

RP 10 322

**CENTRE NATIONAL D'ETUDES  
DES TELECOMMUNICATIONS**

**CENTRE NATIONAL DE LA  
RECHERCHE SCIENTIFIQUE**

**CENTRE DE  
RECHERCHES  
EN PHYSIQUE DE  
L'ENVIRONNEMENT  
TERRESTRE  
ET PLANETAIRE**

**CRPE**

**NOTE TECHNIQUE  
CRPE / 187**

**FILTRAGE RIF RAPIDE:  
ALGORITHMES ET ARCHITECTURES**

09J

oui  
INGE

(p210)

**Par  
Z.J. MOU**

**INSTITUT DE L'INFORMATION  
SCIENTIFIQUE ET TECHNIQUE**  
2, Allée du Parc de Brabois - Tél. 83.50.46.00  
INIST - 54514 VANDŒUVRE LÈS NANCY CEDEX

**RPE/ETP  
38-40, rue du Général Leclerc  
92131 ISSY-LES-MOULINEAUX, FRANCE**

G 75081

CENTRE NATIONAL D'ETUDES  
DES TELECOMMUNICATIONS

Centre Paris B

CENTRE NATIONAL DE LA  
RECHERCHE SCIENTIFIQUE

Département TOAE

**CENTRE DE RECHERCHES EN PHYSIQUE DE  
L'ENVIRONNEMENT TERRESTRE ET PLANETAIRE**

NOTE TECHNIQUE CRPE/187

**FILTRAGE RIF RAPIDE : ALGORITHMES  
ET ARCHITECTURES**

par

**Z.J. MOU**

**RPE/ETP**

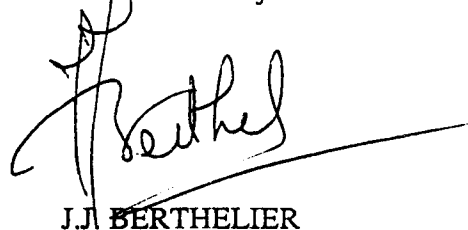
38-40 rue du Général Leclerc  
92131 ISSY-LES-MOULINEAUX

Le Directeur



G. SOMMERIA

Le Directeur Adjoint



J.J. BERTHELIER

Juin 1990

# LISTE DE DIFFUSION SYSTEMATIQUE

## CNET

MM.	POITEVIN	Directeur du CNET
	THABARD	Directeur Adjoint du CNET
	COLONNA	Adjoint Militaire au Directeur du CNET
	MERLIN	Directeur des Programmes
	BLOCH	DICET
	THUE	DICET
MME	HENAFF	DICET
MM.	PIGNAL	PAB
	RAMAT	PAB
	NOBLANC	PAB-BAG
	ABOUDARHAM	PAB-SHM
	HOCQUET	PAB-STC
	THEBAULT	PAB-STC
MME	PARIS	PAB-RPE
MM.	BAUDIN	PAB-RPE
	BERTHELIER	PAB-RPE
	BIC	PAB-RPE
	CERISIER	PAB-RPE
	GENDRIN	PAB-RPE
	LAVERGNAT	PAB-RPE
	ROBERT	PAB-RPE
	ROUX	PAB-RPE
	SOMMERIA	PAB-RPE
	TESTUD	PAB-RPE
	VIDAL-MADJAR	PAB-RPE

## CNRS

MM.	BERROIR	TOAE
	CHARPENTIER	SPI
MME	SAHAL	TOAE
MM.	COUTURIER	INSU
MME	LEFEUVRE	AD3
M.	DUVAL	AD5

## CNES

MMES	AMMAR
	DEBOUZY
MM.	BAUDOIN
	FELLOUS
	HERNANDEZ (Toulouse)

## Bibliothèques

CNET-SDI (3)  
 CNET-EDB  
 CNET-RPE (Issy) (5)  
 CNET-RPE (St Maur) (2)  
 Observatoire de Meudon  
 CNRS-SA  
 CNRS-INIST  
 CNRS-LPCE

# LISTE COMPLEMENTAIRE

LAA/TSS/CMC	COMBESURE
LAA/TSS/CMC	GILLOIRE
LAA/TSS/CMC	LAMBLIN
LAA/TSS/CMC	PETIT
LAA/SLC/BSA	HERVE
LAA/RSM/SIM	HAMIDI RIDHA
LAB/MER/STA	RENAN
LAB/SMR/TCM	JONCOUR
LAB/IFE/CIP	TREGUIER
PAB/STC/PSN	MAITRET
PAB/SHM/SMC	TORTELIER
PAB/RPE/ETP	BENESTY
PAB/RPE/ITS	CELIN

## CNS - GRENOBLE

DIR/SVP	ARNDT
AMS/CAT	WITTMANN
CCI/AMS	BALESTRO
CCI/AMS	MAGINOT
CCI/AMS	PRIVAT
CCI/AMS	ROQUIER

## CCETT - RENNES

STM/TRM	LEMAUVIEL
SRL/MNC	LESAFFRE
SRL/MTT	PALICOT
SRL/MTT	LANOISELEE
DOCUMENTATION	

## EXTERIEUR

ENST	BARRAL
ENST	GRENIER
ENST	MOU
ENST	RENARD
ENST	BIBLIOTHEQUE

## RESUME

Le filtre à réponse impulsionnelle finie (RIF) joue un rôle des plus importants dans le traitement numérique du signal et représente souvent la principale charge de calcul dans une application soit en logiciel soit en matériel.

Cette thèse est divisé en deux parties. La première partie de la thèse traite le problème de la réduction de la complexité arithmétique du filtre RIF. Nous fournissons un ensemble d'algorithmes permettant de 'casser' le filtre habituel en plusieurs sous-filtres échantillonnés, de telle manière que le nombre d'opérations à effectuer se trouve réduit. La deuxième partie étudie non seulement l'implantation de ces sous-filtres mais plus généralement l'architecture des filtres RIF en vue de leur intégration VLSI.

Nous présentons une approche unifiée pour tous les algorithmes rapides de filtrage RIF. Le théorème du reste chinois (TRC) constitue la base de l'approche. Tout d'abord nous formulons le filtrage RIF comme un produit polynomial. Ensuite l'application du TRC se fait en trois étapes: 1) interpolation; 2) filtrage; 3) reconstruction. L'approche se termine par le recouvrement. Sous une présentation pseudocyclique, il est facile de démontrer quelques propriétés utiles des algorithmes. Les algorithmes classiques sont examinés dans ce cadre. Mais l'unification de ces algorithmes n'est pas notre seul objectif. Nous présentons aussi des nouvelles possibilités apportées par cette approche, qui permet d'établir en particulier tous les algorithmes intermédiaires entre traitements temporels et fréquentiels. Nous traitons les algorithmes de petite longueur en détail. Ces algorithmes permettent de réduire la complexité arithmétique en gardant comme brique de base des filtres RIF d'ordre plus petit. Ils sont donc ouverts à diverses implantations.

Nous étudions ensuite l'aspect arithmétique du multiplieur-accumulateur d'une part, et de nouvelles architectures d'autre part. Une conception compacte et régulière de l'arbre de Wallace est proposée pour surmonter les difficultés qui empêchaient l'application de cette structure par ailleurs très efficace au niveau du temps de calcul. Nous présentons quelques nouveaux accumulateurs rapides. L'architecture du filtre RIF par l'arithmétique distribuée est analysée en détail. Nous présentons de nouvelles structures ayant les caractéristiques suivantes: sans ROM; avec l'additionneur carry-save comme brique de base; avec accumulation rapide. En particulier, un nouveau codage est proposé pour profiter de la symétrie des coefficients afin de réduire la quantité de matériel et d'accélérer le calcul.

## SUMMARY

The finite impulse response (FIR) filter plays one of the most important roles in digital signal processing. It often represents the major computational load in a system either in software or in hardware.

The dissertation is divided in two parts. The first part deals with the problem of reducing the arithmetic complexity in FIR filtering. We present a class of algorithms allowing to "cut" a usual filter into several subfilters in such a way that the number of operations is reduced. The second part studies not only the implementation of these subfilters but more generally the architecture of FIR filters for Very Large Scale Integration (VLSI).

We present a unified approach to all the fast FIR filtering algorithms. The Chinese Remainder Theorem (CRT) constitutes the basis of the approach. First of all, we formulate the FIR filtering equation as a polynomial product. Then the application of CRT is made in three steps: 1) interpolation; 2) filtering; 3) reconstruction. The approach finishes by overlapping. Under a pseudocyclic presentation, some interesting properties of algorithms are demonstrated. The classical FIR filtering algorithms are examined in the context of this approach. But the unification of all the FIR filtering algorithms is not our only objective. We present also several new possibilities promised by the approach, which in particular allows to establish all the intermediate algorithms between processings in time and in frequency domain. We explain the short-length FIR filtering algorithms in detail. These algorithms allow to reduce the arithmetic complexity while maintaining moderate-length FIR filters as building blocks. Hence they are open to various implementations.

Our study is then concerned with the arithmetic aspect in one hand and new architectures in the other hand for VLSI implementation of FIR filters. Some improved structures are presented for Wallace-tree and fast accumulators. The distributed arithmetic implementation of FIR filters is also analyzed in more detail. We propose new structures having the following characteristics: without ROM; with carry-save adder as building block; with fast accumulators. In particular a new encoding technique is suggested for implementing symmetric FIR filters.

(Les pages 1 à 132 correspondent à une rédaction détaillée en anglais des résultats obtenus. Les pages 133 à 165 en donnent une version concise en français.)

## Table of Contents

Chapter 1.....	1
Introduction .....	1
1.1 FIR filtering algorithms.....	1
1.2 FIR filter architectures.....	4
1.3 Terminology and remarks .....	5
Chapter 2.....	7
A unified approach to the fast FIR filtering algorithms.....	7
2.1 Formulation of FIR filtering as a polynomial product.....	8
2.2 Fast computation of a polynomial product.....	9
2.2.1 The Chinese Remainder Theorem .....	10
2.2.2 Interpolation.....	10
2.2.3 Filtering .....	11
2.2.4 Reconstruction by the CRT .....	11
2.3 Overlap.....	12
2.4 Pseudocyclic convolution.....	13
2.4.1 Diagonalization of a pseudocirculant matrix.....	14
2.4.2 Transposition of an $F(N,N)$ algorithm .....	14
2.4.3 Identical arithmetic complexity in direct and transposed forms .....	17
2.5 Remarks.....	18

Chapter 3.....	19
Review on the FIR filtering algorithms and new possibilities.....	19
3.1 Review on the classical algorithms .....	19
3.1.1 FFT-based algorithms .....	20
3.1.2 Algorithms using aperiodic convolutions .....	22
3.1.3 Winograd algorithms.....	25
3.2 Short length FIR filtering algorithms.....	28
3.3 Shorter FFT-based algorithms .....	29
3.4 Short length complex FIR filtering algorithms.....	33
3.5 Remarks.....	37
Chapter 4.....	41
Short length FIR filtering algorithms .....	41
4.1 Introduction.....	41
4.2 Article 1.....	42
4.3 Article 2.....	50
Chapter 5.....	79
Fast multiplier-accumulator .....	79
5.1 Multiplier-accumulator.....	79
5.2 Reduction of the number of partial products.....	79
5.2.1 Booth's encoding.....	80
5.2.2 The modified Booth's encoding (MBE) .....	81
5.3 Multioperand summation.....	83
5.3.1 Using carry-save adder (CSA).....	83

5.3.2 Wallace-tree and its compact layout .....	85
5.4 Fast accumulator.....	88
5.4.1 Carry-save (uni-operand) accumulator .....	89
5.4.2 Bi-operand and multi-operand accumulator .....	90
5.5 Pipelined multiplier-accumulator and new schemes .....	92
5.6 Remarks.....	95
Chapter 6.....	97
New results on the distributed arithmetic implementation of FIR filters .....	97
6.1 The principle of distributed arithmetic .....	98
6.2 New structures of general FIR filters .....	100
6.2.1 Fast accumulation.....	100
6.2.2 Adder-based structures .....	101
6.2.3 Using the modified Booth's encoding.....	105
6.3 Symmetric FIR filters.....	107
6.3.1 Direct implementation .....	107
6.3.2 Using a new encoding .....	108
6.3.3 Pipelining.....	111
6.4 Bit-parallel implementation .....	112
6.4.1 Bit-slice approach.....	112
6.4.2 Symmetric FIR filters .....	112
6.5 Remarks.....	113
Chapter 7.....	115



Conclusion and perspectives.....	115
7.1 Conclusion.....	115
7.2 Perspectives .....	116
7.2.1 Multiprocessor implementation of FIR filters .....	116
7.2.2 Application of fast FIR filtering algorithms to LMS adaptive filters.....	116
7.2.3 Silicon compiler for FIR filter implementation.....	117
Appendix.....	119
Fixed-point error analysis of short-length FIR filtering algorithms.....	119
1. Major assumptions.....	119
2. Quantization error in direct FIR filter computation.....	121
3. Illustrative example for error analysis in short-length FIR filtering algorithms .....	121
3.1 Analysis for direct radix-2 algorithm.....	121
3.2 Analysis for transposed radix-2 algorithm.....	122
3.3 Consideration for reduction of errors.....	123
3.4 Error analysis for quantization of coefficients .....	124
4. Conclusion .....	125
References.....	127
Version concise en français.....	133

# Chapter 1

## Introduction

The finite impulse response (FIR) filter plays one of the most important roles in modern digital signal processing. It exhibits some nice properties:

- 1) allowing linear phase implementation so that there is no phase distortion after filtering;
- 2) stability;
- 3) arbitrary approximation to any frequency response;
- 4) good quantization performance.

These properties make the FIR filter widely used in a large number of applications, for example, the telecommunications. However the main disadvantage of the FIR filter is the requirement of a great number of arithmetic operations in both software and hardware implementations.

The first part of this thesis is concerned with the problem of reducing the arithmetic complexity of FIR filters. The main issue is to construct and implement fast algorithms.

Since digital signal shows better performance than analog signal in diverse aspects such as precision, sensitivity to environment, conservability, etc., many traditional analog devices are being replaced by digital ones. However digital devices run in general more slowly than their analog counterparts. As real time signal processing requires higher and higher speed, another question arises: how to design a high speed FIR filter? A study on this problem constitutes the second part of the thesis.

### 1.1 FIR filtering algorithms

In the mid-60 the rediscovery of fast Fourier transform (FFT) algorithm by Cooley and Tukey [Coo65] has changed the world of digital signal processing. It is soon realized that FIR filters can be computed efficiently through FFT [Sto66]. Stockham's algorithm seems to be the earliest fast FIR filtering algorithm. This algorithm still enjoys a great popularity, if not the greatest. Agarwal and Burrus [Aga74] proposed some fast aperiodic

convolution algorithms for FIR filtering. Their algorithms are characterized by direct yet fast computation instead of transformation. Winograd [Win80] also presented an original class of FIR filtering algorithms without transformation. At the same time he gave the lower bound of the arithmetic complexity of computing an FIR filter.

The main feature in these algorithms is to compute  $N$  consecutive outputs together so that redundancy between  $N$  computations can be removed. It is tacitly assumed that  $N$  is not smaller than the filter's length. These algorithms were derived and tended to be optimized under such an assumption that a multiplication costs much more time than an addition.

The progress in semiconductor and computer technologies has altered the criteria for a fast algorithm. Multiplication time is no longer dominant in modern computers. For example in the widely used TMS320 Digital Signal Processor series, a multiplication plus an accumulation requires the same time as an addition alone does. However the classical FIR filtering algorithms are not suitable for implementation on these processors, because of their complex structure. The need for algorithms not only computationally efficient but also structurally implementable emerges.

This is our motivation to present a new class of FIR filtering algorithms [Mou87]. These algorithms are also proposed independently by other authors for implementation using filter banks [Vet88] or in VLSI [Kwa87]. They present a number of good properties such as arithmetic operation reduction and regular structure, particularly suitable for real time signal processing.

All these algorithms appear quite different from each other. Nevertheless they inherently belong to the same class of algorithms. In Chapter 2, we present a unified approach to the fast FIR filtering algorithms. The Chinese Remainder Theorem (CRT), which is well known for deriving fast polynomial product algorithms, constitutes the basis for that approach.

First of all, we show that the FIR filtering can be formulated as an polynomial product. Then the application of the CRT takes place in three steps: 1) interpolation; 2) filtering (multiplication in wide sense); 3) reconstruction. Finally overlapping is performed to get the filter's output. Overlap in FIR filtering algorithms differs them from other convolution algorithms. By taking into account the overlap, we may further reduce the number of operations. The FIR filtering algorithms thus constructed are in the form of a

pseudocyclic convolution: a pseudocirculant matrix [Vai88] multiplied by a vector. All the fast algorithms can be seen as a diagonalization of the pseudocirculant matrix. Under the pseudocyclic convolution, we can prove easily certain useful properties of the algorithms such as transposition and identical complexity in both direct and transposed form.

In Chapter 3, the classical algorithms of Stockham, Agarwal&Burrus, and Winograd will be reviewed in the light of the approach. Their derivations, relations to each other will be addressed, leading to a better understanding of the advantages as well as the limits of these algorithms. However the unification of the algorithms is not our unique objective. We will show some new possibilities promised by the approach, which in particular allows to establish all the intermediate algorithms between processing in frequency and in time. Among them, three types of algorithms are of special interest: short-length FIR filtering algorithms (SLFIR), shorter-FFT based algorithms (SFFTA) and short length complex FIR filtering algorithms (SLCFIR). The first one reduces the arithmetic complexity by maintaining the multiply-accumulate structure, which is suitable for many implementations. SFFTA allows to establish a number of algorithms meeting various trade-offs between arithmetic complexity, structural regularity and system delay. SLCFIR algorithms are the complex counterpart of SLFIR, suitable for fast complex FIR filtering. The combination of SFFTA and SLCFIR will result in some algorithms requiring less operations than the classical FFT-based ones, a fact going to the opposite of the general belief that the latter is the most efficient for large block processing.

Chapter 4 deals with the short length FIR filtering algorithms (SLFIR) in a detailed manner, based on two articles of the author (joint with P.Duhamel) [Mou87,Mou89]. These algorithms compute together a few number of filter outputs, allowing reducing the arithmetic complexity while maintaining smaller FIR filters as computing kernels. Hence, they are open to a wide range of implementations, including Digital Signal Processors (DSP's), VLSI, and even general purpose computers. The inherent parallelism of the algorithms allows also single-instruction-multiple-data (SIMD) multi-processor implementation. General rules are provided for combining several SLFIR algorithms to construct composite length FIR filtering algorithms. Their arithmetic complexities are evaluated and compared to FFT based algorithms under various criteria.

## 1.2 FIR filter architectures

The VLSI technology has greatly boosted the implementation of the digital devices. As emphasized by the title of the thesis, our device of interest is the FIR filter. There are various ways of implementing an FIR filter. For low speed applications, such as speech processing, we use mostly Digital Signal Processors where the principal working horse is a multiplier-accumulator. For high speed applications, such as telecommunications, we should fully integrate an FIR filter either in bit serial or in bit parallel methodology. The architecture is very important to an IC implementation. It predominates the speed and design complexity which are essential to a realization. Moreover in the first part the new filtering algorithms require FIR filters as building blocks. Thus we consecrate the second part to the study of some architectures for an FIR filter.

Chapter 5 is devoted to the review of arithmetic aspects of multiplier accumulator design and the investigation of new schemes. This study also serves as preparation for the subsequent chapters. We will review the reduction of partial products using Booth's encoding [Boo51] and the modified Booth's encoding [Mac61], summing of multiple operands using carry-save array and Wallace tree [Wal64], and fast accumulators. A compact yet regular Wallace tree design is proposed to overcome a belief that Wallace tree is not practical or not implementable [Vui81, Mon88]. Some new accumulators will be presented which allow faster implementation than ever known architectures to date.

Chapter 6 addresses the distributed arithmetic implementation of digital filters. It is often characterized by a lookup table or a ROM [Cro73, Bur77]. As the technology evolves, the ROM becomes the bottleneck to speed and occupies large chip area. We will present some new structures having following features: without ROM, massive use of carry-save adders and Wallace tree, and fast accumulation. Symmetric FIR filter is widely used for obtaining linear phase in signal processing. Its implementation can be further simplified owing to the symmetric coefficients. The architectures of symmetric filter is particularly studied. A new encoding is proposed to increase the speed both in bit serial and in bit-parallel design. Fully parallel implementation of FIR filter is also addressed. The new encoding proves to be more efficient than the modified Booth's encoding for parallel symmetric filter implementation.

Chapter 7 concludes the thesis and discusses the future research directions.

### 1.3 Terminology and remarks

For the sake of a clear presentation, it is necessary to clarify certain terminologies. It is often confused between linear convolution, cyclic convolution, aperiodic convolution, polynomial product and FIR filtering. What are their relations and their differences? We begin our explanation by stating their differences:

a) linear convolution, aperiodic convolution, polynomial product are the same operation on two finite length sequences: one of length  $M$ , the other of length  $N$ . Their computation are often expressed as a polynomial product of a  $(M-1)$ -degree polynomial and a  $(N-1)$ -degree one:

$$\begin{aligned} & y_0 + y_1 z + y_2 z^2 + \dots + y_{M+N-1} z^{M+N-1} \\ &= (h_0 + h_1 z + h_2 z^2 + \dots + h_{M-1} z^{M-1})(x_0 + x_1 z + x_2 z^2 + \dots + x_{N-1} z^{N-1}) \end{aligned} \quad (1.1)$$

The result is a sequence of  $M+N-1$  numbers. In the sequel, only the terms "aperiodic convolution"(AC in short) and "polynomial product" are employed. When  $M=N$ , we often call it a radix- $M$  aperiodic convolution or polynomial product.

b) cyclic convolution is an operation on two sequences of equal length  $M$ . Under polynomial presentation, it is well illustrated by the product of two  $(M-1)$ -degree polynomials modulo  $(z^M-1)$ :

$$\begin{aligned} & y_0 + y_1 z + y_2 z^2 + \dots + y_{M-1} z^{M-1} \\ &= (h_0 + h_1 z + h_2 z^2 + \dots + h_{M-1} z^{M-1})(x_0 + x_1 z + x_2 z^2 + \dots + x_{M-1} z^{M-1}) \bmod (z^M - 1) \end{aligned} \quad (1.2)$$

The result is also a sequence of  $M$  numbers. We will term it as length- $M$  cyclic convolution.

c) FIR filtering is an operation on two sequences: an infinite one and a finite one of length  $M$ . Using  $z$ -transform, it can be formulate as a product of two polynomials: one of infinite degree , the other of degree  $M-1$ .

$$\begin{aligned} & y_0 + y_1 z + y_2 z^2 + \dots \\ &= (h_0 + h_1 z + h_2 z^2 + \dots + h_{M-1} z^{M-1})(x_0 + x_1 z + x_2 z^2 + \dots) \end{aligned} \quad (1.3)$$

The result is an infinite sequence. This operation is referred to as length-M FIR filtering.

Although the three operations are quite different from each other, they are closely related:

- 1) an aperiodic convolution of a length-M sequence and a length-N one can be computed by a length-(M+N-1) cyclic convolution.
- 2) a length-L FIR filtering can be computed by sectioning the infinite sequence into blocks of M numbers, then using an aperiodic convolution of a length-M sequence and a length-L one plus overlap-add. According to a), cyclic convolution applies too.
- 3) a length-L FIR filtering can be computed by sectioning the infinite sequence into overlapped blocks of N numbers with K overlapping ones ( $K < N$ ), then using length-N cyclic convolution. This method is often called overlap-save.

