesis of Digital Systems. *Proceedings of the IEEE*, 78(2), February 1990.

Annexe A

Exemple de modélisation conjointe d'architecture et d'application

Cette annexe présente les fichiers de description correspondant à l'application et à l'architecture prises en exemple dans la *section V.3*.

A.1 Description de l'architecture

Le *tableau A.1* présente le fichier de description de l'architecture modélisée par la *figure V.3 page 96*.

Ressource source	Ressource destination	Latence
ENTREE	LIEN1	0
LIEN1	CCP1	0
LIEN1	LIEN2	1
CCP1	LIEN2	6
LIEN2	CCP2	0
LIEN2	MIXER	0
LIEN2	LIEN3	1
CCP2	LIEN3	6
LIEN3	BOUCLE	0
BOUCLE	MIXER	3
MIXER	LIEN1	5
LIEN3	SORTIE	0

TABLEAU A.1: *Fichier de description de l'architecture*

Chaque ligne contient 3 colonnes : la première colonne donne le nom de la ressource émettrice, la seconde donne le nom de la ressource réceptrice et la dernière, le nombre

de cycles pour qu'une donnée arrivant à l'entrée de la ressource émettrice traverse les différentes couches de pipeline la séparant de l'entrée de la ressource réceptrice.

Dans cette description, les registres n'apparaissent pas. Leur influence sur l'architecture est incorporée dans la description des liens d'interconnexions qui ont différentes latences en fonction de la ressource destination.

La description de l'architecture se résume donc à la liste des connexions directes entre deux composants ainsi que la latence pour qu'une donnée entrant dans le composant initial engendre un résultat entrant dans le composant suivant. Les seuls composants qui sont utilisés dans la liste sont les unités de calcul et les liens de communications.

A.2 Description de l'application

Le *tableau A.2* présente le fichier de description des connexions entre les fonctions de l'application modélisée par la *figure V.4 page 97*. Ce fichier correspond donc aux flèches de cette figure ainsi qu'aux nombres qui leur sont attachés.

Fonction source	Fonction destination	Decimation
E	I1	1
E	I2	1
I1	MIXER	1
I2	MIXER	0
DPLL	MIXER	0
MIXER	NYQUIST	2
NYQUIST	DPLL	0
NYQUIST	S	1

TABLEAU A.2: *Fichier de description des connexions entre les fonctions de l'application*

Chaque ligne contient 3 colonnes : la première colonne donne le nom de la fonction émettrice, la seconde donne le nom de la fonction réceptrice et la dernière, le nombre de donnée que la fonction réceptrice doit attendre de la fonction émettrice avant de commencer un nouveau calcul. Si ce nombre est nul, cela veut dire que des données seront transmises, mais qu'elles n'auront aucune répercussion sur le lancement des calculs dans la fonction réceptrice.

Le reste de l'application modélisée par la *figure V.4*, à savoir la fréquence des calculs des générateurs est présentée par le fichier de description des fonctions génératrices.

Chaque ligne contient 2 colonnes : la première colonne donne le nom de la fonction génératrice et la seconde donne le nombre de cycles de travail entre la génération de

Fonction	fréquence de génération
E	8
DPLL	8

TABLEAU A.3: *Description des fonctions génératrices*

2 données consécutives par ces fonctions. Ce nombre dépend donc du rapport entre la fréquence d'horloge et la fréquence de sortie des résultats.

La description de l'application se résume donc à deux fichiers, l'un qui décrit les connexions entre les éléments de l'application et l'autre qui donne des informations supplémentaires sur des éléments particuliers de l'application (les générateurs de données).

A.3 Description de l'assignation des composants

Le *tableau A.4* présente le fichier de description de l'assignation des fonctions de l'application, assignation modélisée par la *figure V.5 page 99*.

Fonction	Ressource	Debit
E	ENTREE	1
I1	CCP1	1
I2	CCP1	1
MIXER	MIXER	2
NYQUIST	CCP2	1
DPLL	BOUCLE	8
S	SORTIE	1

TABLEAU A.4: *Description de l'assignation des composants*

Sur chaque ligne, la première colonne contient le nom d'une fonction à exécuter tandis que la deuxième colonne contient le nom de la ressource sur laquelle cette fonction sera réalisée et la dernière colonne contient le débit maximal possible en entrée du composant en nombre de cycles. Un nombre de 1 signifie donc que le calcul est immédiat et qu'une nouvelle donnée pourra être utilisée au prochain cycle. Un nombre de 2 signifie qu'il faudra attendre au moins 2 cycles avant de pouvoir recevoir une autre donnée sur cette ressource.