In the first part, we deal with above all the FIR filtering algorithms. The filter's coefficients are taken as constant or fixed. However they can be extended to computing adaptive transversal filters [Duh89] in a fast way.

In the second part, our attention is mainly concentrated on the FIR filter architectures. However, the result in the Chapter 5 is applicable to a wider range of digital system design: multipliers, ALU of a general purpose processors and inner product computers, etc. The result in the Chapter 6 is also suitable for implementing inner product computers and adaptive transversal filters.

## Chapter 2

### A unified approach to the fast FIR filtering algorithms

FIR digital filters are widely used in digital signal filtering, and belong to the set of the most time consuming tasks in many systems. It is therefore of interest to derive efficient algorithms for computing FIR filtering. Many different approaches have been applied to solve this problem. Cyclic convolution based algorithms were first proposed [Sto66] soon after the rediscovery of fast Fourier transform algorithm (FFT). Aperiodic convolution algorithms were also applied to the FIR filtering [Aga74]. These algorithms are very efficient in terms of arithmetic complexity, and are therefore well suited for implementation on general purpose computers. A direct approach working directly on the FIR filtering equations was also proposed by Winograd [Win80]. All these algorithms are featured by the fact that arithmetic efficiency is obtained by working on large data blocks, and involve a global exchange of data inside this large vector.

The evolution of VLSI technology and the emergence of Digital Signal Processors have greatly boosted the real time signal processing. However, these kinds of implementations bring about some new constraints, and all the above algorithms are not always efficiently implemented through these new technologies because of their complex structure. For example, DSP's are highly optimized for implementing a multiply-accumulate operation. Therefore, all those "fast" algorithms were less efficient than the straightforward one when implemented on DSP's, except for very large filters or data blocks.

To overcome these problems, the authors proposed recently a new class of algorithms [Mou87]. Similar approaches were independently proposed at the same time by Vetterli [Vet88] and Kwan-Tsim [Kwa87].

At first glance, the new algorithms are quite different from the classical FIR filtering algorithms (large block convolution). Nevertheless, in



this paper we will show that all the above algorithms can be derived in a unified way and that this new approach provides all the intermediates between direct FIR computation and large block fast convolution algorithms, and between processing in time and in frequency, while opening some new possibilities.

Our objective is to unify all the main FIR filtering algorithms and to review the basic aspects of their derivation, in order to contribute to a better understanding to this problem, while providing some directions of the search for new algorithms.

The first step of the approach consists in a formulation of FIR filtering as a product of two polynomials (also used in [Mou87,Kwa87,Vet88]), which is then computed through the Chinese Remainder Theorem (CRT). Interpolating the polynomial product in different domains leads to different classes of algorithms while interpolating in different points leads to different algorithms in the same class. Thirdly this formulation is shown to lead to overlap-add algorithms. Transposition of the algorithms can result in all the overlap-save ones. We will show that the overlap-add and overlap-save techniques are essential to all kinds of algorithms, even to the FIR transversal filter itself. Since overlap-add and overlap-save schemes are the transpose of each other we show that they have the same arithmetic complexity.

## 2.1 Formulation of FIR filtering as a polynomial product

The FIR filtering is usually expressed as a convolution of two sequences: a finite sequence  $\{h_i\}$  and an infinite one  $\{x_i\}$ . The result of the convolution is also an infinite sequence  $\{y_i\}$ . Their relation is as follows:

$$y_n = \sum_{i=0}^{L-1} x_{n-i} h_i \quad n = 0, 1, 2, \dots, \infty \quad (2.1)$$

It can be also written in the z-domain as a polynomial product of  $H(z)$  and  $X(z)$ :

$$Y(z) = H(z)X(z) \quad (2.2)$$

$H(z)$ ,  $X(z)$  and  $Y(z)$  are the z transform of  $\{h_i\}$ ,  $\{x_i\}$  and  $\{y_i\}$  respectively.  $H(z)$  is a polynomial of finite degree  $(L-1)$  while the other two are of infinite degree.

By applying polyphase decomposition [Bel74] to each of the three terms in eq.(2), we get a polynomial product as follows:

$$\begin{aligned}
& Y_0 + Y_1 z^{-1} + \dots + Y_{N-1} z^{-N+1} \\
& = (H_0 + H_1 z^{-1} + \dots + H_{N-1} z^{-N+1}) (X_0 + X_1 z^{-1} + \dots + X_{N-1} z^{-N+1})
\end{aligned} \tag{2.3}$$

where  $\{H_i\}$ ,  $\{X_i\}$  and  $\{Y_i\}$  are polynomials in  $z^{-N}$ :

$$\begin{aligned}
H_j(z^N) &= \sum_{m=0}^{LN-1} h_{mN+j} z^{-mN} \\
X_k(z^N) &= \sum_{m=0}^{\infty} x_{mN+k} z^{-mN} \quad j,k,i = 0,1,\dots,\infty \\
Y_i(z^N) &= \sum_{m=0}^{\infty} y_{mN+i} z^{-mN}
\end{aligned} \tag{2.4}$$

In fact,  $H(z^{-1})$ ,  $X(z^{-1})$  and  $Y(z^{-1})$  are each decimated into  $N$  interleaved sequences. The decimation rate  $N$  will be an important factor in our approach to construct fast FIR filtering algorithms. It represents the number of consecutive outputs to be computed together.

## 2.2 Fast computation of a polynomial product

The right side of eq.(2.3) is a product of two finite degree polynomials whose coefficients are themselves polynomials. The product has  $2N-1$  coefficients:

$$\begin{aligned}
Q(z) &= P_1(z) P_2(z) \\
&= C_0 + C_1 z + \dots + C_{N-1} z^{(N-1)} + \dots + C_{2N-2} z^{(2N-2)}
\end{aligned} \tag{2.5}$$

where

$$\begin{aligned}
P_1(z) &= H_0 + H_1 z + \dots + H_{N-1} z^{N-1} \\
P_2(z) &= X_0 + X_1 z + \dots + X_{N-1} z^{N-1}
\end{aligned} \tag{2.6}$$

Direct computation of  $\{C_i\}$  would require  $N^2$  multiplications between  $\{H_j\}$  and  $\{X_i\}$ . Although such multiplications are in fact still FIR filterings, we take them first as usual multiplications. Nevertheless the application of the Chinese Remainder Theorem (CRT) is known to largely reduce the arithmetic

complexity. Using the CRT, Winograd has proved that  $(2N-1)$  multiplications are sufficient to compute  $\{C_i\}$ , instead of  $N^2$ . The theory of Winograd establishes the basis for fast algorithms.

Since the CRT plays an important role in fast algorithm derivation, let us first describe it.

### 2.2.1 The Chinese Remainder Theorem

This theorem dates back to A.D.100 [Sod86]. Originally, it is a part of number theory. We are only interested in its extension to polynomial ring, since we discuss the polynomial product.

**Chinese Remainder Theorem:** Given a ring of polynomials modulo  $P(z)$  and

$$P(z) = \prod_{i=1}^s p_i(z)$$

$\{p_i(z)\}$  are relatively prime, i.e.,  $p_i(z)$  has no common factors with  $p_j(z)$  when  $i \neq j$  for  $i, j \in \{1, 2, \dots, s\}$ . Then for any polynomial  $Q(z)$ , the following equation holds:

$$Q(z) \equiv \sum_{i=1}^s T_i(z) q_i(z) \pmod{P(z)}$$

where

$$\begin{aligned} q_i(z) &\equiv Q(z) \pmod{p_i(z)} \\ T_i(z) &\equiv 1 \pmod{p_i(z)} \\ &\equiv 0 \pmod{p_j(z)} \quad \text{for } j \neq i. \end{aligned}$$

The theorem states that, given only the residues  $\{q_i(z)\}$  of an unknown polynomial  $Q(z)$  in  $s$  distinct polynomial rings modulo  $p_i(z)$ , we can reconstruct  $Q(z)$  in the polynomial ring modulo  $P(z)$  from these residues.  $Q(z)$  will be unique if  $P(z)$  has higher order. The reconstructing polynomials  $\{T_i(z)\}$  depend only on  $\{p_i(z)\}$  and can be obtained by solving the following equation:

$$T_i(z) \equiv T'_i(z) \prod_{\substack{j=1 \\ j \neq i}}^s p_j(z) \equiv 1 \pmod{p_i(z)}$$

### 2.2.2 Interpolation

The use of the CRT for computing a polynomial product is made mostly

by choosing :

$$p_i(z) = z - a_i; \quad i=0,1,\dots,2N-2 \text{ and } a_i \neq a_j \text{ if } j \neq i \quad (2.7)$$

It is easy to evaluate:

$$\begin{aligned} P_1(a_i) &\equiv P_1(z) \bmod (z - a_i) = \sum_{k=0}^{M-1} H_k a_i^k \\ P_2(a_i) &\equiv P_2(z) \bmod (z - a_i) = \sum_{j=0}^{N-1} X_j a_i^j \end{aligned} \quad (2.8)$$

### 2.2.3 Filtering

For obtaining the final result  $Q(z)$ , we need to know the values  $Q(a_i)$  which is the residue  $q_i(z)$  of  $Q(z)$  modulo  $(z - a_i)$ . Since  $Q(z)$  is the product of  $P_1$  and  $P_2$ , we have

$$q_i(z) = Q(a_i) = P_1(a_i) P_2(a_i); \quad i=0,1,\dots,2N-2 \quad (2.9)$$

Let us recall that  $P_1(a_i)$  and  $P_2(a_i)$  are the combinations of  $\{H_j\}$  and  $\{X_i\}$  respectively, while the latters are functions of  $z^{-N}$ . Hence the above equations are still FIR filterings but at the sampling rate  $N$  times lower than the initial one.

Since the filter coefficients are constant, we can compute  $\{P_1(a_i), i = 0,1,\dots,2N-2\}$  before the filtering. This is one of the reasons that the computational load can be reduced, resulting in fast algorithms. This computation will not be taken into account in the subsequent evaluation of arithmetic complexity.

### 2.2.4 Reconstruction by the CRT

$\{T_i\}$  are precomputed, and when  $P_i(z) = z - a_i$ , it turns out that their expression is fairly simple:

$$T_i = \prod_{\substack{j=0 \\ j \neq i}}^{N+M-2} \frac{z - a_j}{a_i - a_j} = \frac{P(z)}{P'(a_i)(z - a_i)} \quad (2.10)$$

and application of the CRT results in  $Q(z)$ :

$$Q(z) = \sum_{i=0}^{N+M-2} Q(a_i) \prod_{\substack{j=0 \\ i \neq j}}^{N+M-2} \frac{z - a_j}{a_i - a_j} = P(z) \sum_{i=0}^{N+M-2} \frac{Q(a_i)}{P'(a_i)(z - a_i)} \quad (2.11)$$

which is very similar to the interpolation formulae of Lagrange. This is why  $\{a_i\}$  are often referred as interpolation points.

### 2.3 Overlap

By comparing the left side and the right side in eq.(2.3) or (2.5), we get:

$$Y_{N-1} = C_{N-1}$$

$$Y_i = C_i + C_{i+N} z^{-N}; \quad \text{for } i = 0, 1, \dots, N-2 \quad (2.12)$$

We can observe that eq.(2.12) represents a procedure of overlap-add. All the  $\{C_i; i = 0, 1, \dots, 2N-2\}$  are computed at one time and a block of  $N$  outputs  $\{Y_i\}$  is obtained by adding the terms  $\{C_{i+N}; i=0, 1, \dots, N-1\}$  overlapping from the previous block. Till now we have accomplished all the steps for constructing an FIR filtering algorithm, as summarized below :

- 1) decimating  $H(z)$ ,  $X(z)$  and  $Y(z)$  at rate  $N$  to formulate the FIR filtering as an polynomial product ;
- 2) interpolation of  $P_1$  and  $P_2$  at  $2N-1$  appropriate points  $a_i$  ;
- 3) filtering like a dot product :  $Q(a_i) = P_1(a_i) P_2(a_i)$  ,  $i = 0, 1, \dots, 2N-2$ ;
- 4) reconstruction of  $Q(z)$  from its interpolated values  $Q(a_i)$ ;
- 5) overlap according to eq.(2.12).

A general scheme is given in Fig.2.1. The above algorithm computes a block of  $N$  outputs.

The overlap differs the FIR filtering from other kinds of convolutions. It guarantees the continuity of computation and makes the filter running, while the cyclic convolution or the aperiodic convolution is a "local" computation.

It is well-known that a transversal FIR filter can be transposed. The transposed filter has the same transfer function. However the above

algorithms can be also transposed as will be shown in the following. This will lead to some connections between several types of algorithms.

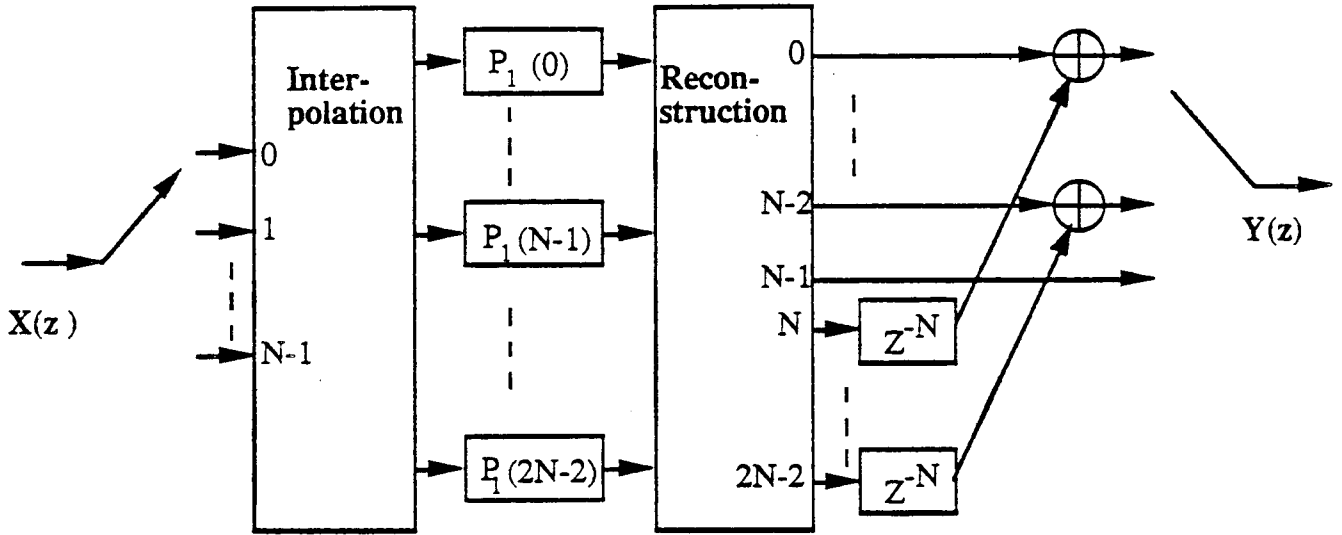


Fig.2.1 General fast FIR filtering scheme

## 2.4 Pseudocyclic convolution

We denote in the following an algorithm computing  $N_1$  outputs of the filter and using  $N_2$  as decimating rate by  $F(N_1, N_2)$ . Although algorithms can be derived for  $N_1 \neq N_2$ , we will show in the Chapter 4 that they involve very complex structures and do not have any advantage over other algorithms. We will concentrate our investigation on the case where  $N_1 = N_2$ .

Considering eq.(2.3) and eq.(2.12) we may write the following equation :

$$\begin{bmatrix} Y_{N-1} \\ Y_{N-2} \\ \vdots \\ Y_0 \end{bmatrix} = \begin{bmatrix} H_0 & H_1 & \dots & H_{N-1} \\ z^{-N}H_{N-1} & H_0 & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & H_1 \\ z^{-N}H_1 & \vdots & \vdots & z^{-N}H_{N-1}H_0 \end{bmatrix} \begin{bmatrix} X_{N-1} \\ X_{N-2} \\ \vdots \\ X_0 \end{bmatrix} \quad (2.13)$$

The right side is the product of a pseudocirculant matrix [Vai88] and a vector, or pseudocyclic convolution, which is in fact an  $F(N, N)$  system. In the context of pseudocircularity, we will show in the following the way to transpose an  $F(N, N)$  system and the way to obtain transposed version.

### 2.4.1 Diagonalization of a pseudocirculant matrix

In cyclic convolution, a fast algorithm diagonalizes a circulant matrix, the diagonalizing matrix is referred as rectangular transform [Aga76]. By analogy, a fast FIR filtering algorithm diagonalizes a pseudocirculant matrix. However the diagonalization here means a generalization of its conventional sense.

For example, a 2x2 pseudocirculant matrix can be diagonalized as follows [Kwa87,Vet88]:

$$\begin{bmatrix} h_0 & h_1 \\ z^{-2} h_1 & h_0 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 0 & z^{-2} \end{bmatrix} \begin{bmatrix} h_0 & & \\ & h_0 + h_1 & \\ & & h_1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \quad (2.14)$$

It is just the matrix representation of the algorithm presented in [Mou87]. In general an NxN pseudocirculant matrix can be diagonalized into an MxM diagonal matrix with  $M \geq 2N-1$ . This lower bound is the natural result of Winograd's theory [Win80]. All the 'fast' algorithms have  $M < N^2$ .

We may also define a rectangular transform for the pseudocyclic convolution. There exists the term  $z^{-N}$  in the transform matrix which recalls the continuity of computing. However most of the subsequent presentations employ other terminologies, in order to differ the FIR filtering algorithms from the cyclic convolution.

Vaidyanathan [Vai88] proposed a diagonalization of pseudocirculant by similarity transform, but this does not lead to fast algorithms, because the elements in the diagonal matrix are complex polynomials rather than simple numbers.

### 2.4.2 Transposition of an F(N,N) algorithm

Let us consider an F(N,N) algorithm as an (N-input, N-output) system shown in Fig.2.2(a). Its transmission matrix is pseudocirculant:

$$P(z) = \begin{bmatrix} H_0 & H_1 & \dots & & H_{N-1} \\ z^{-N}H_{N-1} & H_0 & & & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & & \cdot & \cdot & H_1 \\ z^{-N}H_1 & \cdot & \cdot & \cdot & z^{-N}H_{N-1}H_0 \end{bmatrix} \quad (2.15)$$

The transposed system (shown in Fig.2.2(b)) will have  $P^t(z)$ , which is the transpose of  $P(z)$ , as its transmission matrix according to Tellegen's Theorem for digital networks [Cro83].

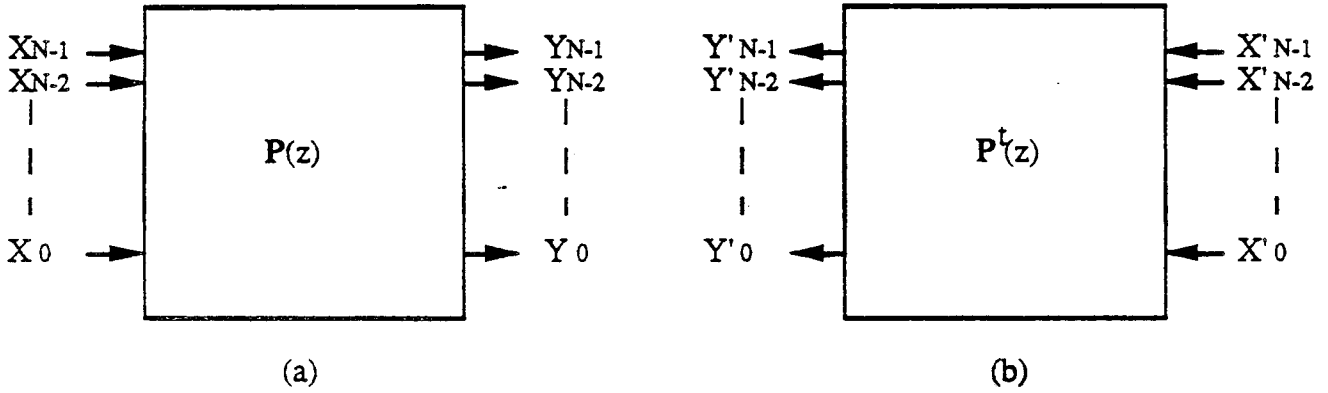


Fig.2.2 (a) initial system; (b) transposed system.

Unlike (1-input, 1-output) systems, an (N-input, N-output) system's transpose will not perform the same function as the initial one unless  $P(z)$  is symmetric, i.e.,  $P(z) = P^t(z)$ . Owing to  $P$ 's Toeplitz structure, we can manage to get the transposed system performing the same function as the initial one.

The transposed system is as follows:

$$\begin{bmatrix} Y'_{N-1} \\ Y'_{N-2} \\ \cdot \\ \cdot \\ \cdot \\ Y'_0 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-N}H_{N-1} & \cdot & \cdot & z^{-N}H_1 \\ H_1 & H_0 & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & & \cdot & \cdot & \cdot \\ \cdot & & & \cdot & z^{-N}H_{N-1} \\ H_{N-1} & \cdot & \cdot & \cdot & H_1 & H_0 \end{bmatrix} \begin{bmatrix} X'_{N-1} \\ X'_{N-2} \\ \cdot \\ \cdot \\ \cdot \\ X'_0 \end{bmatrix} \quad (2.16)$$



If we permute the input elements  $\{X'_i\}$  and the output elements  $\{Y'_i\}$  in such a way that their order is reversed, we will get the transposed system to perform the appropriate function:

$$\begin{bmatrix} Y'_0 \\ Y'_1 \\ \vdots \\ Y'_{N-1} \end{bmatrix} = \begin{bmatrix} H_0 & H_1 & \dots & H_{N-1} \\ z^{-N}H_{N-1} & H_0 & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & H_1 \\ z^{-N}H_1 & \vdots & \vdots & z^{-N}H_{N-1}H_0 \end{bmatrix} \begin{bmatrix} X'_0 \\ X'_1 \\ \vdots \\ X'_{N-1} \end{bmatrix} \quad (2.17)$$

The above demonstration is general, so that all (N-input,N-output) systems whose transmission matrix is Toeplitz can be transposed to perform the same function as the initial system after permuting the inputs and outputs in a reversed order. Thus, it can be applied to all kinds of convolution systems. Winograd has already proposed to transpose circular convolution algorithms in the same manner [Win80]. We summarize this principle as the following theorem.

**Theorem:** Having Toeplitz transmission matrix is sufficient for an (N-input,N-output) system to be transposed and to perform the same function after permuting the inputs and outputs in a reversed order.

Hence, we can transpose an  $F(N,N)$  algorithm in a rather easy way. In fact, a fast  $F(N,N)$  algorithm diagonalizes a pseudocirculant matrix:

$$P(z) = A_{N \times M} H_{M \times M} B_{M \times N} \quad (2.18)$$

where  $H_{M \times M}$  is a diagonal matrix. Then the transposition of  $P(z)$  is

$$P^t(z) = B^t H A^t \quad (2.19)$$

Permuting the inputs and outputs in a reversed order is equivalent to multiply  $J = \text{antidiag}(1,1,\dots,1)$ . Then the following equation holds:

$$P(z) = (JB^t) H (A^tJ) \quad (2.20)$$

Therefore, we get the transposed version of (2.18). The new pair of diagonalizing matrix is  $(JB^t)$  and  $(A^tJ)$ .

Let us consider as an example an F(2,2) algorithm, as given in (2.14). It is expressed in matrix form as:

$$\begin{aligned}
 \begin{bmatrix} y_1 \\ y_0 \end{bmatrix} &= \begin{bmatrix} h_0 & h_1 \\ z^{-2} h_1 & h_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \\
 &= \begin{bmatrix} -1 & 1 & -1 \\ 1 & 0 & z^{-2} \end{bmatrix} \begin{bmatrix} h_0 & & \\ & h_0 + h_1 & \\ & & h_1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix}
 \end{aligned} \tag{2.21}$$

therefore

$$A = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 0 & z^{-2} \end{bmatrix} \quad B = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Following (2.20), we obtain the transposed version of (2.21):

$$\begin{bmatrix} y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} h_0 & & \\ & h_0 + h_1 & \\ & & h_1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ z^{-2} & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \tag{2.22}$$

### 2.4.3 Identical arithmetic complexity in direct and transposed forms

It is clear, following the above explanations, that the multiplicative complexity is not changed by transposition. Let us consider the additive complexity in an (N-input, N-output) system.

A digital network is composed of only branches and nodes. There are two kinds of nodes: a (M,1) summing node that adds M inputs into 1 output, and a (1,M) branching node that branches 1 input into M outputs. A (M,1) summing node can be split into M-1 (2,1) summing nodes. A (1,M) branching node can be also split into M-1 (1,2) branching nodes. Then it is easy to transform the network to an equivalent one having only (2,1) summing nodes and (1,2) branching nodes. After transposition, the summing nodes become branching nodes and vice versa.

The number of additions in the initial network is equal to that of (2,1) summing nodes, denoted by  $N_s$ . The number of additions in the transposed network is equal to that of (2,1) branching nodes in the initial network, denoted by  $N_b$ . The additive complexity in initial and transposed algorithms is identical if and only if  $N_s = N_b$ , and we show in the following that this property holds for (N-input, N-output) systems.

**Proof:** For an (N-input, N-output) network, if we connect the N inputs to the N outputs graphically, we get a closed network where every branch coming out from a node should enter into another node. Then the number of outputs of all nodes is equal to that of inputs of all nodes. A (2,1) summing node has two inputs and one output while a (1,2) branching node has one input and two outputs. We get:

$$N_b + 2N_s = 2N_b + N_s$$

$$N_s = N_b$$

end of proof.

## 2.5 Remarks

A general approach to derive fast FIR filtering algorithms has been described. Some parameters such as decimating rate  $N$  and interpolation points are not defined. We will show in the next chapter how the choice of these parameters affects the resulting algorithm. In fact all the classical algorithms choose  $N$  tacitly greater than the filter's length. However it is also feasible using smaller  $N$  to derive algorithms of interest.

The pseudocyclic convolution is a good representation of the FIR filtering process. Several properties, such as transposability of an  $F(N, N)$  algorithm, equality of arithmetic complexity in direct and transposed form, have been proved under the representation.

## Chapter 3

### Review on the FIR filtering algorithms and new possibilities

In this chapter we will show that the classical algorithms are shown to be special cases of the approach presented in the previous chapter. These algorithms include three typical ones: FFT-based algorithms proposed by Stockham [Sto66]; Algorithms using aperiodic convolutions proposed by Agarwal and Burrus [Aga74]; Winograd algorithms [Win80].

Some new possibilities are also investigated. First the recently proposed algorithms [Mou87,Kwa87,Vet88], termed short length FIR filtering algorithms (SLFIR), are briefly described. A comprehensive study on this class of algorithms is in the next chapter. Secondly we present shorter FFT-based algorithms while in the frequency domain either real or complex FIR filterings are performed. This class of algorithms represents all the intermediate ones between processing in time and in frequency. Short length complex FIR filtering algorithms (SLCFIR) will also be derived using the approach.

It is a general belief that when the filter's length  $L$  is large, classical FFT-based scheme will outperform all the other algorithms. However the combination of short length real and complex FIR filtering algorithms and the shorter FFT-based schemes results in an algorithm requiring even less arithmetic operations than the classical ones as far as  $L \geq 4$ .

#### 3.1 Review on the classical algorithms

The "fast" (in the sense of reducing arithmetic complexity) computation of FIR filtering through FFT and its inverse was soon recognized to be speed-efficient on general-purpose computers [Sto66] after the rediscovery of FFT. High speed convolvers based on FFT's were also built in hardware.

Nevertheless, the FFT-based FIR filtering implies to work on large overlapping signal block, thus resulting in heavy hardware requirement. Furthermore, since most Digital Signal Processor (DSP's) are optimized for a multiply-accumulate structure, transform-based FIR filtering is of no practical interest on DSP's.

There were the motivations for the proposal of a new class of algorithms [Mou87], reducing the arithmetic complexity but retaining the multiply-accumulate structure.

However the above two classes of algorithms have quite different presentations. Nevertheless, we will show in the following that they are just special cases of the general scheme in Fig.2.1.

### 3.1.1 FFT-based algorithms

Let us choose  $N \geq L$ , then

$$\begin{aligned} H_i(z^{-N}) &= h_i & 0 \leq i \leq L-1 \\ &= 0 & L \leq i \leq N-1 \end{aligned} \quad (3.1)$$

Replacing  $\{H_i\}$  in eq.(2.6) by the the above terms, we get:

$$P_1(z) = h_0 + h_1 z + h_2 z^2 + \dots + h_{L-1} z^{L-1} \quad (3.2)$$

$P_2(z)$  remains unchanged as in (2.6). Since the polynomial  $P_1$  is now of degree  $L-1$  and  $L \leq N$ , the minimum number of necessary interpolation points is  $N+L-1$  instead of  $2N-1$ . If we choose  $K(\geq N+L-1)$  points on the unit circle  $\{\exp(-2\pi i/K); i = 0, 1, \dots, K-1\}$  as interpolation points, then

$$\begin{aligned} P_1(a_i) &= \sum_{k=0}^{L-1} h_k W^{ik} \\ P_2(a_i) &= \sum_{k=0}^{N-1} X_k W^{ik} \end{aligned} \quad (3.3)$$

$\{P_1(a_i); i = 0, 1, \dots, K-1\}$  is the length- $K$  DFT of  $\{h_i; i = 0, 1, \dots, L-1\}$  so that FFT's applies for the computation of the interpolation values. Likewise,  $\{P_2(a_i); i =$

$0, 1, \dots, K-1$  is the length- $K$  DFT of  $\{X_i; i = 0, 1, \dots, N-1\}$ . After the pointwise multiplications  $P_1(a_i)P_2(a_i) = Q(a_i)$ , we can show:

$$Q(z) = \sum_{i=0}^{N+L-2} Q(a_i) \sum_{j=0}^{N+L-2} \frac{1}{N} W^{-ij} z^j \quad (3.4)$$

The coefficients of  $Q(z)$  is exactly the inverse DFT of  $\{Q(a_i)\}$ :

$$C_j = \frac{1}{N} \sum_{i=0}^{N+L-2} Q(a_i) W^{-ij} \quad (3.5)$$

The overlap-add technique is then applied to get the correct filter outputs. Fig.3.1 depicts the general scheme for all the FFT-based algorithms.  $\{P_1(a_i)\}$  can be precomputed. Its computation will not be counted in the subsequent evaluation of arithmetic complexity.

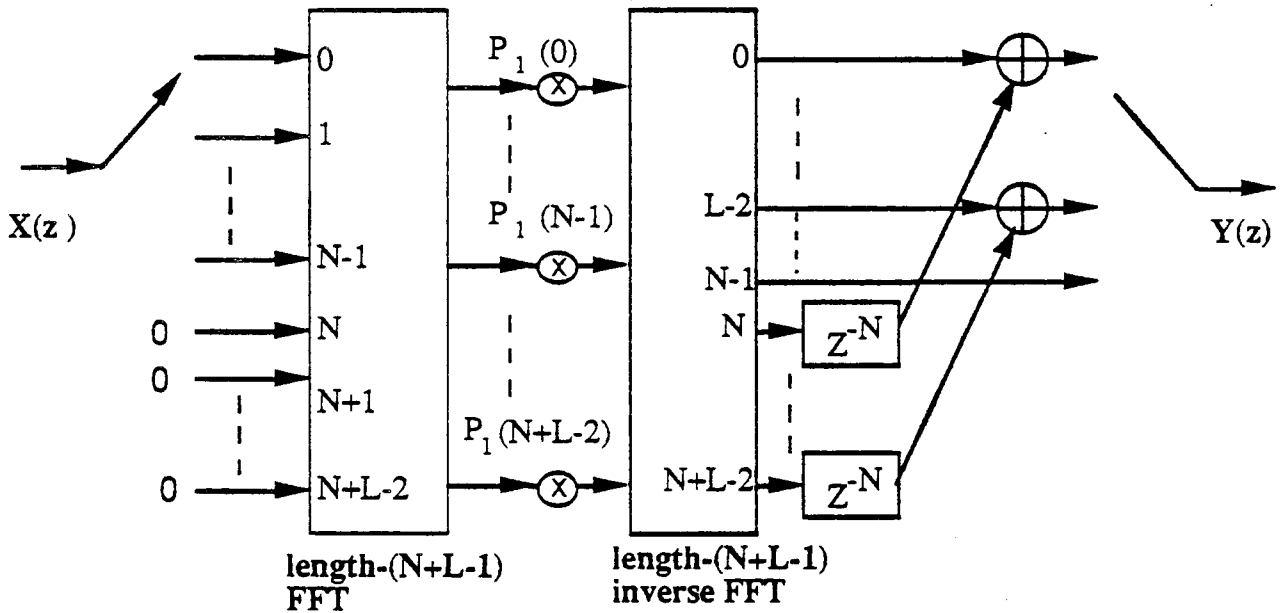


Fig.3.1 FFT-based overlap-add FIR filtering scheme

Due to the complex interpolation points all the computations have to be performed in the complex domain. When the data is real, we can apply the real-valued FFT algorithms to remove the redundancy that appears in the computation [Duh87, Sor87].

Let us take a simple example of computing a length- $2^m$  filter, usually we choose  $N=L=2^m$  so that a length- $2L$  DFT's and its inverse can be used to compute the  $N$  outputs together. Then  $K$  should be  $2L(=2^{m+1})$  instead of  $2L-1$ , since a length- $2L$  DFT's is much more easier to implement than a length- $(2L-1)$  one. Suppose we use the split-radix FFT and eliminate unnecessary additions on input and on output. The resulting arithmetic complexity is:

IIIa) length- $2^m$  real filter (per  $2^m$  outputs):

$$M_r = 2^m (2m-1) + 3$$

$$A_r = 2^m (6m-2) + 3$$

the arithmetic complexity per output is:

$$m_r = 2m-1 + 3 \cdot 2^{-m}$$

$$a_r = 6m-2 + 3 \cdot 2^{-m}$$

$$m_r + a_r = 8m - 3 + 6 \cdot 2^{-m}$$

A 3-real-multiplication-and-3-real-addition [Nus82] algorithm is used for the complex number multiplication for the above evaluation of arithmetic complexity.

There is a slight difference in arithmetic complexity between the above scheme and a length- $2^{m+1}$  cyclic convolution:

$$M_r = M_{cc}(2^{m+1})$$

$$A_r = A_{cc}(2^{m+1}) - 2^m - 2$$

The difference is due to the fact that the input block of length  $2^m$  is padded with  $2^m$  zeroes.

### 3.1.2 Algorithms using aperiodic convolutions

Agarwal and Burrus [Aga74] presented a class of algorithms using small radix aperiodic convolution (AC) algorithms to compute recursively a long polynomial product and then overlapping.

A radix-2 fast aperiodic convolution algorithm can be derived by the CRT using  $\{0,1,\infty\}$  as interpolation points:

$$\begin{aligned} & (x_0 + x_1 z)(h_0 + h_1 z) \\ &= x_0 h_0 + [(x_0 + x_1)(h_0 + h_1) - x_0 h_0 - x_1 h_1]z + x_1 h_1 z^2 \end{aligned} \quad (3.6)$$

Only 3 multiplications are required instead of 4.

We will show how to use the above algorithm to derive FIR filtering algorithms.

Let the decimating rate  $N=L$  (the length of the filter, assumed even). The filtering equation becomes:

$$\begin{aligned} & Y_0 + Y_1 z + \dots + Y_{L-1} z^{L-1} \\ &= (h_0 + h_1 z + \dots + h_{L-1} z^{L-1})(X_0 + X_1 z + \dots + X_{L-1} z^{L-1}) \end{aligned} \quad (3.7)$$

A reorganization of the above equation leads to:

$$\begin{aligned} & (Y_0 + Y_1 z + \dots + Y_{L/2-1} z^{L/2-1}) + z^{L/2} (Y_{L/2} + \dots + Y_{L-1} z^{L/2-1}) \\ &= [(h_0 + h_1 z + \dots + h_{L/2-1} z^{L/2-1}) + z^{L/2} (h_{L/2} + \dots + h_{L-1} z^{L/2-1})] \times \\ & \quad [(X_0 + X_1 z + \dots + X_{L/2-1} z^{L/2-1}) + z^{L/2} (X_{L/2} + \dots + X_{L-1} z^{L/2-1})] \end{aligned} \quad (3.8)$$

or

$$E_0 + E_1 z^{L/2} = (G_0 + G_1 z^{L/2})(F_0 + F_1 z^{L/2})$$

with

$$\begin{aligned} E_0 &= Y_0 + Y_1 z + \dots + Y_{L/2-1} z^{L/2-1} & E_1 &= Y_{L/2} + Y_{L/2+1} z + \dots + Y_{L-1} z^{L/2-1} \\ G_0 &= h_0 + h_1 z + \dots + h_{L/2-1} z^{L/2-1} & G_1 &= h_{L/2} + h_{L/2+1} z + \dots + h_{L-1} z^{L/2-1} \\ F_0 &= X_0 + X_1 z + \dots + X_{L/2-1} z^{L/2-1} & F_1 &= X_{L/2} + X_{L/2+1} z + \dots + X_{L-1} z^{L/2-1} \end{aligned}$$

Such formulation allows us to apply the radix-2 AC algorithm, resulting in:

$$\begin{aligned} & E_0 + E_1 z^{L/2} \\ &= G_0 F_0 + [(G_0 + G_1)(F_0 + F_1) - G_0 F_0 - G_1 F_1] z^{L/2} + G_1 F_1 z^L \end{aligned} \quad (3.9)$$

By overlapping, we get:



$$\begin{aligned} E_0 &= G_0F_0+ G_1F_1z^L \\ E_1 &= (G_0+ G_1) (F_0+F_1) - G_0F_0- G_1F_1 \end{aligned}$$

(3.10)

About 25% reduction can be achieved in the number of arithmetic operations. This gain is obtained by removing the redundancy existing between computing two groups of outputs:{Y<sub>0</sub>,Y<sub>1</sub>,...,Y<sub>L/2-1</sub>} and {Y<sub>L/2</sub>,...,Y<sub>L-1</sub>}.

Since G<sub>0</sub>F<sub>0</sub>, (G<sub>0</sub>+G<sub>1</sub>)(F<sub>0</sub>+F<sub>1</sub>), G<sub>1</sub>F<sub>1</sub> are themselves aperiodic convolutions, we can once more apply the radix-2 AC algorithm to them before overlapping. This is equivalent to a recursive interpolation using the same interpolating points. Further gain in arithmetic complexity can be obatined by removing the redundancy in computing four groups of outputs: {Y<sub>0</sub>,Y<sub>1</sub>,...,Y<sub>L/4-1</sub>}, {Y<sub>L/4</sub>,...,Y<sub>L/2-1</sub>}, {Y<sub>L/2</sub>,,....,Y<sub>3L/4-1</sub>} and {Y<sub>3L/4</sub>,...,Y<sub>L-1</sub>}. Evidently this procedure can be recursively applied, resulting in further reduction in arithmetic complexity. Agarwal and Burrus presented this technique using multidimensional formulation [Aga74].

For L=2<sup>m</sup> , a thorough recursion using the radix-2 algorithm results in an fast algorithm requiring 3<sup>m</sup> multiplicatons, instead of 4<sup>m</sup> by direct computation. The additive complexity is evaluated in [Nus82]. Otherwise (L-1) additions are necessary for overlapping to establish an FIR filtering algorithm. The arithmetic complexity of the filtering algorithms thus constructed is given in Table.3.1.

Table.3.1 Arithmetic complexity of Agarwal&Burrus filtering algorithms (per N outputs).

N	M	A
2	3	4
4	9	22
8	27	88
16	81	310
64	729	3262
128	2187	10168
256	6561	31271

The above formulation of a long aperiodic convolution as a radix-2 one is rather decimation-in-frequency, by analogy of the terms for defining FFT's. It is also feasible to derive a decimation-in-time algorithm by the following formulation:

$$\begin{aligned}
 & (Y_0 + Y_2 z^2 + \dots + Y_{L/2-1} z^{L-2}) + z(Y_1 + \dots + Y_{L-1} z^{L-2}) \\
 &= [(h_0 + h_2 z^2 + \dots + h_{L/2-1} z^{L-2}) + z(h_1 + \dots + h_{L-1} z^{L-2})] \times \\
 & \quad [(X_0 + X_2 z^2 + \dots + X_{L/2-1} z^{L-2}) + z(X_1 + \dots + X_{L-1} z^{L-2})]
 \end{aligned} \tag{3.11}$$

The radix-2 AC algorithm is then applicable to the above equation. Further recursion results in an algorithm with the same arithmetic complexity as before. A general scheme is given in Fig.3.2 for small-radix aperiodic convolution based FIR filtering algorithms.

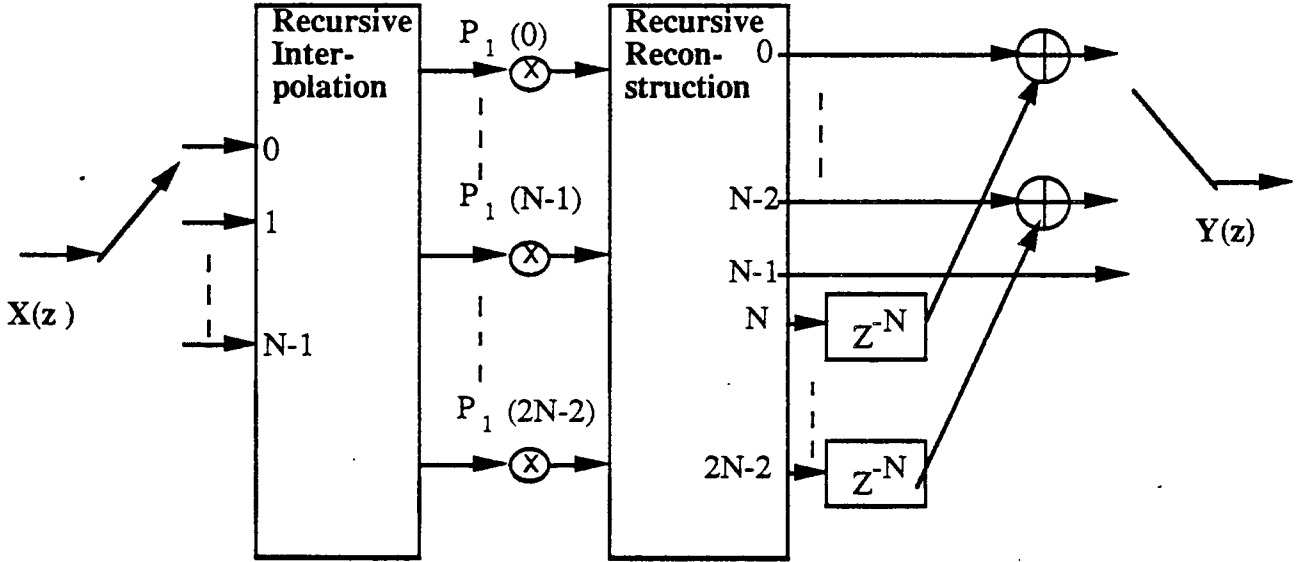


Fig.3.2 Fast FIR filtering scheme by Agarwal&Burrus technique

This kind of algorithms is often preferable for computing small or medium size FIR filters as well as for full precision implementation[Aga74].

### 3.1.3 Winograd algorithms

Winograd's theory [Win80] on the arithmetic complexity of computation is a milestone for fast FIR filtering algorithms as well as for FFT and polynomial product algorithms. He proposed an original class of FIR filtering

algorithms and provided the lower bound on the number of multiplications for computing an FIR filter.

It is intended by Winograd to compute the FIR filtering under the presentation below:

$$\begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-L+1} \end{bmatrix} = \begin{bmatrix} x_{n-L+1} & \cdot & \cdot & x_{n-1} & x_n \\ x_{n-L} & x_{n-L+1} & \cdot & x_{n-1} & \cdot \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n-2L+2} & \cdot & \cdot & x_{n-L} & x_{n-L+1} \end{bmatrix} \begin{bmatrix} h_{L-1} \\ h_{L-2} \\ \vdots \\ h_0 \end{bmatrix} \quad (3.12)$$

It is tacitly assumed to compute  $L$  outputs together, given  $L$  the filter's length.

First of all, a small radix aperiodic convolution algorithm, for example the one in eq.(3.6), is transposed to compute the following operation in a fast way [Win80]:

$$\begin{bmatrix} y_n \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} x_{n-1} & x_n \\ x_{n-2} & x_{n-1} \end{bmatrix} \begin{bmatrix} h_1 \\ h_0 \end{bmatrix} = \begin{bmatrix} x_{n-1} (h_1 + h_0) - (x_{n-1} - x_n) h_0 \\ x_{n-1} (h_1 + h_0) + (x_{n-2} - x_{n-1}) h_1 \end{bmatrix} \quad (3.13)$$

This is an algorithm computing 2 outputs of a length-2 FIR filter with only 3 multiplications. Eq.(3.12) can be formulated in block form:

$$\begin{bmatrix} E1 \\ E0 \end{bmatrix} = \begin{bmatrix} F0 & F1 \\ F2 & F0 \end{bmatrix} \begin{bmatrix} G1 \\ G0 \end{bmatrix} \quad (3.14)$$

with

$$E1 = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-L/2+1} \end{bmatrix} \quad E0 = \begin{bmatrix} y_{n-L/2} \\ y_{n-L/2-1} \\ \vdots \\ y_{n-L+1} \end{bmatrix} \quad G1 = \begin{bmatrix} h_{L-1} \\ h_{L-2} \\ \vdots \\ h_{L/2} \end{bmatrix} \quad G0 = \begin{bmatrix} h_{L/2-1} \\ h_{L/2-2} \\ \vdots \\ h_0 \end{bmatrix}$$

$$F0 = \begin{bmatrix} x_{n-L+1} & \cdot & \cdot & x_{n-L/2-1} & x_{n-L/2} \\ x_{n-L} & x_{n-L+1} & \cdot & \cdot & x_{n-L/2-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ x_{n-3L/2+2} & \cdot & \cdot & x_{n-L} & x_{n-L+1} \end{bmatrix}$$

$$F1 = \begin{bmatrix} x_{n-L/2+1} & \cdot & \cdot & x_{n-1} & x_n \\ x_{n-L/2} & x_{n-L/2+1} & \cdot & \cdot & x_{n-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ x_{n-L+2} & \cdot & \cdot & x_{n-L/2} & x_{n-L/2+1} \end{bmatrix}$$

$$F2 = \begin{bmatrix} x_{n-3L/2+1} & \cdot & \cdot & x_{n-L-1} & x_{n-L} \\ x_{n-3L/2} & x_{n-3L/2+1} & \cdot & \cdot & x_{n-L-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ x_{n-2L+2} & \cdot & \cdot & x_{n-3L/2} & x_{n-3L/2+1} \end{bmatrix}$$

Then (3.13) applies to the above computation for reduction of arithmetic complexity:

$$\begin{bmatrix} E1 \\ E0 \end{bmatrix} = \begin{bmatrix} F0(G1 + G0) - (F0 - F1)G0 \\ F0(G1 + G0) + (F2 - F0)G1 \end{bmatrix} \quad (3.15)$$

Further iteration of the algorithm in (3.13) is feasible in computing  $F0(G1+G0)$ ,  $(F0-F1)G0$ ,  $(F2-F0)G1$ . For  $L=2^m$ , a thorough iteration results in  $3^m$  multiplications. The arithmetic complexity for certain lengths is listed in Table.3.2.

Since eq.(3.13) is just the transpose of the radix-2 AC algorithm in (3.6), the recursive application of (3.13) is the transpose of that of (3.6). Hence we can conclude that Winograd algorithms are the transpose of those of Agarwal and Burrus.

We have proved in last chapter that both the direct and transposed forms of an FIR filtering algorithm have the same complexity. But Table.3.2

and 3.1 show different number of additions. The reason for such difference is that Winograd has taken into account the redundancy due to computing two successive blocks of outputs while Agarwal and Burrus were not concerned with the overlap.

Table.3.2 Arithmetic complexity of Winograd filtering algorithms (per N outputs).

N	M	A
2	3	4
4	9	20
8	27	76
16	81	260
64	729	2660
128	2187	8236
256	6561	25220

For example in case  $L=4$ , we have

$$\begin{aligned}
 F_0 - F_1 &= \begin{bmatrix} x_{n-3} - x_{n-1} & x_{n-2} - x_n \\ x_{n-4} - x_{n-2} & x_{n-3} - x_{n-1} \end{bmatrix} \\
 F_2 - F_0 &= \begin{bmatrix} x_{n-5} - x_{n-3} & x_{n-4} - x_{n-2} \\ x_{n-6} - x_{n-4} & x_{n-5} - x_{n-3} \end{bmatrix}
 \end{aligned} \tag{3.16}$$

Between  $(F_0-F_1)$  and  $(F_2-F_0)$ , there is a common addition  $(x_{n-4} - x_{n-2})$  so that one addition can be saved. Realizing that  $(x_{n-6} - x_{n-4})$  is already computed for last 4 outputs, we can save one more addition. Hence in case  $L=4$  Winograd algorithm requires 2 less additions than Agarwal&Burrus algorithm does as shown in Table.3.1 and 3.2. We may also remove the 2 extra additions in the latter algorithm but it is less systematic than in the Winograd algorithm.

### 3.2 Short length FIR filtering algorithms

Let  $N = 2$ . If we choose  $\{0,1,\infty\}$  as interpolation points, then

$$\begin{aligned}
P_1(a_0) &= H_0(z^2) & P_2(a_0) &= X_0(z^2) \\
P_1(a_1) &= H_0(z^2) + H_1(z^2) & P_2(a_1) &= X_0(z^2) + X_1(z^2) \\
P_1(a_2) &= H_1(z^2) & P_2(a_2) &= X_1(z^2)
\end{aligned} \tag{3.17}$$

Application of the procedure stated in the last chapter leads to the following algorithm:

$$\begin{aligned}
Y_0(z^2) &= P_1(a_0) P_2(a_0) + z^{-2} P_1(a_2) P_2(a_2) \\
Y_1(z^2) &= P_1(a_1) P_2(a_1) - P_1(a_0) P_2(a_0) - P_1(a_2) P_2(a_2)
\end{aligned} \tag{3.18}$$

Different authors have presented this result in their own way [Kwa87,Mou87,Vet88]. Of course further decompositions are still feasible, together with higher radix or mixed radix decomposition.

There are four simplest interpolation points in real number domain  $\{0,1,-1,\infty\}$ . The other variants can be obtained by using interpolation point sets like  $\{0,-1,\infty\}$ ,  $\{0,1,-1\}$  or  $\{1,-1,\infty\}$ . Although interpolation points other than  $\{0,1,-1,\infty\}$  can be used, they will lead to much more complicated algorithms and become out of practical interest. A study of real FIR filtering algorithms with small  $N$  can be found in the next chapter.

### 3.3 Shorter FFT-based algorithms

We present two new schemes in this section as results of the approach: one is based on shorter FFT's (length of FFT < length of the filter), the other is fast short length complex FIR filtering algorithms. We will show also that combining the two schemes may result in less arithmetic operations than conventional ones as mentioned in the last section.

We have shown that when  $N$  is large, the terms  $\{P_1(a_i) P_2(a_i); a_i = \exp(-2\pi i/K)\}$  are the products of two complex numbers. Now if we choose a relatively small  $N$  ( $<L$ ), the  $H_i(z^N)$  are no longer constants but higher degree polynomials and the terms  $P_1(a_i)P_2(a_i)$  become length- $L/N$  either complex or real FIR filtering. The general framework of algorithm will be a length- $(2N-1)$  DFT's plus  $2N-1$  complex FIR filters and then a length- $(2N-1)$  inverse DFT's. Overlap-add is performed at the last step. Nevertheless a length- $2N$  FFT transformer is simpler and thus is preferred [Kwa87,Mou88]. An alternate scheme is the

transposed version, i.e., the overlap-save scheme as depicted in Fig.3.3. When the data are real, further improvement can be brought about by applying the fast cyclic convolution algorithm for real data [Duh87,Sor87] to the computation of the polynomial product.

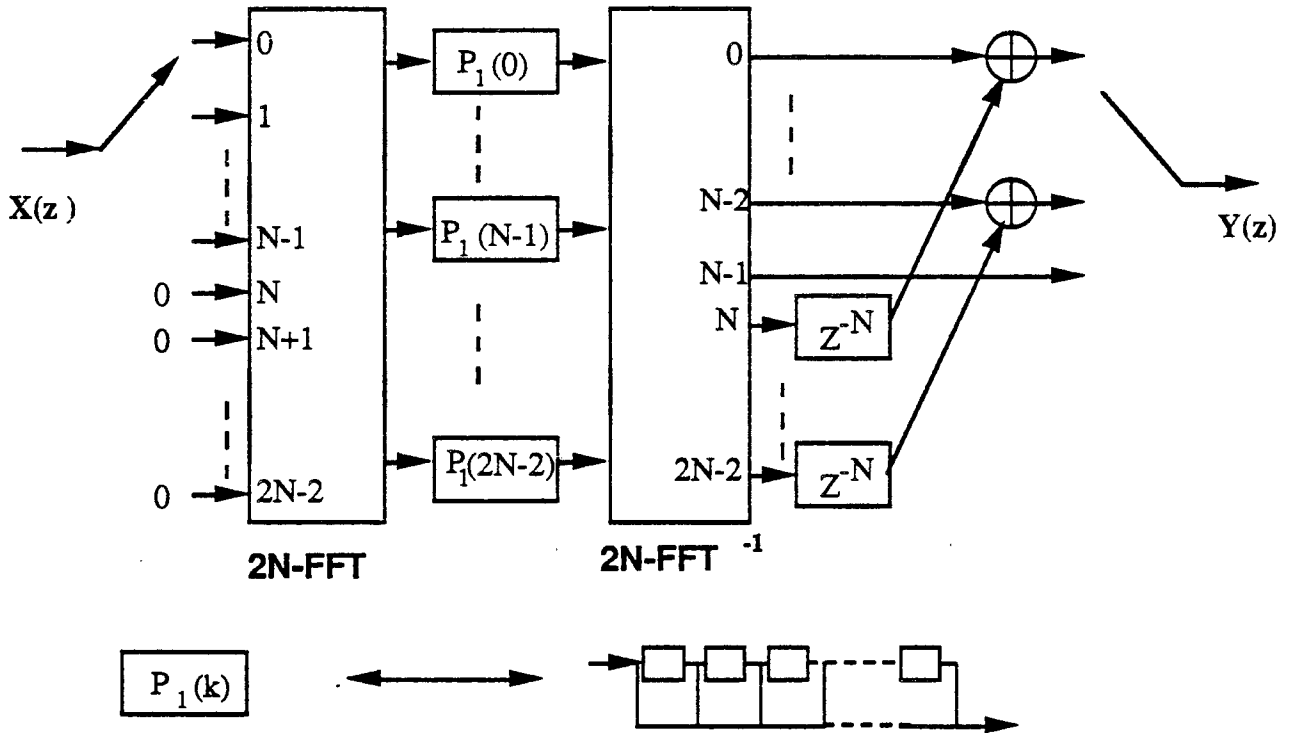


Fig.3.3 Shorter FFT-based FIR filtering scheme

In Fig.3.3, the FFT and FFT $^{-1}$  are used to perform the polynomial product, the remaining task is to compute the complex filters between FFT and FFT $^{-1}$ .

Let us restudy the case of computing a filter of length- $2^m$  for the sake of comparison. Now we choose  $N=L/2=2^{m-1}$ . Then according to eq.(2.4) we have

$$H_i = h_i + h_{i+L/2} z^{-L/2}; \quad i = 0, 1, \dots, L/2 - 1 \quad (3.19)$$

Using  $\{a_i = W^i = \exp(-2\pi i/2N) = \exp(-2\pi i/L); i=0, 1, \dots, L-1\}$  as interpolation points, we obtain a scheme based on length- $L$  FFT and its inverse, instead of length- $2L$  one. we derive explicitly the length-2 complex FIR filters in the frequency domain:

$$\begin{aligned}
 P_1(W^i) &= \frac{1}{L} \sum_{j=0}^{L/2-1} H_j W^{ij} \\
 &= \left( \frac{1}{L} \sum_{j=0}^{L/2-1} h_j W^{ij} \right) + \left( \frac{1}{L} \sum_{j=0}^{L/2-1} h_{j+L/2} W^{ij} \right) z^{-L/2} \quad i = 0, 1, \dots, L-1
 \end{aligned}
 \tag{3.20}$$

The scaling coefficient  $1/L$  is due to the inverse FFT in the scheme. By incorporating  $1/L$  into the complex FIR filters,  $L$  multiplications are avoided and the scaling is naturally performed. Since  $N=L/2$ ,  $2^{m-1}$  outputs are computed. The resulting arithmetic complexity is:

IIIb) length- $2^m$  real filter (per  $2^{m-1}$  outputs):

$$\begin{aligned}
 M_r &= 2^m m + 2 \\
 A_r &= 2^{m-1} (6m - 5) + 1
 \end{aligned}$$

the arithmetic complexity per output is:

$$\begin{aligned}
 m_r &= 2m + 2 \cdot 2^{-m+1} \\
 a_r &= 6m - 5 + 2 \cdot 2^{-m+1} \\
 m_r + a_r &= 8m - 5 + 8 \cdot 2^{-m}
 \end{aligned}$$

Taking into account the symmetry of the Fourier transform of real signal, we have used in the frequency domain only two length-2 real FIR filters and  $N/2-1$  ( $= 2^{m-1} - 1$ ) length-2 complex FIR filters. The fast computation of complex FIR filtering is similar to the fast complex number multiplication algorithm with 3 real multiplications and 3 real additions. This can be shown as follows. Complex FIR filtering is described as:

$$Y_r(z) + jY_i(z) = [H_r(z) + jH_i(z)][X_r(z) + jX_i(z)] \tag{3.21}$$

By applying the fast complex number multiplication algorithm, we obtain the following complex FIR filtering algorithm:

$$\begin{aligned}
 Y_r(z) &= H_r(z) [X_r(z) + X_i(z)] - [H_r(z) + H_i(z)] X_i(z) \\
 Y_i(z) &= H_r(z) [X_r(z) + X_i(z)] - [H_r(z) - H_i(z)] X_r(z)
 \end{aligned}
 \tag{3.22}$$



Fig.3.4 depicts the fast complex FIR filtering scheme. Note that  $(X_r+X_i)$  needs only one addition per output, since all the terms computed for the previous outputs can be stored and used for the present output. For a length- $L$  complex FIR filter only  $3L$  real multiplications and  $3L$  real additions are required per complex output datum. Then (6 mults, 6 adds) are used to compute each of the length-2 complex FIR filters.

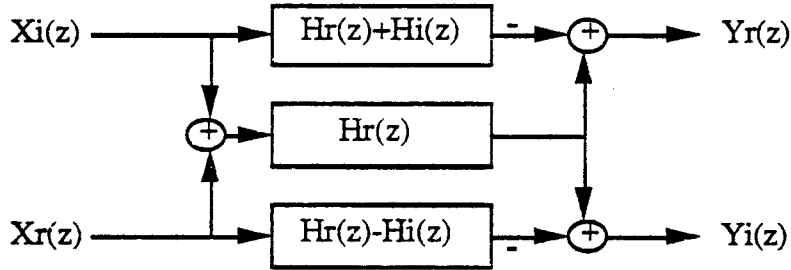


Fig.3.4 Complex FIR filtering using three real FIR filters

Comparing a) with b), we find that the new scheme requires more multiplications and less additions. Furthermore the sum of computational operations per output is reduced. In certain computers, a multiplication and an addition require almost the same computing time. The new scheme may be very competitive with the conventional one. It is also competitive in other computers since the difference of arithmetic complexity between the two schemes is very small. An important difference is that the conventional scheme needs length- $2L$  DFT's while the new scheme uses only length- $L$  DFT's. In general, smaller length DFT's can be more efficiently implemented. This is another advantage of the new scheme. Since the block length is proportional to the delay in processing, the system delay is also halved, a fact of interest for real time processing.

We can also choose  $N=L/4$  and  $\{a_i\}=\{\exp(-4\pi i/L); i=0,1,\dots,L/2-1\}$ . This results in a scheme with length- $L/2$  DFT's and length-4 FIR filters in the frequency domain. The arithmetic complexity is as follows:

IIIc) length- $2^m$  real filter (per  $2^{m-2}$  outputs):

$$M_r = 2^{m-2} (2m+4)$$

$$A_r = 2^{m-2} (6m - 5)$$

the arithmetic complexity per output is:

$$m_r = 2m+4$$

$$a_r = 6m - 5$$

$$m_r + a_r = 8m - 1$$

This scheme results in a slight increase in the arithmetic complexity. But such increase may be traded off by the reduction of the length of DFT's and of system delay.

### 3.4 Short length complex FIR filtering algorithms

Fast short length complex FIR filters find their applications both in the above schemes and in arbitrary length complex FIR filtering in the same manner as in real FIR filtering. This is our motivation to study these algorithms.

Eq.(3.18) is also applicable to complex filters. There are six simplest interpolation points in complex number domain  $\{0,1,-1,j,-j,\infty\}$ . Then we can construct 20 variants from these points for complex filtering. Further development of these variants is similar to (3.18) and will not be given here. We point out only that the F(2,2) algorithms with  $\{0,1,\infty\}$ ,  $\{0,-1,\infty\}$ ,  $\{0,-j,\infty\}$  and  $\{0,j,\infty\}$  as interpolation points require the least arithmetic complexity, i.e., 3 complex multiplications (cmults) and 4 complex additions (cadds) or (9 mults, 17 adds) in terms of real operations.

An optimal F(3,3) algorithm requires 5 interpolation points. Since there are six simplest interpolation points, we may choose 5 points out of them to construct an optimal algorithm. In [Bla84], we can find a radix-3 complex polynomial product algorithm. It uses  $\{0,1,-1,j,-j\}$  as interpolation points, leading to a (5 cmults, 15 cadds) algorithm. After overlap, it becomes an F(3,3) algorithm with (5 cmults, 17 cadds) or (15 mults, 51 adds).

However by using  $\{0,1,j,-j,\infty\}$  as interpolation points, we can obtain a better algorithm with less additions:

(a1) complex F(3,3) algorithm with (5 cmults, 15 cadds) or (15 mults, 45 adds)

$a_0=x_0$	$b_0=h_0$
$a_1=x_2$	$b_1=h_2$
$a_2=x_0+x_1+x_2$	$b_2=h_0+h_1+h_2$
$a_3=(x_0-x_2)+jx_1$	$b_3=h_0-h_2+jh_1$
$a_4=(x_0-x_2)-jx_1$	$b_4=h_0-h_2-jh_1$
$m_i=a_i \cdot b_i ; i=0,1,2,3,4$	
$u_0=m_0+m_1$	
$u_1=m_3+m_4$	
$u_2=m_3-m_4$	
$u_3=-u_0+m_2$	
$y_0=m_0+(u_3+u_1) z^{-3}$	
$y_1=u_3-ju_2+m_1z^{-3}$	
$y_2=u_0+ju_2-u_1$	

For an optimal  $F(N,N)$  algorithm, we need  $(2N-1)$  interpolation points. As  $N \geq 4$  more than seven interpolation points are required. The six previous points are no longer sufficient. Which points should be chosen next? It seems  $\{a_i\} = \{\pm 1 \pm j\}$  are good candidates, since  $a_1^2 = \pm 2j$  and  $a_1^4 = -4$ . Only a few more additions are required with the increase of the exponent.

An  $F(5,5)$  algorithm using  $\{0, \pm 1, \pm j, \pm 1 \pm j\}$  as interpolation points is given in the appendix. It requires (9 cmults, 77 cadds) or (27 mults, 181 adds). Although the  $F(5,5)$  is optimal in terms of multiplication count, it requires a great deal of additions. This increase in the number of additions may not justify the minimization of the number of multiplications in certain implementations where an addition and a multiplication require almost the same time.

Then it is desirable to construct some suboptimal algorithms which require more than  $2N-1$  multiplications but substantially less additions. An suboptimal  $F(4,4)$  algorithm can be derived by twice applying the eq.(3.18), leading to a (9 cmults, 20 cadds) or (27 mults, 67 adds) algorithm. Another way of obtaining suboptimal algorithm is proposed in [Bal86]. In their method, the derivation of the polynomial  $Q(z)$  at points  $\{0, \infty\}$  are used in order to use less interpolation points. For example, an  $F(4,4)$  algorithm can be constructed using the interpolation values at  $\{0, \pm 1, \pm j, \infty\}$  and the derivation at  $\{0\}$ . The last

value ( $=x_0h_1+x_1h_0$ ) requires two multiplications instead of one multiplication per interpolation points. But the number of additions can be reduced. We give this algorithm as follows:

(a2) complex  $F(4,4)$  algorithm with (8 cmults, 23 cadds) or (24 mults, 69 adds)

$a_0=x_0$	$b_0=h_0$
$a_1=x_0$	$b_1=h_1$
$a_2=x_1$	$b_2=h_0$
$a_3=x_3$	$b_3=h_3$
$a_4=(x_0+x_2)+(x_1+x_3)$	$b_4=(h_0+h_2+h_1+h_3)/4$
$a_5=(x_0+x_2)-(x_1+x_3)$	$b_5=(h_0+h_2-h_1-h_3)/4$
$a_6=(x_0-x_2)+(x_1-x_3)$	$b_6=(h_0-h_2+h_1-h_3)/4$
$a_7=(x_0-x_2)-(x_1-x_3)$	$b_7=(h_0-h_2-h_1+h_3)/4$

$m_i=a_i \cdot b_i; i=0,1,\dots,8$

$u_0=m_4+m_5$   
 $u_1=m_4-m_5$   
 $u_2=m_6+m_7$   
 $u_3=m_6-m_7$   
 $p_0=u_0+u_2$   
 $p_1=u_1+u_3$   
 $p_2=u_0-u_2$   
 $p_3=u_1-u_3$   
 $p_4=m_1+m_2$

$y_0=m_0+(p_0-m_0)z^{-4}$   
 $y_1=p_4+(p_1-p_4)z^{-4}$   
 $y_2=(p_2-m_3)+m_3z^{-4}$   
 $y_3=p_3$

In eq.(3.22) a length- $L$  complex filter is transformed into 3 length- $L$  real FIR filters instead of 4 by direct computation. Although fast real short length FIR filtering algorithms can be applied to compute the three length- $L$  real FIR filters, we will show that deriving algorithms at first in complex domain results in more efficient ones.

Let us applying eq.(3.18) to complex filters. Then a length-L complex FIR filter can be computed by three length-L/2 complex FIR filters. Further applying eq.(3.22), we replace each of the three complex filters by three real ones of the same length. The resulting complexity is:

$$\begin{aligned} & 4 \text{ cadds} + 3 ( 3 \text{ adds} + 3 \text{ length-L/2 real filters) per two complex outputs} \\ & = 17 \text{ adds} + 9 \text{ length-L/2 real filters per two complex outputs} \end{aligned}$$

If we first apply eq.(3.22) then eq.(3.18), the complexity will be:

$$\begin{aligned} & 3 \times 2 \text{ adds} + 3 ( 4 \text{ adds} + 3 \text{ length-L/2 real filters) per two complex outputs} \\ & = 18 \text{ adds} + 9 \text{ length-L/2 real filters per two complex outputs} \end{aligned}$$

The first algorithm requires less additions.

The above discussion is in the case  $N=2$ . For higher  $N$  it is still more recommended to apply short length complex FIR filtering algorithms first and eq.(3.22) then, since there are more simple interpolation points in the complex domain than in the real one.

We may apply the  $F(2,2)$  and  $F(4,4)$  algorithm to the scheme b) and c) of computing the length- $2^m$  FIR filter. The  $F(2,2)$  algorithm needs to consider two outputs together in the frequency domain so that  $2^m$  outputs of the filter should be computed together. By analogy, applying  $F(4,4)$  algorithm needs to compute  $2^m$  outputs of the filter together. The resulting arithmetic complexities are:

IIIId) computing the length- $2^m$  real FIR filter using  $F(2,2)$  algorithm in frequency domain (per  $2^m$  outputs)

$$M_r = 2^m(2m-1.5)+5$$

$$A_r = 2^m(6m-2.5)+3$$

the arithmetic complexity per output

$$m_r = 2m-1.5+5 \cdot 2^{-m}$$

$$a_r = 6m-2.5+3 \cdot 2^{-m}$$

$$m_r + a_r = 8m-4+8 \cdot 2^{-m}$$

To our surprise, this scheme results in reductions both in the number of multiplications and in the number of additions for  $m \geq 2$ , compared to the conventional scheme a).

IIIe) computing the length- $2^m$  real FIR filter using  $F(4,4)$  algorithm in frequency domain (per  $2^m$  outputs)

$$M_r = 2^m(2m-2)+10$$

$$A_r = 2^m(6m+2.75)+1$$

the arithmetic complexity per output

$$m_r = 2m-2+10 \cdot 2^{-m}$$

$$a_r = 6m + 2.75 + 2^{-m}$$

$$m_r + a_r = 8m + 0.75 + 11 \cdot 2^{-m}$$

Although this algorithm results in a reduction in the number of multiplications, but it requires more additions and computes a block of  $2^m$  outputs together. Hence this scheme is less of interest than the others. It indicates also that the application of short length complex FIR filtering algorithms in such schemes should be limited. Direct computation of the filters in frequency domain may meet better trade-off between arithmetic complexity and structural regularity.

### 3.5 Remarks

In this chapter, we have presented different fast FIR filtering algorithms, classical ones as well as new structures, in the context of the approach in chapter 2. Most algorithms belong to the class of overlap-add scheme. By transposing the overlap-add scheme, we get another class of algorithms called overlap-save. Winograd algorithms look much different from the others, since they are *a priori* overlap-save algorithms. However they are basically the transposes of the Agarwal&Burrus algorithms.

The approach allows us to establish all the intermediates between direct FIR computation and the usual fast algorithms based on cyclic convolution of large blocks. Shorter FFT-based algorithms are developed. They represent a class of algorithms taking into account the arithmetic complexity,

structural regularity and system delay. Short length complex FIR filtering algorithms are studied. Combining the last two classes of algorithms, we have got some algorithms which have less arithmetic complexity than the well-known FFT-based ones, which are thought to be the most efficient for large block processing.

The fast aperiodic and cyclic convolution algorithms play an important role in fast FIR filtering. But they are not equivalent to FIR filtering computation. The other aspects should be equally considered, since the choice of  $N$  in the formulation, the overlapping techniques have important influences on the structure of realization.

### Appendix

Complex  $F(5,5)$  algorithm using  $\{0, \pm 1 \pm j, \pm 1 \pm j\}$  as interpolation points with (9 cmults, 77 cadds) or (27 mults, 181 adds).

```

c0=x0+x4
c1=c0+x2
c2=c0-x2
c3=x1+x3
c4=x1-x3
a0=c1+c3          b0=(h0+h4+h2+h1+h3)/20
a1=c2-jc4         b1=[h0+h4-h2-j(h1-h3)]/20
a2=c1-c3          b2=(h0+h4+h2-h1-h3)/20
a3=c2+jc4         b3=[h0+h4-h2+j(h1-h3)]/20
c5=x4+x4
c6=c5+c5
c7=x0-c6
c8=x1+jx1
c9=c7+c8
c10=c7-c8
c11=c7-jc8
c12=c7+jc8
c13=x3+jx3
c14=x2+c13
c15=x2-c13
c16=x2-jc13

```

$c17=x2+jc13$	
$c18=c14+c14$	
$c19=c15+c15$	
$c20=c16+c16$	
$c21=c17+c17$	
$a4=c9+jc18$	$b4=\{(h0-4h4)+x1(1+j)+2j[x2+x3(1+j)]\} / [40(j-1)]$
$a5=c10+jc19$	$b5=\{(h0-4h4)-x1(1+j)+2j[x2-x3(1+j)]\} / [40(1-j)]$
$a6=c11+jc20$	$b6=\{(h0-4h4)-jx1(1+j)+2j[x2-jx3(1+j)]\} / [40j(1-j)]$
$a7=c12+jc21$	$b7=\{(h0-4h4)+jx1(1+j)+2j[x2+jx3(1+j)]\} / [40j(j-1)]$
$a8=x4$	$b8=h4/5$

$mi=ai \ bi ; \quad i=0,1,\dots,8$

$u0=m0+m1$	$u1=m2+m3$
$u2=m0-m1$	$u3=m2-m3$
$p0=u0+u1$	$p1=u2+u3$
$p2=u0-u1$	$p3=u2-u3$
$u4=m4-m5$	$u5=m6-m7$
$u6=m4+m5$	$u7=m6+m7$
$u8=u4-ju5$	$u9=u6-u7$
$u10=u4+ju5$	$u11=ju8-u8$
$q0=u11+u11$	$q1=j(u9+u9)$
$q2=u10+ju10$	$q3=u6+u7$
$v1=m8+m8$	$v2=v1+v1$
$v3=v2+v2$	$v4=v3+v3$
$s0=p0-m8$	$s1=s0+s0$
$s2=s1+s1$	$s3=q1+q1$
$s4=s3+s3$	$s5=q2+q2$
$s6=s5+s5$	$s7=q3+q3$
$s8=s7+s7$	$t0=q0-v4$
$t1=s2+t0$	$t2=s4+q1$
$t3=s6+q2$	$t4=s8+q3$
$t5=s2-t0$	$t6=s4-q1$
$t7=s6-q2$	$t8=s8-q3$
$y0=t1+t6 \ z^{-5}$	$y1=t2+t7 \ z^{-5}$
$y2=t3+t8 \ z^{-5}$	$y3=t4+(v2+m8) \ z^{-5}$
$y4=t5$	



## Chapter 4

### Short length FIR filtering algorithms

#### 4.1 Introduction

As the classical algorithms are structurally too complex for implementation on the Digital Signal Processor, new algorithms are needed to take into account the structure of the processor. This motivated us to present a new class of algorithms. This chapter deals with the short length FIR filtering algorithms (SLFIR) in a detailed manner. These algorithms compute together a few number of filter outputs, allowing reducing the arithmetic complexity while maintaining smaller FIR filters as computing kernels. Hence, they are open to a wide range of implementations, including Digital Signal Processors (DSP's), VLSI, and even general purpose computers. The inherent parallelism of the algorithms allows also multiprocessor implementation. General rules are provided for combining several SLFIR algorithms to construct composite length FIR filtering algorithms. Their arithmetic complexities are evaluated and compared to FFT-based algorithms under various criteria.

This chapter is based on two articles of the author (joint with P.Duhamel) [Mou87,Mou89]. The first one shows a simple example, giving insight to the algorithm derivation both in time domain and in z-transform domain. The second one presents comprehensively the short length FIR filtering algorithms. Diverse aspects are discussed such as the derivation, higher-radix algorithms, composite length algorithms, performance comparison with FFT-based algorithms. Several algorithms are given.

A fixed-point error analysis is made for the short length FIR filtering algorithms in the appendix.

## 4.2 Article 1

## FAST FIR FILTERING: ALGORITHMS AND IMPLEMENTATIONS

Z.J. MOU and P. DUHAMEL

CNET/PAB/RPE, 38-40, rue du Général Leclerc, 92131 Issy-les-Moulineaux, France

Received 16 February 1987

Revised 25 May 1987

**Abstract.** We first establish through a simple example a new fast FIR filtering algorithm based on a divide-and-conquer approach. This algorithm does not require the use of overlap techniques as is usual in the approaches based on cyclic or aperiodic convolutions. We outline the advantages of the proposed algorithm when implemented both in software and in hardware. Finally, we give a systematic way of deriving these algorithms.

**Zusammenfassung.** Anhand eines Beispiels wird zunächst einer schneller Algorithmus für die nichtrekursive Filterung vorgestellt, der auf einer gemeinsamen Auswertung einer zuvor aufgeteilten Berechnung beruht. Dieser Algorithmus benötigt keine Signalüberlappungs-Techniken wie die Methoden, die eine zyklische oder azyklische Faltung für die lineare Filterung ausnützen. Anschließend wird auf die wesentlichen Vorteile der Algorithmus bei einer Hardware- oder Software-Realisierung eingegangen. Schließlich wird angegeben, wie derartige Algorithmen systematisch hergeleitet werden können.

**Résumé.** Tout d'abord, nous montrons, à l'aide d'un exemple, l'existence d'algorithmes rapides de filtrage nonrecursif basés sur une approche de division et évaluation. Cet algorithme ne nécessite pas l'usage de techniques de 'recouvrement', comme c'est le cas dans les approches à base de convolution cyclique ou aperiodique. Puis, nous montrons les avantages essentiels de cet algorithme, selon qu'il est implanté en logiciel ou en matériel. Enfin, nous donnons une méthode permettant de dériver systématiquement les algorithmes de cette classe.

**Keywords.** Fast algorithm, FIR filtering, arithmetic complexity.

## 1. Introduction

Our aim, in this paper, is to provide fast algorithms for the direct computation of the output of a length- $N$  digital filter:

$$y_n = \sum_{i=0}^{N-1} x_{n-i} h_i, \quad n = 0, 1, 2, \dots \quad (1)$$

Most of the fast algorithms used for computing (1) are based on fast transforms. This implies the use of the cyclic convolution (2) as an intermediate step,

$$y'_n = \sum_{i=0}^{N'-1} x_{(n-i)_{N'}} h'_i, \quad n = 0, 1, 2, \dots, N'-1, \quad (2)$$

where  $(n-i)_{N'} = (n-i) \bmod N'$ . Here,  $N'$  has to be chosen greater than  $N$ , and  $\{x_i\}$  and  $\{h_i\}$  have to be extended up to  $N'$ . The  $\{y_n\}$  are reconstructed from  $\{y'_n\}$  (overlap-add or overlap-save technique). This implies an overall organization of the fast algorithm which is rather intricate and expensive in terms of memory and communication cost, even if the number of arithmetic operations to be performed per output has diminished [3, 7].

Some work was also performed on aperiodic convolution algorithms, thus allowing the computation of (1) with the aperiodic convolution (3) as an intermediate step [1, 2, 8],

$$y_n^* = \sum_{i=0}^{N-1} x_{n-i} h_i, \quad n = 0, 1, 2, \dots, M-1, \quad M \geq N, \tag{3}$$

but these algorithms still require the overlap-add or overlap-save technique to reconstruct the FIR filter output from the  $\{y_n^*\}$ . Unfortunately, the structure of these algorithms is weaker than that of cyclic-convolution algorithms. Therefore, they are not widely used.

Both kinds of approaches can be named indirect ones. Furthermore, they completely lose the FIR filtering structure, which is very easy to implement in hardware or in software (Digital Signal Processors—DSPs—are generally optimized for FIR filtering structures). Nevertheless, a lot of work has been done to reduce the computational cost of these cyclic and aperiodic convolution algorithms [3, 7].

But very little work has been done directly on the FIR filtering equation (1), thus taking into account the infinite length of both input and output sequences.

Winograd [10] has done pioneering work in FIR filtering algorithms. His research was undertaken to establish low-order FIR filtering algorithms, and then to extend them by tensor product. The methodology he proposed seems to be a little far from practical use, because low-order FIR filters are not common in practice, and because the way Winograd proposed to extend them using tensor products still requires large memory for large  $N$ . Nevertheless, most of the results explained in the following are based on this pioneering work, but extend its practical usefulness and provide systematic methods to establish the new algorithms.

Here, we propose new fast FIR filtering algorithms, which will be shown to be of interest both in software and in hardware implementations. In software, we build well-structured algorithms reducing the number of arithmetic operations. In hardware, we propose FIR filter structures with higher speed and/or lower cost than the usual ones, with a special mention to distributed arithmetic implementations for which this approach seems to be particularly well suited.

Finally, as a conclusion, we outline that these preliminary results can be extended in many ways.

2. A simple case

2.1. Derivation of the algorithm

Equation (1) can be written in scalar product form as

$$y_n = (x_n \ x_{n-1} \ \cdots \ x_{n-N+1}) \begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_{N-1} \end{pmatrix} \tag{4}$$

and, if two outputs  $y_{n-1}$  and  $y_n$  are to be computed together, we can write, in matrix form,

$$\begin{pmatrix} y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} x_{n-1} & x_{n-2} & \cdots & x_{n-N} \\ x_n & x_{n-1} & \cdots & x_{n-N+1} \end{pmatrix} \begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_{N-1} \end{pmatrix}. \tag{5}$$

Let us suppose  $N$  to be even, and group the even and odd coefficients of  $\{h_i, i = 1, 2, \dots, N - 1\}$  as follows,

$$\begin{pmatrix} y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} x_{n-1} & x_{n-3} & \cdots & x_{n-N+1} & x_{n-2} & x_{n-4} & \cdots & x_{n-N} \\ x_n & x_{n-2} & \cdots & x_{n-N+2} & x_{n-1} & x_{n-3} & \cdots & x_{n-N+1} \end{pmatrix} \begin{pmatrix} h_0 \\ h_2 \\ \vdots \\ h_{N-2} \\ h_1 \\ h_3 \\ \vdots \\ h_{N-1} \end{pmatrix}. \tag{6}$$

Let us define

$$A = (x_{n-1}, x_{n-3}, \dots, x_{n-N+1}), \quad B = (x_{n-2}, x_{n-4}, \dots, x_{n-N}), \quad C = (x_n, x_{n-2}, \dots, x_{n-N+2}),$$
$$H_0 = (h_0, h_2, \dots, h_{N-2})^T, \quad H_1 = (h_1, h_3, \dots, h_{N-1})^T.$$

Equation (6) can now be rewritten as

$$\begin{pmatrix} y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} A & B \\ C & A \end{pmatrix} \begin{pmatrix} H_0 \\ H_1 \end{pmatrix}. \tag{7}$$

Thus, we make apparent the redundancy between the computation of the two outputs  $y_{n-1}$  and  $y_n$ .

A straightforward computation of (7) would require approximately the same computational load as four filters of length  $\frac{1}{2}N$ . But the reader familiar with the fast algorithms (or the usual 3 multi-3 adds complex multiplication algorithm) will soon realize that it is possible to obtain  $y_{n-1}$  and  $y_n$  by the following formula:

$$\begin{pmatrix} y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} A(H_0 + H_1) + (B - A)H_1 \\ A(H_0 + H_1) - (A - C)H_0 \end{pmatrix}. \tag{8}$$

The filtering operation  $A(H_0 + H_1)$  is now common between the two terms, and the overall computational load is now approximately that of three filters of length  $\frac{1}{2}N$ , plus the linear combination  $(B - A)$  and  $(A - C)$  (the combination  $H_0 + H_1$  is precomputed for given coefficients):

$$B - A = (x_{n-2} - x_{n-1}, x_{n-4} - x_{n-3}, \dots, x_{n-N} - x_{n-N+1}),$$
$$A - C = (x_{n-1} - x_n, x_{n-3} - x_{n-2}, \dots, x_{n-N+1} - x_{n-N+2}). \tag{9}$$

Equation (9) seems to require a lot of additions. But, if we remember that the previous set of two outputs  $(y_{n-2}, y_{n-3})$  already required nearly the same operations, we see that, for each output set  $(y_n, y_{n-1})$ , only two new input additions are to be computed:  $(x_{n-2} - x_{n-1}, x_{n-1} - x_n)$ . The other elements of  $(B - A)$  and  $(A - C)$  are already stored in the FIR filtering processes of length  $\frac{1}{2}N$ . This results in the diagram of Fig. 1.

In obtaining the above algorithm, we have explicitly taken advantage of computing several outputs at a time to improve the compromise speed/complexity of an FIR filter, while retaining most of its structure, which allows to retain information from the previous computations to obtain the new outputs.

Further decompositions are of course still feasible.

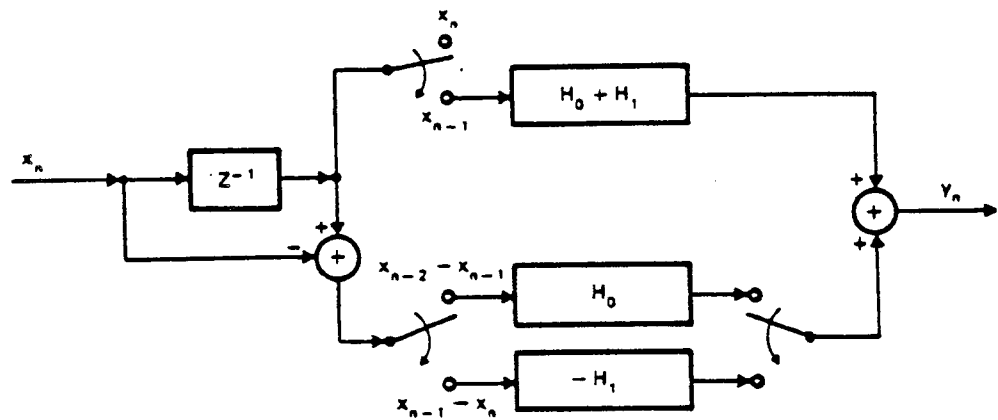


Fig. 1. An FIR filter with reduced arithmetic complexity.

2.2. Arithmetic complexity

The number of operations to be usually performed per output for an  $N$ -tap digital filter is

$N - 1$  adds,  $N$  mults per output. (10)

The proposed algorithm requires two input additions, two output additions, and three  $\frac{1}{2}N$ -tap digital filters to compute two outputs, i.e.,

$2 + \frac{3}{2}(N - 1)$  adds,  $\frac{3}{2}N$  mults per output, (11)

which means that both numbers of additions and multiplications have been reduced (about 25% improvement).

If  $k$  successive decompositions are performed (i.e.,  $2^k$  outputs are computed together), then the number of arithmetic operations to be performed per output is

$4[(\frac{3}{2})^k - 1] + (\frac{3}{2})^k(N/2^k - 1)$  adds,  $(\frac{3}{2})^k N/2^k$  mults per output. (12)

2.3. Implementation

When implemented on a DSP, this algorithm will be efficient as long as the overhead, due to the input/output additions ( $4[(\frac{3}{2})^k - 1]$  in equation (12)) and the storage time will be shorter than the improvement in the number of multiply-accumulates. This means that, for large  $N$ , the first decomposition will certainly be of interest, while the efficiency of the following ones remains to be checked.

It is also seen that the general scheme of Fig. 1 is well suited for multiprocessor implementation: compared to [6], implementation of the above algorithm would have saved a lot of hardware.

It can easily be seen from Fig. 1 that the sampling rate of the inputs of the three  $\frac{1}{2}N$ -tap filters is divided by two compared to the initial one: we use three low-speed  $\frac{1}{2}N$ -tap filters to build an  $N$ -tap filter with higher sampling rate.

This can be useful in any hardware implementation: if we have at our disposal a kind of filter of length  $N$  with maximal processing speed  $f_m$ , three such filters allow to construct a filter of length  $2N$  and maximal processing speed  $2f_m$ , at the cost of four input/output additions.

There is a case where this approach is specially well adapted: the implementation of digital filters by distributed arithmetic [4, 9]. In fact, it is already usual to break the initial filter into subfilters, connected

by additions, to reduce the size of the ROMs involved. The proposed algorithm can hence be used to increase the throughput of the distributed arithmetic filter, while reducing the number of ROMs involved. All these implementations are under consideration. Results will be reported on in a subsequent paper.

3. Systematic derivation of fast FIR filtering algorithms

The matrix formulation (5) → (8) used to derive the simple example of Section 2 may be direct and easy to understand, but the following formulation in the  $z$ -domain allows us to obtain strong support from a wealthy bank of algorithms: aperiodic convolution algorithms or polynomial product algorithms [1, 2, 3, 7, 8, 10]. It will provide a systematic way of deriving these fast FIR filtering algorithms.

Let us decimate the output, input, and filter coefficients of (1) by 2. (We shall thus derive one step of a 'radix-2' decimation-in-time algorithm.) In terms of  $z$ -transform, we have

$$\begin{aligned} Y_0(z) &= \sum_{i=0}^{\infty} y_{2i} z^{-i}, & Y_1(z) &= \sum_{i=0}^{\infty} y_{2i+1} z^{-i}, \\ X_0(z) &= \sum_{i=0}^{\infty} x_{2i} z^{-i}, & X_1(z) &= \sum_{i=0}^{\infty} x_{2i+1} z^{-i}, \\ H_0(z) &= \sum_{i=0}^{N/2-1} h_{2i} z^{-i}, & H_1(z) &= \sum_{i=0}^{N/2-1} h_{2i+1} z^{-i}, \end{aligned} \tag{13}$$

and the  $z$ -transform of (1), given in equation (14), can be rewritten as given in (15) or (16):

$$Y(z) = X(z) \cdot H(z), \tag{14}$$

$$Y_0(z^2) + z^{-1} Y_1(z^2) = [X_0(z^2) + z^{-1} X_1(z^2)][H_0(z^2) + z^{-1} H_1(z^2)], \tag{15}$$

$$\begin{aligned} Y_0(z^2) &= X_0(z^2) H_0(z^2) + z^{-2} X_1(z^2) H_1(z^2), \\ Y_1(z^2) &= X_1(z^2) H_0(z^2) + X_0(z^2) H_1(z^2). \end{aligned} \tag{16}$$

Equations (15) and (16) can be understood in different ways: equation (15) is in the form of a polynomial product (or equivalently of an aperiodic convolution), the coefficients of which are filters. A direct implementation is given in Fig. 2.

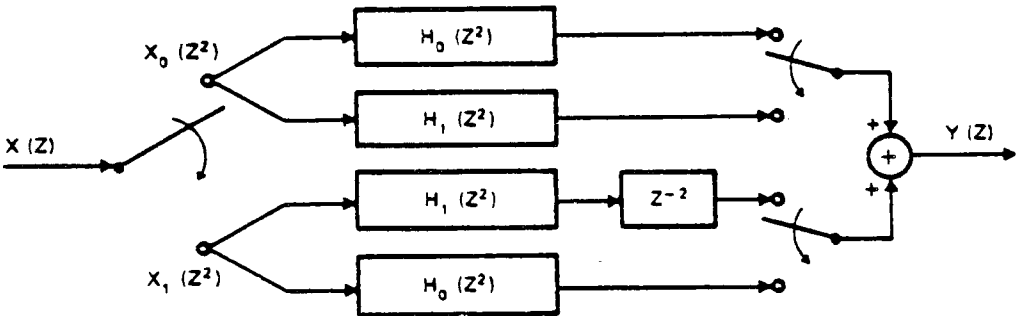


Fig. 2. Direct implementation of the decimated filtering equation.

But, if we define

$$\begin{aligned} y_0 &= Y_0(z^2), & y_1 &= Y_1(z^2), \\ a_0 &= X_0(z^2), & a_1 &= X_1(z^2), \\ h_0 &= H_0(z^2), & h_1 &= H_1(z^2), \end{aligned} \quad (17)$$

then we can apply the usual aperiodic convolution algorithm, as given in [7]:

$$m_1 = a_0 h_0, \quad m_2 = (a_0 + a_1)(h_0 + h_1), \quad m_3 = a_1 h_1, \quad (18a, b, c)$$

to obtain

$$y_0 + z^{-1} y_1 = (m_1 + z^{-2} m_3) + z^{-1} (m_2 - m_1 - m_3). \quad (19)$$

All the 'multiplications'  $m_i$  involved in (19) now represent filtering equations. Coming back to the initial notation, one gets

$$\begin{aligned} Y_0(z^2) &= X_0(z^2) H_0(z^2) + z^{-2} X_1(z^2) H_1(z^2), \\ Y_1(z^2) &= [X_0(z^2) + X_1(z^2)] [H_0(z^2) + H_1(z^2)] - X_0(z^2) H_0(z^2) - X_1(z^2) H_1(z^2), \end{aligned} \quad (20)$$

resulting in the diagram of Fig. 3.

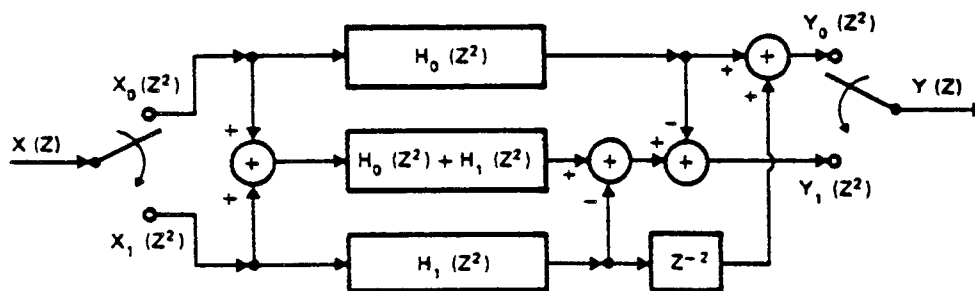


Fig. 3. The implementation obtained by directly applying a length-2 aperiodic convolution algorithm.

Another possibility is to take equation (16) as a starting point and, after defining

$$a_0 = z^{-2} X_1(z^2), \quad a_1 = X_0(z^2), \quad a_2 = X_1(z^2), \quad (21)$$

one can easily realise that equation (16) is in the form of an FIR filter of length 2, two outputs of which are computed:

$$y_0 = a_1 h_0 + a_0 h_1, \quad y_1 = a_2 h_0 + a_1 h_1. \quad (22)$$

But, it was shown by Winograd [10] that FIR filtering algorithms could be obtained by the so-called 'transposition' of polynomial products. By transposing (19), we get the following algorithm to compute (22):

$$y_0 = a_1 (h_0 + h_1) + (a_0 - a_1) h_1, \quad y_1 = a_1 (h_0 + h_1) - (a_1 - a_2) h_0, \quad (23)$$

which, back to the initial terminology, turns out to be

$$\begin{aligned} Y_0(z^2) &= X_0(z^2) [H_0(z^2) + H_1(z^2)] + [z^{-2} X_1(z^2) - X_0(z^2)] H_1(z^2), \\ Y_1(z^2) &= X_0(z^2) [H_0(z^2) + H_1(z^2)] - [X_0(z^2) - X_1(z^2)] H_0(z^2), \end{aligned} \quad (24)$$

leading to the diagram of Fig. 4.

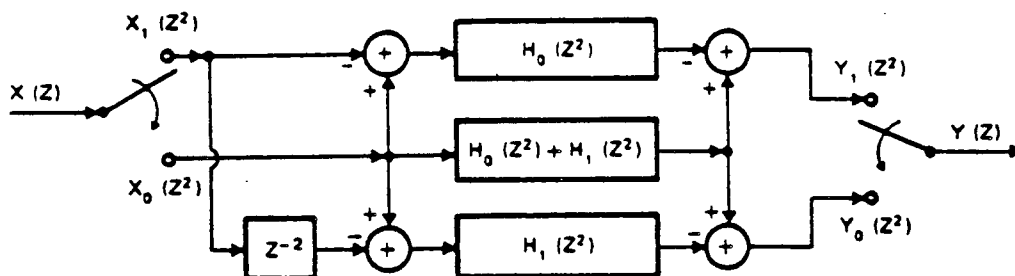


Fig. 4. The implementation obtained by applying a 2-tap filtering algorithm.

From the point of view of a network, the scheme of Fig. 4 is exactly the transposition of the one of Fig. 3 (see [5] for details on network transposition). Therefore, we can get new filtering schemes by either algorithm transposition or network transposition.

Both (20) and (24) are equally efficient in terms of arithmetic operation count, but their implementation is different.

But, what is the relationship with the algorithm explained in Section 2?

Careful examination of equations (8) and (24) shows that (24) is the  $z$ -domain representation of (8), and that the scheme of Fig. 4 can be modified into the form of Fig. 1 if we change the order of switching and adding: they are exactly equivalent. However, we have not exhausted all the possible ways to implement radix-2 algorithms: a systematic research gives sixteen different ones.

This approach can of course be generalized to higher decimation ratios. It is also possible to use different decimation ratios on output than on input, at the cost of a slightly more involved algorithm.

We have thus demonstrated that most of the work done on the aperiodic convolution algorithms directly applies to fast FIR filtering without the need of overlap-add or overlap-save techniques, thus saving memory in the implementation, and keeping the algorithm's simplicity.

Furthermore, since, with the proposed approach, most of the multiply-accumulate structure of the FIR filter is retained, it allows to choose the best tradeoff between structural and arithmetic complexity, for a given type of implementation.

#### 4. Conclusion

We have presented FIR filtering algorithms for software and hardware implementations. These algorithms are shown to have a regular structure, retaining the multiply-accumulate structure of the FIR filter. This is why we believe it possible to improve the compromise speed/complexity in any case of implementation: FIR filtering on DSPs, on general purpose computers, distributed arithmetic, and so on. Further work will be reported on in future.

The theory explained in Section 3 also allows to derive higher radix algorithms and mixed radix algorithms, and can easily be generalized to multi-dimensional FIR filtering.

#### References

- [1] R.C. Agarwal and C.S. Burrus, "Fast one-dimensional digital convolution by multidimensional techniques", *IEEE Trans. Acoust., Speech, Signal Process.*, Vol. 22, February 1974, pp. 1-10.
- [2] R.C. Agarwal and J.W. Cooley, "New algorithms for digital convolution", *IEEE Trans. Acoust., Speech, Signal Process.*, Vol. ASSP-25, October 1977, pp. 392-410.



- [3] R.E. Blahut, *Fast Algorithms for Digital Signal Processing*, Addison-Wesley, Reading, MA, 1985.
- [4] C.S. Burrus, "Digital filter structures described by distributed arithmetic", *IEEE Trans. Circuits & Systems*, Vol. CAS-24, December 1977, pp. 674-680.
- [5] R.E. Crochiere and L.R. Rabiner, *Multirate Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1983.
- [6] K. Hayashi, K.K. Dhan, K. Sugahara and K. Hirano, "Design of high-speed digital filters suitable for multi-DSP implementation", *IEEE Trans. Circuits & Systems*, Vol. CAS-33, February 1986, pp. 202-216.
- [7] H.J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*, Springer, Berlin/New York, 1981.
- [8] P.C. Palla, A. Antoniou and S.D. Morgera, "Higher radix aperiodic convolution algorithms", *IEEE Trans. Acoust., Speech, Signal Process.*, Vol. ASSP-34, February 1986, pp. 60-68.
- [9] A. Peled and B. Liu, "A new hardware realization of digital filters", *IEEE Trans. Acoust., Speech, Signal Process.*, Vol. ASSP-22, December 1974, pp. 456-462.
- [10] S. Winograd, "Arithmetic complexity of computations", *CBMS-NSF Regional Conf. Series in Applied Mathematics*, Siam Publications No. 33, 1980.

### 4.3 Article 2

#### **Short length FIR filters and their use in fast non recursive filtering**

(Submitted to IEEE Transactions on ASSP, Mai 1989)

*Abstract* -This paper provides the basic tools required for an efficient use of the recently proposed fast FIR algorithms. These algorithms allow not only to reduce the arithmetic complexity but also maintain partially the multiply-accumulate structure, thus resulting in efficient implementations.

A set of basic algorithms is derived, together with some rules for combining them. Their efficiency is compared with that of classical schemes in the case of three different criteria, corresponding to various types of implementation. It is shown that this class of algorithms (which includes classical ones as special cases) allows to find the best tradeoff corresponding to any criterion.

1. Introduction
2. General description of the algorithm
3. Short-length FIR filtering algorithms
4. Composite length algorithms
5. Conclusion

## 1. Introduction

A lot of algorithms are known to reduce the arithmetic complexity of FIR filtering. The widely used ones are indirect algorithms, based either on the cyclic convolution or on the aperiodic convolution using fast transforms as an intermediate step. Direct methods without transforms were also proposed by Winograd [1].

Both direct and indirect methods require large block processing: they make use of the redundancy between at least  $L$  successive output computations ( $L$  is the length of the filter) to reduce the number of operations to be performed per output point.

Furthermore, the structure of the resulting algorithm has completely changed : the initial computation is mainly based on a multiply-accumulate (MAC) structure, while the fast algorithms always involve global exchange of data inside a large vector of size at least  $2L$ .

These are the main reasons why the above fast algorithms are not of wide interest for real-time filtering : Hardware implementations require pipelining the whole system with many intermediate memories, which results in a large amount of hardware. On another side, software implementations on Digital Signal Processors (DSP's) are not very efficient, except for very large  $L$ , since those fast algorithms have lost the multiply-accumulate structure for which all DSP's are optimized.

In other words, the usefulness of such algorithms was diminished because the reduction in arithmetic requirements per output point was obtained at the expense of a loss of structural regularity.

But structural regularity is difficult to quantify : Hardware implementations do not require the same kind of regularity as VLSI implementations do and "structural regularity" is still another matter when thinking of DSP implementations.

Anyway, one fact remains: MAC structure is very efficient on any type of implementation, including those on general purpose computer.

Recently, a new class of fast FIR filtering algorithms taking these considerations into account was proposed [2,3,4]: These algorithms retain partially the FIR filter structure, while reducing the arithmetic complexity. They allow various tradeoffs between structural regularity and arithmetic efficiency, including all classical schemes as special

cases [10]. This flexibility in the derivation of the algorithms allows to find the best possible solution in any type of implementation.

The purpose of this paper is to provide the basic tools required for the derivation of algorithms meeting various tradeoffs in different implementations.

A brief description of these new algorithms is provided in Section 2. This description allows to understand the structure of the new algorithms: Short-length FIR filters with reduced arithmetic complexity where all multiplications are replaced by decimated subfilters. Since the process can be reiterated on the subfilters, the short-length filters are recognized to be the basic building tools of these fast algorithms.

Hence, Section 3 is concerned with the derivation of a set of algorithms. This section is mostly based on Winograd's work. We bring some improvements in the number of additions by recognizing that the FIR filtering, seen as a running process, involves a pseudocirculant matrix [5] instead of a general Toeplitz one. Another advantage of this presentation is the easy understanding of the transposition principle in the context of multi-input multi-output systems, overlapping between blocks being naturally taken into account. Using this pseudocirculant presentation, we can derive the transposed version of all fast FIR filtering algorithms in a very easy manner.

Section 4 addresses the case of multifactor algorithms. Iterating the basic process raises the question of the best ordering of the short modules and of the length where the decomposition has to be stopped. We provide the rules for obtaining the ordering of factors which results in the lowest arithmetic complexity. A comparison with classical algorithms (FFT-based ones) is also provided in the case of real valued signals.

Section 5 concludes and explains some open problems.

## **2. General description of the algorithm**

Let us consider the filtering of a sequence  $\{x_i\}$  by a length-L FIR filter with fixed coefficients  $\{h_i\}$  :

(1)

$$y_n = \sum_{i=0}^{L-1} x_{n-i} h_i \quad n = 0, 1, 2, \dots, \infty$$

In z-domain formulation, this convolution becomes a polynomial product :

(2)

$$Y(z) = H(z) X(z)$$

Where X and Y are of infinite degree while H(z) has degree L-1: In z domain, the filtering equation, seen as a running process, is described by the product of an infinite degree polynomial and a finite degree one.

Let us now decimate each of the three terms in eq.(2) into N interleaved sequences:

(3)

$$H_j(z) = \sum_{m=0}^{LN-1} h_{mN+j} z^{-m} \quad ; j = 0, 1, \dots, N-1$$

$$X_k(z) = \sum_{m=0}^{\infty} x_{mN+k} z^{-m} \quad ; k = 0, 1, \dots, N-1$$

$$Y_i(z) = \sum_{m=0}^{\infty} y_{mN+i} z^{-m} \quad ; i = 0, 1, \dots, N-1$$

Eq.(2) then becomes :

(4)

$$\sum_{i=0}^{N-1} Y_i(z^N) z^{-i} = \sum_{j=0}^{N-1} H_j(z^N) z^{-j} \sum_{k=0}^{N-1} X_k(z^N) z^{-k}$$

Eq.(4) is in the form of a polynomial product or an aperiodic convolution. The two polynomials to be multiplied have finite degree N-1, and their coefficients are themselves polynomials, either of finite degree, such as  $\{H_i\}$ , or of infinite degree, such as  $\{X_i\}$  or  $\{Y_i\}$ .

Let us now forget for a while that the coefficients of the  $(N-1)^{th}$  degree polynomials are also polynomials, and apply a fast polynomial product algorithm to compute the polynomial product in eq.(4). It is well known, since the work of Winograd [1] that the product of two polynomials with N coefficients can be obtained with a minimum of  $2N-1$  general multiplications. This minimum can be reached for small N, while for larger N the optimal algorithm involves too many additions to be of practical interest. In that case, suboptimal ones are often preferred. Hence, application of these polynomial product algorithms to eq.(4) will result in a scheme requiring  $2N-1$  "products", each one being in fact the product of a finite degree polynomial by an infinite degree sequence, that is an FIR filtering of length-L/N.

Compared to the initial situation, the arithmetic complexity is now as follows :

Eq.(4) requires  $N^2$  filterings of length  $L/N$ , which is about  $L$  multiply-accumulates (Macs) per output (it is only a rearrangement of the initial equation), while the fast polynomial product based scheme requires  $(2N-1)$  filterings of length  $L/N$ , which is about  $L(2N-1)/N^2$  Macs per output. Thus the improvement in arithmetic complexity is proportional to the length of the filter, and this is obtained at a fixed cost, depending on  $N$ . This means that, for large  $L$ , this approach will always be of interest. Precise comparisons are provided in Section 3 for each algorithm.

Slight additional improvements can be obtained by further considering eq.(4): In fact, eq.(4) contains not only the polynomial product, which allows the arithmetic complexity to be reduced, but also the so-called "overlap" in classical FFT-based schemes. By equating both sides of eq.(4), we have :

(5)

$$Y_{N-1} = \sum_{i=0}^{N-1} X_{N-1-i} H_i$$

$$Y_k = z^{-N} \sum_{i=k+1}^{N-1} X_{N+k-i} H_i + \sum_{i=0}^k X_{k-i} H_i \quad 0 \leq k \leq N-2$$

or in matrix form:

(6)

$$\begin{bmatrix} Y_{N-1} \\ Y_{N-2} \\ \vdots \\ Y_0 \end{bmatrix} = \begin{bmatrix} H_0 & H_1 & \dots & H_{N-1} \\ z^{-N} H_{N-1} & H_0 & \dots & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & H_1 \\ z^{-N} H_1 & \cdot & \cdot & z^{-N} H_{N-1} H_0 \end{bmatrix} \begin{bmatrix} X_{N-1} \\ X_{N-2} \\ \vdots \\ X_0 \end{bmatrix}$$

The right side of eq.(6) is the product of a pseudocirculant matrix [5] and a vector. Note that  $\{X_i\}$  and  $\{H_i\}$  play a symmetric role, hence can be exchanged in eq.(6). The equation is clearly in the form of a length- $N$  FIR filter whose coefficients are  $\{H_i\}$ , of which  $N$  outputs  $\{Y_i\}$  are computed. Following the notations of Winograd [1], we shall denote in the following an algorithm computing  $M$  outputs of a length- $N$  FIR filter by an  $F(M, N)$  algorithm. Considering the FIR filter as a whole, and the fast FIR algorithm as the

"diagonalization" of the pseudocirculant matrix of eq.(6) results in some circumstances in a reduction of the number of additions involved, compared with the usual approach which separates the polynomial product and the overlap. This will be seen in Section 3.

With the explanation above, the proposed algorithms can be understood as a multidimensional formulation of the FIR filtering, where computations along one dimension are performed through an efficient  $F(N,N)$  algorithm, resulting in a reduced arithmetic complexity, while the other dimension uses a direct computation, thus allowing the process to be a running one.

Let us also point out that, if the FIR filter of eq.(5) or eq.(6) is computed through an FFT-based scheme, and with the appropriate choice of  $N$  versus  $L$ , the usual FFT-based implementation of FIR filters can be seen to be a member of that class of algorithms. This is explained in [10], where it is shown that all fast FIR schemes including FFT-based ones can be expressed as :

- decimation of the involved sequences ( $N$  on input and output,  $M$  on the filter)
- evaluation of the obtained polynomials at  $N+M-1$  "interpolation points"  $\{\alpha_i\}$ ;
- "dot product" (or filtering);
- reconstruction of the resulting polynomials and overlap.

All the algorithms differ only by the choice of  $N$ ,  $M$ , and  $\{\alpha_i\}$ .

Section 3 provides various short-length FIR filtering algorithms in the case of real valued sequences.

### 3. Short-length FIR filtering algorithms

Let us first explain in some detail the simplest case : an  $F(2,2)$  algorithm, as given in [2,3,4]: considering  $N = 2$ ,  $\{\alpha_i\} = \{0,1,\infty\}$ , we obtain :

$$\begin{aligned}
 \text{(a1)} \quad & a_0 = x_0 & b_0 &= h_0 \\
 & a_1 = x_0 + x_1 & b_1 &= h_0 + h_1 \\
 & a_2 = x_1 & b_2 &= h_1 \\
 & m_i = a_i b_i ; & i &= 0,1,2 \\
 & y_0 = m_0 + z^{-2} m_2
 \end{aligned}$$

$$y_1 = m_1 - m_0 - m_2$$

When used in eq.(5) above, this algorithm results in the filtering scheme of Fig.1 where the problem of computing 2 outputs of a length N filter is turned into that of computing one output of three length-N/2 filters, at the cost of 4 adds per block of 2 outputs.

Comparison of arithmetic complexities are now as follows : the initial scheme has a cost of L mults and (L-1) adds per output point, to be compared with

3/4 L mults per output

$2 + 3/2 (L/2 - 1) = 3/4 L + 1/2$  adds per output.

It is seen that, excepted for very small L, both numbers of multiplications and additions have been reduced.

Successive decompositions are feasible, up to the point where the desired tradeoff has been obtained. Of course, this tradeoff depends on the implementation. On general purpose computers where a multiplication and an addition require about the same amount of time, the tradeoff allowing the fastest implementation will certainly correspond to a decomposition near the one minimizing the total number of arithmetic operations. On Digital Signal Processors, a multiply-accumulate operation will in general cost one clock cycle, and an appropriate criterion is certainly to count a Mac as a single operation. A third criterion of interest is the minimum number of multiplications. In the following, tables giving the minimum numbers for these three criteria will be provided.

Nevertheless, in nearly all type of implementations, the situation is much alike: the decomposition provided in Fig.1 reduces the arithmetic load in a manner proportional to L, the length of the filter, at a fixed cost (initialization of one Mac loop, or one Mac loop plus two adds). Hence the balance between the performances of algorithms depends on the timing spent in the initializations. The precise N by which the splitting becomes of interest therefore depends on the specific machine or circuit. Anyway, this type of splitting will always be of interest for large length filters, or even medium-size ones.

The remaining part of this section provides the simplest F(M,N) algorithms that can be used to reduce the arithmetic complexity of a length-L filtering. Different versions are provided, resulting in various operation counts, and various sensitivities to roundoff noise, a point which will not be dealt with in this paper.



The following algorithm is an F(2,2) algorithm with interpolation points  $\{\alpha_i\} = \{0, -1, \infty\}$ . It has exactly the same complexity as the previous one :

$$\begin{aligned}
 \text{(a2)} \quad & a_0 = x_0 & b_0 &= h_0 \\
 & a_1 = x_0 - x_1 & b_1 &= h_0 - h_1 \\
 & a_2 = x_1 & b_2 &= h_1 \\
 & m_i = a_i b_i ; & i &= 0, 1, 2 \\
 & y_0 = m_0 + z^{-2} m_2 \\
 & y_1 = m_0 + m_2 - m_1
 \end{aligned}$$

For the sake of completeness, two algorithms computing F(2,2) with  $\{\alpha_i\} = \{0, 1, -1\}$  and  $\{\alpha_i\} = \{1, -1, \infty\}$  are provided in Appendix B. They may be of interest as long as roundoff noise is concerned, but they require 3 multiplications and 6 additions, that is one more addition per output. Note that on some implementations (and essentially on DSP's), this is not a real drawback, since these additions are now of the type  $a \pm b$ , which can be efficiently implemented in many cases.

It is well known in the case of the usual implementation of a digital filter that the transposition of a graph provides a digital filter with the same transfer function. Winograd has proposed an approach allowing to obtain short-length FIR filtering algorithms by transposing polynomial product algorithms. We have shown that simple overlapping of polynomial products in eq.(5) is sufficient to construct FIR filtering algorithms. The transposition of polynomial products is not necessary. It only provides alternative versions of the algorithms. In the context of pseudocirculant matrix, we can transpose the algorithms in a much easier way, and we can prove that the total arithmetic complexity of all these algorithms is not changed by the transposition operation (both number of multiplications and additions). More details are given in Appendix A.

The following algorithms are the transposed versions of algorithms (a1) and (a2).

$$\begin{aligned}
 \text{(a3)} \quad & a_0 = x_0 - x_1 & b_0 &= h_0 \\
 & a_1 = x_0 & b_1 &= h_0 + h_1 \\
 & a_2 = z^{-2} x_1 - x_0 & b_2 &= h_1 \\
 & m_i = a_i b_i ; & i &= 0, 1, 2
 \end{aligned}$$

$$\begin{aligned} y_0 &= m_1 + m_2 \\ y_1 &= m_1 - m_0 \end{aligned}$$

$$\begin{aligned} \text{(a4)} \quad a_0 &= x_0 + x_1 & b_0 &= h_0 \\ a_1 &= x_0 & b_1 &= h_0 - h_1 \\ a_2 &= z^{-2} x_1 + x_0 & b_2 &= h_1 \end{aligned}$$

$$m_i = a_i b_i ; \quad i = 0, 1, 2$$

$$\begin{aligned} y_0 &= m_1 + m_2 \\ y_1 &= m_0 - m_1 \end{aligned}$$

The use of (a3) in a large FIR filter is provided in Fig.2.

Iterating the above algorithms results in radix-2 FIR algorithms, with a tree-like structure, as proposed in [4].

Higher radix algorithms can also be derived, and should be more efficient, as seen at the end of Section 2, since the ratio  $(2N-1)/N^2$  decreases. However, an optimal F(3,3) algorithm would require 5 different interpolation points, that is one more than the simplest ones:  $\{\alpha_i\} = \{0, 1, -1, ?, \infty\}$ , and the next simplest choices of the last interpolation point  $\{\pm 2, \pm 1/2\}$  result in an increased number of additions, and an increased sensitivity to roundoff noise. This is the reason why it is advisable to use a suboptimal F(3,3) algorithm which provides a better tradeoff between the number of multiplications and the number of additions.

Such an algorithm can be obtained by applying twice the algorithm (a1) as follows. Let us remark that (a1) is based on the following equation :

$$\begin{aligned} \text{(8)} \quad & (x_0 + x_1 z^{-1}) (h_0 + h_1 z^{-1}) \\ &= x_0 h_0 + x_1 h_1 z^{-2} + [(x_0 + x_1) (h_0 + h_1) - x_0 h_0 - x_1 h_1] z^{-1} \end{aligned}$$

Inspired by eq.(8) we rewrite the radix-3 aperiodic convolution equation as follows :

$$\begin{aligned} \text{(9)} \quad & [x_0 + (x_1 + x_2 z^{-1}) z^{-1}] [h_0 + (h_1 + h_2 z^{-1}) z^{-1}] \\ &= (x_0 + Fz^{-1}) (h_0 + Gz^{-1}) \\ &= x_0 h_0 + [(x_0 + F) (h_0 + G) - x_0 h_0 - FG] z^{-1} + FGz^{-2} \end{aligned}$$

Then, the algorithm in eq.(8) is once more applied to the computation of

$(x_0+F)(h_0+G)$  and  $FG$  which are still radix-2 aperiodic convolutions. This results in an aperiodic convolution algorithm requiring 6 mults and 9 adds (an optimal one would require 5 mults and 20 adds [11, pp.86]). Overlap has then to be performed by merging the terms  $z^0$  and  $z^{-3}$ , and the term  $z^{-1}$  and  $z^{-4}$ , with the appropriate delay. Hence, the  $F(3,3)$  algorithm seems to require 2 more adds than the corresponding radix-3 aperiodic convolution algorithm. Nevertheless, considering the redundancy during the overlap in  $F(3,3)$  algorithm allows a further reduction of the number of additions to 10 adds, the lowest number of operations to our knowledge.

$$\begin{array}{ll}
 \text{(a5)} & a_0 = x_0 & b_0 = h_0 \\
 & a_1 = x_1 & b_1 = h_1 \\
 & a_2 = x_2 & b_2 = h_2 \\
 & a_3 = x_0 + x_1 & b_3 = h_0 + h_1 \\
 & a_4 = x_1 + x_2 & b_4 = h_1 + h_2 \\
 & a_5 = x_0 + a_4 & b_5 = h_0 + h_1 + h_2
 \end{array}$$

$$m_i = a_i b_i; \quad i = 0, 1, 2, 3, 4, 5$$

$$t_0 = m_0 - m_2 z^{-3}$$

$$t_1 = m_3 - m_1$$

$$t_2 = m_4 - m_1$$

$$y_0 = t_0 + t_2 z^{-3}$$

$$y_1 = t_1 - t_0$$

$$y_2 = m_5 - t_1 - t_2$$

The use of this  $F(3,3)$  algorithm to reduce the arithmetic complexity of a larger FIR filter is provided in Fig.3, showing that the overall structure is that of a multirate filter bank, where  $\{H_1 + H_2, H_1, H_0 + H_1, H_2, H_0, H_0 + H_1 + H_2\}$  are decimated FIR filters.

Transposition of algorithm (a5) results in the following one, which is depicted in Fig.4.

$$\begin{array}{ll}
 \text{(a6)} & a_0 = x_2 - x_1 & b_0 = h_0 \\
 & a_1 = (x_0 - x_2 z^{-3}) - (x_1 - x_0) & b_1 = h_1 \\
 & a_2 = -a_0 z^{-3} & b_2 = h_2 \\
 & a_3 = (x_1 - x_0) & b_3 = h_0 + h_1 \\
 & a_4 = (x_0 - x_2 z^{-3}) & b_4 = h_1 + h_2
 \end{array}$$

$$a_5 = x_0$$

$$b_5 = h_0 + h_1 + h_2$$

$$m_i = a_i b_i ; \quad i = 0, 1, 2, 3, 4, 5$$

$$y_0 = m_2 + (m_4 + m_5)$$

$$y_1 = m_1 + m_3 + (m_4 + m_5)$$

$$y_2 = m_0 + m_3 + m_5$$

When used in a length-L FIR filter, both schemes (a5) and (a6) require  $2L/3$  multiplications and  $(2L+4)/3$  additions per output point, to be compared with L and L-1 operations respectively in the direct computation. This means that the computational load has been reduced by nearly 1/3.

Careful examination of Fig.1 - 2 and 3 - 4 shows that the distribution of the additions between the input samples and the subfilters' outputs is not the same in the initial algorithms and their transposed versions: In all cases, transposed algorithms have more input additions and less output additions. This fact should give them more robustness towards quantization noise.

Another case, which looks interesting at first glance is as follows: why not decimating the filter by a factor of 2, and X and Y by a factor of 3? This would be solved by an F(3,2) algorithm, which requires  $3 + 2 - 1 = 4$  interpolating points, that is the very number of the simplest interpolating points  $\{0, 1, -1, \infty\}$ . This means that F(3,2) or F(2,3) are the largest filtering modules that can be computed efficiently with an optimum number of multiplications :

$$\begin{array}{ll} \text{(a7)} & a_0 = x_3 - x_1 & b_0 = h_0 \\ & a_1 = x_1 + x_2 & b_1 = (h_0 + h_1)/2 \\ & a_2 = x_1 - x_2 & b_2 = (h_0 - h_1)/2 \\ & a_3 = x_2 - x_0 & b_3 = h_1 \end{array}$$

$$m_i = a_i b_i ; \quad i = 0, 1, 2, 3$$

$$y_0 = (m_1 + m_2) - m_3$$

$$y_1 = m_1 - m_2$$

$$y_2 = m_0 + (m_1 + m_2)$$

This algorithm looks promising, since the same performance as an F(3,3)

algorithm is obtained with a simpler one:  $F(3,2)$  requires 4 multiplications and 8 additions, which means that it reduces of the number of Macs by  $1/3$ , at the cost of 8 additions. Nevertheless, problems arise when using this algorithm for speeding up the computation of a large length- $L$  filter. The overall structure is depicted in Fig.5, where the main computing modules are a kind of  $1/3$  FIR decimators. Obtaining 3 successive outputs of the filter requires the computation of 4 length- $L/2$  inner products plus 13 adds, to be compared with algorithm (a5) which requires 6 length- $L/3$  inner products, plus 10 adds. Therefore, (a7) and (a5) have nearly the same arithmetic complexity. But (a7) requires  $3L$  memory registers, instead of  $2L$  registers in (a5), and a more complex control system, since the inner products are not true FIR filters any more.

All aperiodic convolution algorithms can be turned into FIR filtering algorithms by appropriate overlap. Suboptimal higher-radix ( $\geq 5$ ) aperiodic convolution algorithms can be derived using the approach in [12]. An  $F(5,5)$  algorithm based on this approach is given in Appendix B. This algorithm, requiring 12 mults and 40 adds is not optimum as far as the number of multiplications is concerned, but reaches the best tradeoff we could obtain.

All these  $F(N,N)$  algorithms are clearly similar to the ones proposed by Winograd [1]. They differ essentially on several points :

First, the recognition that, by using decimated sequences, the arithmetic complexity of an FIR filter can be reduced as soon as two outputs of the filter are computed regardless of the filter's length. Winograd's algorithms always require the computation of at least as many outputs as the filter's order.

Second, a straightforward derivation through eq.(4) of the  $F(N,N)$  algorithms by polynomial product algorithms which were extensively studied in the literature.

Third, a reduction in the number of additions, which is made feasible by naturally taking into account the "overlap" between two consecutive blocks of outputs (Hence the use of the pseudocirculant matrix).

Fourth, a new interpretation, in the context of pseudocirculant matrix, of algorithm transposition and a systematic method of obtaining transposed versions. (See appendix A)

#### 4. Composite length algorithms

We have explained in Section 2 that the application of an  $F(N,N)$  algorithm could

break the computation of a length- $L$  FIR filter into that of several length- $L/N$  filters, in such a manner that the arithmetic complexity is decreased. Nevertheless, the same process can iteratively be applied to the subfilters of length  $L/N$  as well, leading to composite length algorithms.

This section is concerned with the problem of finding the best way of combining the small-length filters, depending on various criteria.

We first evaluate the arithmetic complexity of a length  $L = N_1 N_2 L_2$  filter, when using two successive decompositions by  $F(N_1, N_1)$  first, and then by  $F(N_2, N_2)$ . Let us assume that, with the notations of Section 3, the length- $N_i$  filter requires  $M_i$  multiplications and  $A_i$  additions.

The first decomposition by  $F(N_1, N_1)$  turns the initial problem into that of computing  $M_1$  filters of length  $N_2 L_2$  at the cost of  $A_1$  additions. Further decompositions of the length- $N_2 L_2$  subfilters using  $F(N_2, N_2)$  require the consideration of a block of  $N_2$  outputs of these subfilters (hence a block of  $N_1 N_2$  outputs of the whole filter). Each of these subfilters is then transformed into  $M_2$  filters of length  $L_2$  at the cost of  $A_2$  additions.

The global decomposition thus has the following arithmetic complexity:

$$N_2 A_1 \text{ additions} + M_1 [M_2 \text{ length-} L_2 \text{ subfilters} + A_2 \text{ additions}].$$

That is :

$$(10) \quad M = M_1 M_2 L_2$$

$$(11) \quad A = N_2 A_1 + M_1 A_2 + M_1 M_2 (L_2 - 1)$$

Let  $m_i = M_i/N_i$  and  $a_i = A_i/N_i$  where  $i=1,2$ .  $m_i$  and  $a_i$  are the numbers of operations (mults and adds respectively) per output required by  $F(N_i, N_i)$ . Since eq.(10) and (11) are the arithmetic load for computing  $(N_1 N_2)$  outputs, the numbers of operations per output for the whole algorithm are :

$$(12) \quad m = m_1 m_2 L_2$$

$$(13) \quad a = a_1 + m_1 a_2 + m_1 m_2 (L_2 - 1)$$

An application of  $F(N_2, N_2)$  first, followed by  $F(N_1, N_1)$  would result in the same number of multiplications, and a number of additions which will be higher than eq.(13)

as long as :

$$(14) \quad a_1 + m_1 a_2 < a_2 + m_2 a_1 \quad \text{or} \quad (m_1 - 1)/a_1 < (m_2 - 1)/a_2$$

or, equivalently:

$$(15) \quad (M_1 - N_1)/A_1 < (M_2 - N_2)/A_2$$

Let us define

$$(16) \quad Q[F(N_i, N_i)] = (m_i - 1)/a_i = (M_i - N_i)/A_i$$

Q is a parameter specific of an algorithm. Its use has already been proposed in [8] for the cyclic convolution. Eq.(15) means that the lowest number of additions is obtained by first applying the short length FIR filter with the smallest Q, and then the one with the second smallest Q and so on. Table.1 provides  $Q[F(N, N)]$  for the most useful short-length filters. Two useful properties of  $Q[F(N, N)]$  are as follows :

- A straightforward FIR filtering "algorithm" has  $Q[F(1, N)] = 1$ , whatever N is. This means that, in order to minimize the number of additions, they must be located at the "center" of the overall algorithm, as was implicitly assumed up to that point.

- Iteratively applying an algorithm  $F(p, p)$  to obtain  $F(p^k, p^k)$  results in an algorithm with the same coefficient Q:

$$(17) \quad Q[F(p^k, p^k)] = Q[F(p, p)]$$

The demonstration is easily obtained by simply recognizing that applying first  $F(p^k, p^k)$  and second  $F(p, p)$  or applying them in the reverse order results in the same  $F(p^{k+1}, p^{k+1})$ .

Hence, as an example, an optimal ordering for  $L = 120$   $L_i = 2^3 \times 3 \times 5$   $L_i$  would be :

$$F(5, 5), F(2, 2), F(2, 2), F(2, 2), F(3, 3), F(1, L_i)$$

Of course, when it is desired to implement a length-L filter, it is very unlikely that  $F(L, L)$  is the most suitable algorithm for a specific type of implementation, even in the case where L is composite. The best thing to do, is to search for the tradeoff minimizing

some criteria depending on the implementation.

It is not our propose here to perform such an optimization in a special case, but we shall try to show, in the following, that improvements are feasible whatever the criterion is.

Assuming that  $L = N_1 \dots N_i \dots N_r L_r$ , and that a fast algorithm  $F(N_i, N_i)$  is used for all  $N_i$ , a straightforward one being used for  $L_r$ , the general formulae for evaluating the arithmetic complexity per output of a length- $L$  filter are as follows :

(18)

$$m = L_r \prod_{i=1}^r m_i$$

(19)

$$a = \sum_{i=1}^r a_i \prod_{j=1}^{i-1} m_j + (L_r - 1) \prod_{i=1}^r m_i$$

A first criterion of interest would be the minimization of the number of multiplications. Examination of eq.(18) shows that such a minimization is performed by fully decomposing  $L$  into  $N_1 \dots N_i \dots N_r L_r$ , and this results in the number of operations given in Table.2.

All the operation counts are provided assuming that the signal and the filter are real-valued, and the comparison is made with the real-valued FFT-based schemes [6,9] of twice the filter's length. Table.2 shows that the proposed approach is more efficient than FFT-based schemes up to  $L = 36$ , and very competitive up to  $L = 64$  which covers most useful lengths.

However, with today's technology, multiplication timings are not the dominant part of the computations any more when a parallel multiplier is built in the computer, and a very useful criterion is the sum of the number of additions and multiplications, i.e.,  $M+A$ .

Table 3 provides a description of the algorithms requiring the minimum value for such a criterion. It is seen that, although the basic  $F(N,N)$  algorithms do not improve the criterion, all the composite ones can be improved, and are even more efficient than FFT-based schemes up to  $N = 64$ . Consider  $N = 16$ , for example: FFT-based schemes hardly improve the direct computation (29.5 operations per point versus 31), while  $F(2,2)$ -based algorithm requires only 19.6 operations per point.



The previous criterion is well suited for general purpose computer implementation, where all operations are performed sequentially, but Digital Signal Processors still state another problem. In fact, DSP's perform generally a whole multiply-accumulate(Mac) operation in a single clock cycle, so that a Mac should be considered as a single operation, which is not more costly than an addition alone.

Table.4 provides a description of the algorithm minimizing the corresponding criterion: the sum of the number of Macs and I/O additions. Of course, this criterion is a rough measure of efficiency, since initialization time of Mac loops is not taken into account. Nevertheless, what Table.4 shows is that even this kind of criterion, taking partially into account the structure of DSP's, can be improved using our approach : a length - 64 filter can be implemented using this type of algorithms with nearly half the total number of operations (Macs+ I/O adds) per output point, compared to the trivial algorithm. And this is obtained with a block size of only 8 points. This shows that moderate up to large length filters can be efficiently implemented on DSP's using these techniques.

Two points should be emphasized here :

A lot of systems requiring digital filtering have constraints on the input/output delay, which prevents the use of FFT-based implementation of digital filter. Our approach allows to obtain a reduction of the arithmetic complexity whatever the block size is. This means that our approach allows to reduce the arithmetic complexity by taking into account such requirements as a constraint on the I/O delay.

Another remark, which is not apparent in the tables, is that for a given filter length  $L$ , there are often several algorithms providing comparable performance, for a given criterion. It allows us to choose among them the one which is the most suited for the specific implementation.

## **5. Conclusion**

We have presented a new class of algorithms for FIR filtering, showing that the basic building tools are short-length FIR modules in which the "multiplications" are replaced by decimated subfilters.

We have provided also the basic tools required for implementing these algorithms:

First, we propose short-length FIR filter modules of the Winograd-type with a small number of multiplications and the smallest number of additions. Their transposed versions are also provided.

Second, we give some rules concerning the best way of cascading these short-length FIR modules to obtain composite-length algorithms.

Finally, we show that, for three different criteria, this class of algorithms allows to obtain better tradeoffs than the previously known algorithms, which should make them useful in any kind of implementation.

It is concluded that the presented algorithms allow not only to compute from moderate to long length FIR filtering on DSP's in a more efficient manner, but also to compute short-length (<64) FIR filtering more efficiently than FFT-based algorithms when considering the total number of operations.

These algorithms also suggest efficient multiprocessor implementations due to their inherent parallelism, and efficient realization in VLSI, since their implementations require only local communication, instead of a global exchange of data, as is the case for FFT-based algorithms.

#### Appendix Several short length FIR algorithms.

F(2,2) algorithm with (3 multiplications, 6 additions),  $\{\alpha_i\} = \{0, 1, -1\}$ .

$$\begin{aligned} \text{(a8)} \quad a_0 &= x_0 & b_0 &= h_0 \\ a_1 &= x_0 + x_1 & b_1 &= (h_0 + h_1)/2 \\ a_2 &= x_0 - x_1 & b_2 &= (h_0 - h_1)/2 \end{aligned}$$

$$m_i = a_i b_i; \quad i = 0, 1, 2$$

$$y_0 = m_0 + z^{-2}(m_1 + m_2 - m_0)$$

$$y_1 = m_1 - m_2$$

F(2,2) algorithm with (3 multiplications, 6 additions),  $\{\alpha_i\} = \{1, -1, \infty\}$ .

$$\text{(a9)} \quad a_0 = x_0 + x_1 \quad b_0 = (h_0 + h_1)/2$$

$$\begin{aligned} a_1 &= x_0 - x_1 & b_1 &= (h_0 - h_1)/2 \\ a_2 &= x_1 & b_2 &= h_1 \end{aligned}$$

$$m_i = a_i b_i ; \quad i = 0, 1, 2$$

$$\begin{aligned} y_0 &= m_0 + m_1 - m_2 + z^{-2}m_2 \\ y_1 &= m_0 - m_1 \end{aligned}$$

F(5,5) algorithm with (12 multiplications, 40 additions). This algorithm is based on the approach in [12].

$$\begin{aligned} (a10) \quad c_0 &= x_0 + x_3 & g_0 &= (h_0 + h_3)/2 \\ c_1 &= x_1 + x_4 & g_1 &= (h_1 + h_4)/2 \\ c_2 &= x_2 & g_2 &= h_2/2 \\ c_3 &= x_0 - x_3 & g_3 &= (h_3 - h_0)/2 \\ c_4 &= x_1 - x_4 & g_4 &= (h_4 - h_1)/2 \\ c_5 &= x_2 & g_5 &= -h_2/2 \end{aligned}$$

$$\begin{aligned} a_0 &= c_0 + c_1 + c_2 & b_0 &= (g_0 + g_1 + g_2)/3 \\ a_1 &= c_0 - c_2 & b_1 &= g_0 - g_2 \\ a_2 &= c_1 - c_2 & b_2 &= g_1 - g_2 \\ a_3 &= a_1 + a_2 & b_3 &= (b_1 + b_2)/3 \\ a_4 &= c_3 - c_4 + c_5 & b_4 &= (g_3 - g_4 + g_5)/3 \\ a_5 &= c_3 - c_5 & b_5 &= (-2g_3 - g_4 + g_5)/3 \\ a_6 &= c_3 + c_4 & b_6 &= (g_3 + 2g_4 + g_5)/3 \\ a_7 &= c_4 + c_5 & b_7 &= (g_3 - g_4 - 2g_5)/3 \\ a_8 &= x_0 & b_8 &= h_0 \\ a_9 &= x_0 & b_9 &= h_1 \\ a_{10} &= x_1 & b_{10} &= h_0 \\ a_{11} &= x_4 & b_{11} &= h_4 \end{aligned}$$

$$m_i = a_i b_i ; \quad i = 0, 1, \dots, 11$$

$$\begin{aligned} u_0 &= m_1 - m_3 \\ u_1 &= m_2 - m_3 \end{aligned}$$

$$\begin{aligned} d_0 &= m_0 + u_0 \\ d_1 &= m_0 - u_0 - u_1 \end{aligned}$$

$$d_2 = m_0 + u_1$$

$$d_3 = m_4 - m_5 + m_7$$

$$d_4 = -m_4 + m_6 + m_7$$

$$d_5 = m_4 + m_5 + m_6$$

$$d_6 = m_9 + m_{10}$$

$$f_0 = d_2 - d_5$$

$$f_1 = d_1 - d_4$$

$$f_2 = d_0 - d_3$$

$$f_3 = d_2 + d_5$$

$$f_4 = d_1 + d_4$$

$$f_5 = d_0 + d_3$$

$$y_0 = m_8 + z^{-5} f_5$$

$$y_1 = d_6 + z^{-5} (f_0 - m_8)$$

$$y_2 = f_2 - m_{11} + z^{-5} (f_1 - d_6)$$

$$y_3 = f_3 + z^{-5} m_{11}$$

$$y_4 = f_4$$

## References

- [1] S. Winograd, "Arithmetic complexity of computations", CBMS-NSF Regional Conf. Series in Applied Mathematics, SIAM Publications No. 33, 1980.
- [2] Z.J. Mou, P. Duhamel, "Fast FIR filtering: algorithms and implementations", Signal Processing, Dec. 1987, pp.377-384.
- [3] M. Vetterli, "Running FIR and IIR filtering using multirate filter banks", IEEE Trans. Acoust. Speech, Signal Processing, Vol. 36, N°5, May 1988, pp.730-738.
- [4] H.K. Kwan, M.T. Tsim, "High speed 1-D FIR digital filtering architecture using polynomial convolution", in Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing, Dallas, USA, April, 1987, pp.1863-1866.
- [5] P.P. Vaidyanathan, S.K. Mitra, "Polyphase networks, block digital filtering, LPTV systems and alias-free QMF banks: A unified approach based on pseudocirculants," IEEE Trans. Acoust. Speech, Signal Processing, Vol. 36, N°3, March 1988, pp.381-391.
- [6] P. Duhamel, M. Vetterli, "Improved Fourier and Hartley transform algorithms: application to cyclic convolution of real data", IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-35, N°6, June 1988, pp.818-824.
- [7] S. Winograd, "Some bilinear forms whose multiplicative complexity depends on the field of constants", Mathematical Systems Theory 10 (Sept. 1977), pp.169-180. Also in Number Theory in Digital Signal Processing (ed. by J.H. McClellan and C.M. Rader), Prentice-Hall, Englewood Cliffs, N.J., 1979.
- [8] R.C. Agarwal and J.W. Cooley, "New algorithms for digital convolution", IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-25, Oct.1977, pp. 392-410.
- [9] H.V. Sorensen, D.L. Jones, M.T. Heideman, C.S. Burrus, "Real-valued fast Fourier transform algorithms", IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-35, N°6, June 1988, pp.849-863.
- [10] Z.J. Mou, P. Duhamel, "A unified approach to the fast FIR filtering algorithms", in Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing, New-York, USA, April 1988, pp.1914-1917.
- [11] R.E. Blahut, Fast Algorithms for Digital Signal Processing, Addison-Wesley, Reading, MA, 1985.
- [12] P.C. Balla, A. Antoniou, S.D. Morgera, "Higher radix aperiodic convolution algorithms", IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-34, N°1, February 1986, pp.60-68.
- [13] R.E. Crochiere, L.R. Rabiner, Multirate Digital Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1983.

Table.1    Quality factors for several short-length algorithms.

Algorithms	Q
F(1,N)	1
F(2,2)	0.25
F(3,3)	0.3
F(5,5)	0.175

Table.2 Arithmetic complexity of an F(N,N) algorithm by short-length FIR and by FFT-based cyclic convolution.

N	Short-length FIR				FFT-based FIR				Direct FIR	
	M	m	A	a	M <sub>FFT</sub>	m	A <sub>FFT</sub>	a	m	a
2	3	1.5	4	2					2	1
3	6	2	10	3.3					3	2
4	9	2.25	20	5	15	3.75	43	10.75	4	3
5	12	2.4	40	8					5	4
6	18	3	42	7	33	5.5	83	13.83	6	5
8	27	3.38	76	9.5	43	5.38	131	16.38	8	7
9	36	4	90	10					9	8
10	36	3.6	128	12.8					10	9
12	54	4.5	150	12.5					12	11
15	72	4.8	240	16	112	7.47	353	23.53	15	14
16	81	5.06	260	16.25	115	7.19	355	22.19	16	15
18	108	6	306	17					18	17
20	108	5.4	400	20					20	19
24	162	6.75	498	20.75					24	23
25	144	5.76	680	27.2					25	24
27	216	8	630	23.33					27	26
30	216	7.2	744	24.8	225	7.50	829	27.63	30	29
32	243	7.59	844	26.38	291	9.09	899	28.09	32	31
36	324	9	990	27.5	271	7.53	1084	30.11	36	35
60	648	10.8	2280	38	455	7.58	1955	32.58	60	59
64	729	11.39	2660	41.56	701	10.95	2179	34.05	64	63
128	2187	17.09	8236	64.34	1667	13.02	5123	40.02	128	127
256	6561	25.63	25220	98.52	3483	13.61	11779	46.01	256	255
512	19683	38.44	76684	149.77	8707	17.01	26627	52.01	512	511
1024	59049	57.67	232100	226.66	19459	19.00	59395	58.00	1024	1023

Table.3 Minimum sum of operations (Mults + Adds). (mxn) means the decomposition by a fast F(m,m) algorithm using optimal ordering followed by direct length-n FIR filters. (FFT) means using FFT-based schemes.

N	Decomposition	M+A	/N	Block size
2	direct	6	3	1
3	direct	15	5	1
4	(2x2)	26	6.5	2
5	direct	45	9	1
6	(3x2)	56	9.33	3
8	(2x2x2)	94	11.75	4
9	(3x3)	120	13.33	3
10	(5x2)	152	15.2	5
12	(2x3x2)	192	16	6
15	(5x3)	300	20	5
16	(2x2x2x2)	314	19.63	8
18	(2x3x3)	396	22	6
20	(5x2x2)	472	23.6	10
24	(2x2x3x2)	624	26	12
25	(5x5)	740	29.6	5
27	(3x3x3)	810	30	9
30	(5x3x2)	912	30.4	10
32	(2x2x2x2x2)	1006	31.44	16
36	(2x2x3x3)	1260	35	12
60	(5x2x3x2)	2784	46.4	30
64	(FFT)	2880	45.00	64
128	(FFT)	6790	53.05	128
256	(FFT)	15262	59.62	256
512	(FFT)	35334	69.01	512
1024	(FFT)	78854	77.01	1024



Table.4 Minimum number of Macs (Length of scalar products + I/O adds). (mxn) means the decomposition by a fast F(m,m) algorithm using optimal ordering followed by direct length-n FIR filters. (FFT) means using FFT-based schemes.

N	Decomposition	MAC	/ N	Block size
2	direct	4	2	1
3	direct	9	3	1
4	direct	16	4	1
5	direct	25	5	1
6	direct	36	6	1
8	direct	64	8	1
9	direct	81	9	1
10	(2x5)	95	9.5	2
12	(2x6)	132	11	2
15	(3x5)	200	13.33	3
16	(2x8)	224	14	2
18	(2x9)	279	15.5	2
20	(2x2x5)	325	16.25	4
24	(2x2x6)	444	18.5	4
25	(5x5)	500	20	5
27	(3x9)	576	21.33	3
32	(2x2x8)	736	23	4
36	(2x2x9)	909	25.25	4
60	(5x2x6)	2064	34.4	10
64	(2x2x2x8)	2336	36.5	8
128	(2 <sup>4</sup> x8)	7264	56.75	16
256	(2 <sup>5</sup> x8)	22304	87.13	32
512	(2 <sup>6</sup> x8)	67936	132.69	64
1024	(2 <sup>7</sup> x8)	205856	201.03	128

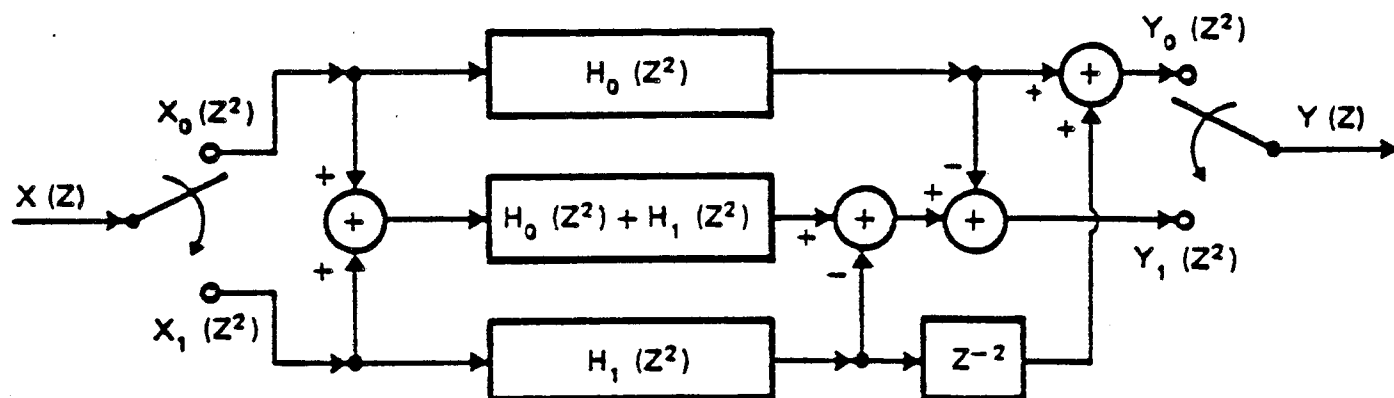


Figure 1

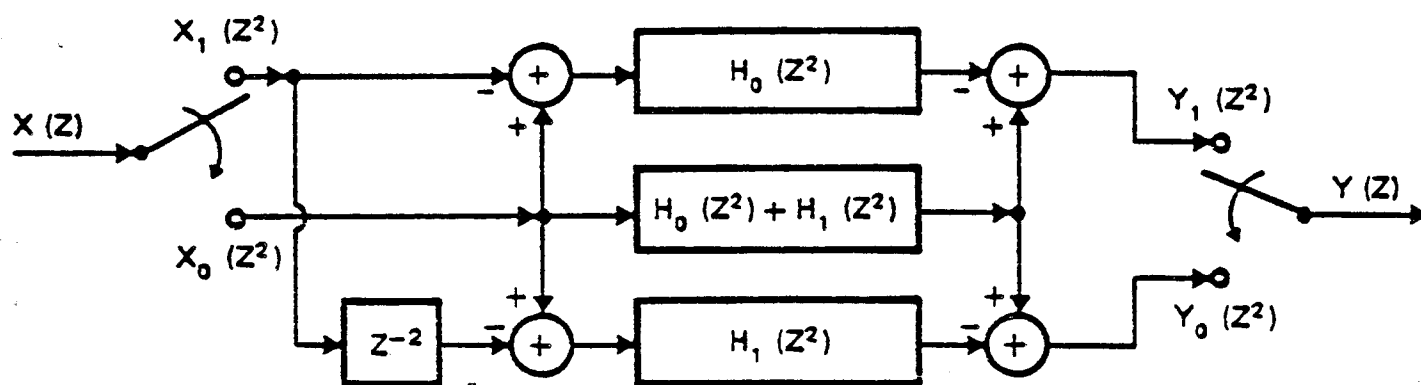


Figure 2

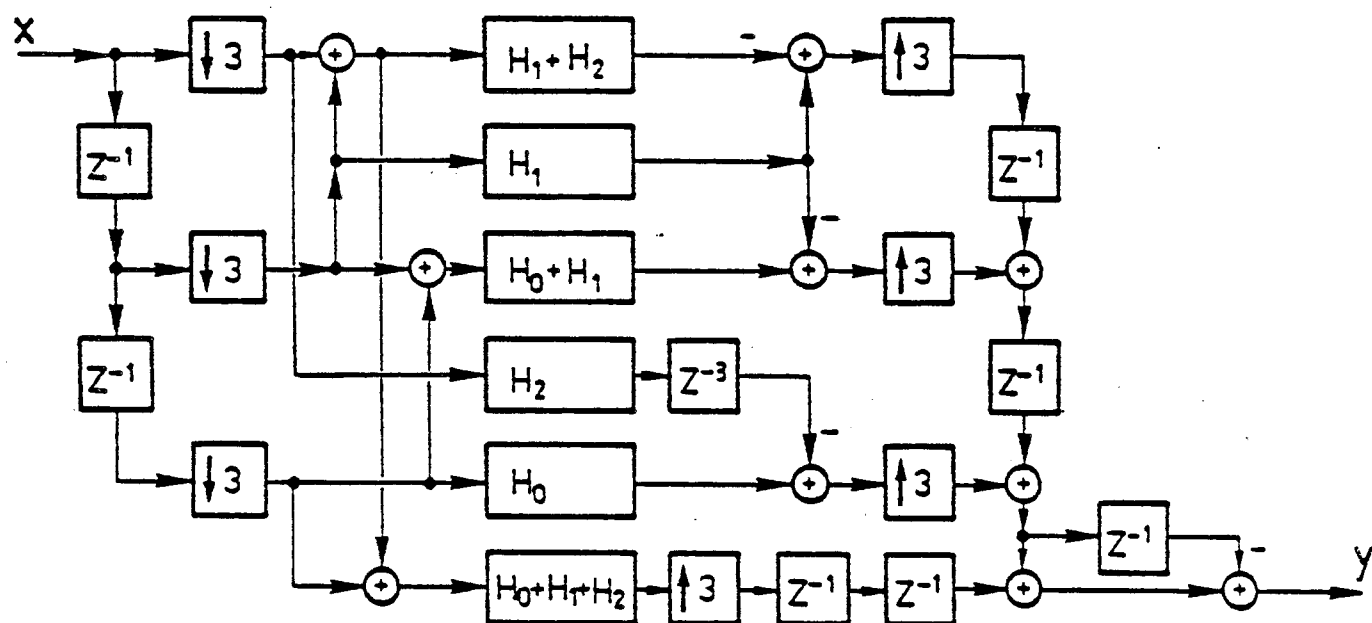


Fig. 3

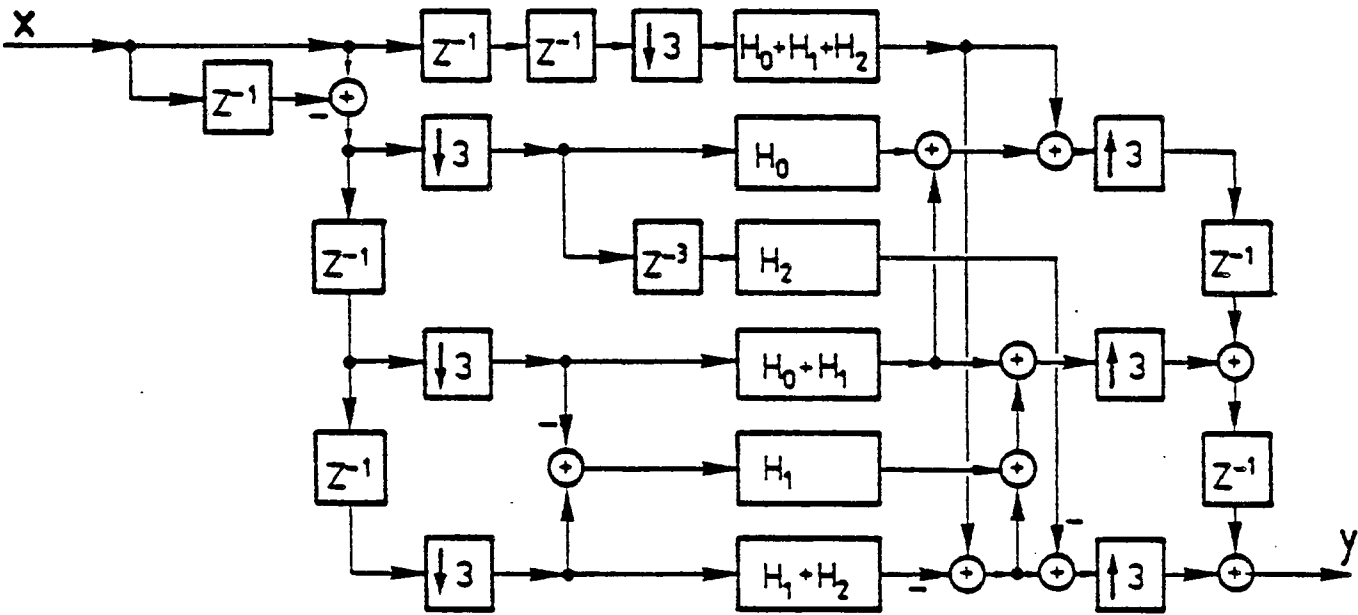


Fig 4

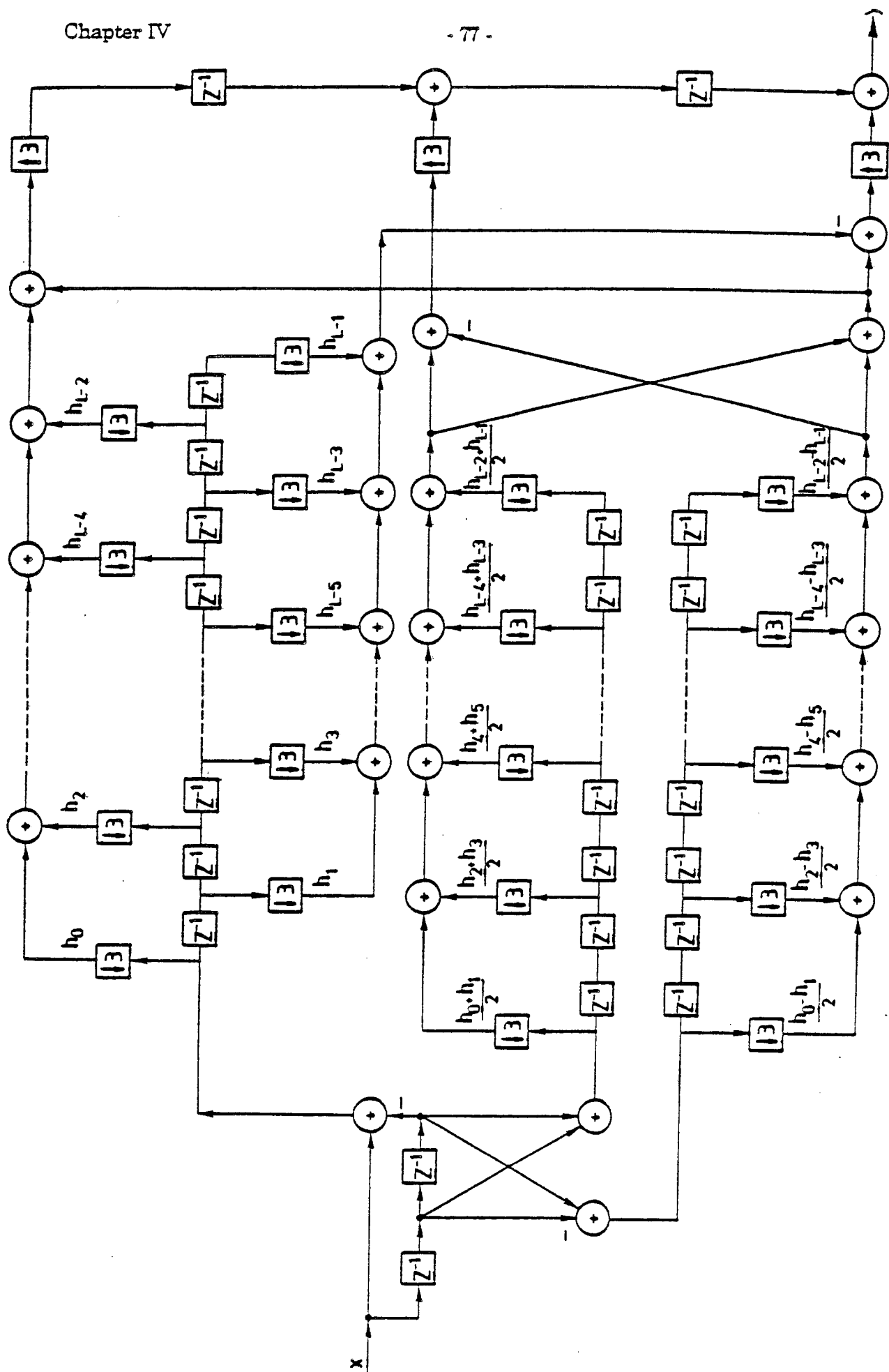


Fig 5

## Chapter 5

### Fast multiplier-accumulator

#### 5.1 Multiplier-accumulator

Multiplier-accumulator is instrumental to modern digital signal processing. It constitutes the heart of the widely used Digital Signal Processors (DSP's) as well as the general purpose computer.[And57]

The DSP's are often optimized for computing an FIR filter which is in form of an inner product and calculated by successive multiply-accumulates. The time required by a multiply-accumulate serves as a major benchmark for various DSP's [Lee88].

This chapter will deal with the arithmetic aspects of multiplier-accumulator as well as its VLSI layout design. The contents of this chapter provides also the basis for the subsequent chapters where we study the architectures of the FIR filter which can be taken as a generalized multiplier-accumulator.

We will also propose some new schemes for designing fast multiplier-accumulators.

#### 5.2 Reduction of the number of partial products

In general a  $B \times N$  ( $B \leq N$ ) multiplier is equivalent to the summing of  $B$  operands which are called partial products, with appropriate shift: (Assume the numbers are in 2's complement.)

$$\begin{aligned}
 XH &= (-x_0 + \sum_{i=1}^{B-1} x_i 2^{-i})H \\
 &= -x_0 H + \sum_{i=1}^{B-1} (x_i H) 2^{-i}
 \end{aligned} \tag{5.1}$$

$\{x_i H; i=0,1,\dots,B-1\}$  are the partial products.

If the number of partial products can be reduced, in consequent, the speed of the summing can be increased.

Booth's encoding [Boo51] allows to reduce more than half the number of partial products. However Booth's encoding is difficult to implement because of the variable number and variable position of the partial products, except for one of the two multiplicative factors is constant. The most widely used technique is the modified Booth's encoding [Mac61] which constantly halves the number of partial products.

### 5.2.1 Booth's encoding

This technique is based on the following relation of a binary number (not in 2's complement):

$$1111\dots 1 = 2^n + 2^{n-1} + 2^{n-2} + \dots + 2 + 1 = 2^{n+1} - 1 \quad (5.2a)$$

or in 2's complement:

$$1111\dots 1 = -2^n + 2^{n-1} + 2^{n-2} + \dots + 2 + 1 = -1 \quad (5.2b)$$

This means that a string of 1's can be transformed into two or one effective bits. It is particularly of interest in designing multiplier which uses actually binary arithmetic. By the principle of (5.2),  $n$  effective partial products are reduced to only two or one products. Two kinds of partial products are possible:  $\{-H, H\}$ . It is originally proposed by Booth [Boo51]. Since the Booth's encoding uses negative bits, it is also known as canonical signed bit (CSB) representation.

For example a 10-bit number in 2's complement after encoding becomes:

$$1110011011 = 00(-1)0100(-1)0(-1) \quad (5.3)$$

In practical design, only four partial products need to be taken into account instead of 10. In general, more than 50% of partial products can be reduced.

However the big problem for implementation is that we can not systematically determine how many and which partial products should be taken into account.

This difficulty limits heavily the application of Booth's encoding to multiplier design, except the case either  $X$  or  $H$  is constant. In that case, Booth's encoding applies because we can predetermine the number and the position of effective partial products. Hence it is particularly of interest for full custom design.

### 5.2.2 The modified Booth's encoding (MBE)

By considering three bits in  $X$ , with one bit commonly shared by the neighboring groups of three bits, we obtain:

$$\begin{aligned}
 X &= -x_0 + \sum_{i=1}^{B-1} x_i 2^{-i} \\
 &= 2^{-1} \sum_{i=0; i \text{ even}}^{B-1} (-2x_i + x_{i+1} + x_{i+2}) 4^{-i} \\
 &\text{where } x_B = 0 (B \text{ even}) \text{ and } x_B = x_{B+1} = 0 (B \text{ odd}).
 \end{aligned} \tag{5.4}$$

This formulation provides the basis for the modified Booth's encoding [Mac61, Rub75]. The binary number is transformed into a base-4 number. Its digits

Table.5.1. Truth table of control signal generation for the modified Booth's encoding.

$x_i$	$x_{i+1}$	$x_{i+2}$	S(i) Shift	N(i) Null	C(i) Complement
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	1	0
1	0	0	1	1	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	0	0	1



$\{-2x_i+x_{i+1}+x_{i+2}\}$  take one of the five values:  $\{-2,-1,0,1,2\}$ . When  $X$  multiplies  $H$ , five partial products are possible:  $\{-2H, -H, 0, H, 2H\}$ . In total the number of partial products is reduced to  $B/2$  for  $B$  even or to  $(B+1)/2$  for  $B$  odd.

The generation of an effective partial product  $P(i)=(-2x_i+x_{i+1}+x_{i+2})H$  depends on the value of  $(-2x_i+x_{i+1}+x_{i+2})$ . So, we can encode the five values of  $(-2x_i+x_{i+1}+x_{i+2})$  into three control signals: Shift  $S(i)$ , Null  $N(i)$  and Complement  $C(i)$ . The truth table is given in Table.5.1.

From the Table.5.1, we get the following logic relations:

$$\begin{aligned} S &= \overline{x_{i+1}} \oplus x_{i+2} \\ N &= \overline{x_i} \overline{x_{i+1}} \overline{x_{i+2}} + x_i x_{i+1} x_{i+2} \\ C &= x_i \end{aligned} \quad (5.5)$$

These control signals will determine how to select or generate one of the five partial products. The effective partial products  $P(i)$  is generated according to the equation below:

$$p_j(h_j h_{j+1}) = [(\overline{S}h_j + Sh_{j+1})N] \oplus C \quad (5.6)$$

and

$$P(i) = \sum_{j=0}^{b-1} p_j 2^{-j} + C(i) 2^{-(b-1)}$$

$p_j$  is the  $j$ th bit of  $P(i)$  and  $h_j$  is the  $j$ th bit of  $H$ .  $p_j$  is the function of  $\{h_j, h_{j+1}\}$ .

In Fig.5.1 we depict the scheme for partial product generation. It is not the only way for the generation. Many alternatives are discussed in [Mon88, Mou89].

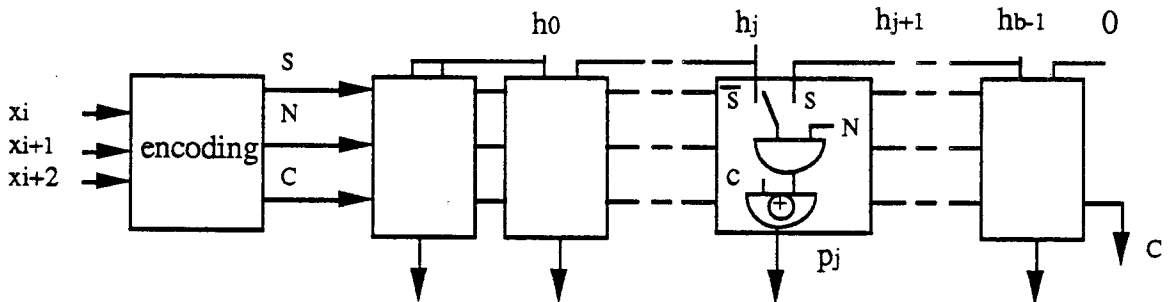


Fig.5.1 Partial product generation using encoding

Since the MBE results in a constant reduction of the number of partial products to one half, and the partial products are easy to generate, it is preferred in general multiplier design.

### 5.3 Multioperand summation

Having reduced and generated the partial products, we should sum them up. This section deals with the fast multioperand summation. Conventional  $B \times N$  multiplier requires  $(B+N)t_{fa}$  as multiplication time,  $t_{fa}$  denoting the time required by a full adder. The pipeline is often difficult in such multiplier since the carries and sums have different directions of propagation. We will show in the following that the multiplication time can be reduced.

#### 5.3.1 Using carry-save adder (CSA)

CSA leads to important improvement in computing speed, compared to carry-propagation adder (CPA). A CSA adds 3 numbers into 2 numbers while a CPA adds 2 numbers into 1 number. Fig.5.2 shows the 2 adders.

In a computation we are often interested only in the final result. Hence during the computation, we may use different representation of the intermediate information as long as the computation can be performed much rapidly. The carry-save adder supports such a representation that all the intermediate results are two numbers:  $C$  and  $S$ , which are independent of one another. The final result is the sum of the two numbers. In fact, it is not necessary to specify them as carry and sum.

A multiplication can be seen as the summation of a number of operands or partial products. In Fig.5.3, we depict a scheme of summing up  $N$  numbers using a direct CSA array. We need  $N-2$  CSAs to sum up the  $N$  numbers into 2 numbers. Then the computing time is  $(N-2)t_{fa}$ , given  $t_{fa}$  the time required by a full adder.

In designing a multiplier, we should add the final  $C$  and  $S$  to get the product. However since our objective is to design multiplier-accumulator, we will leave the two numbers as the input to the subsequent computations.

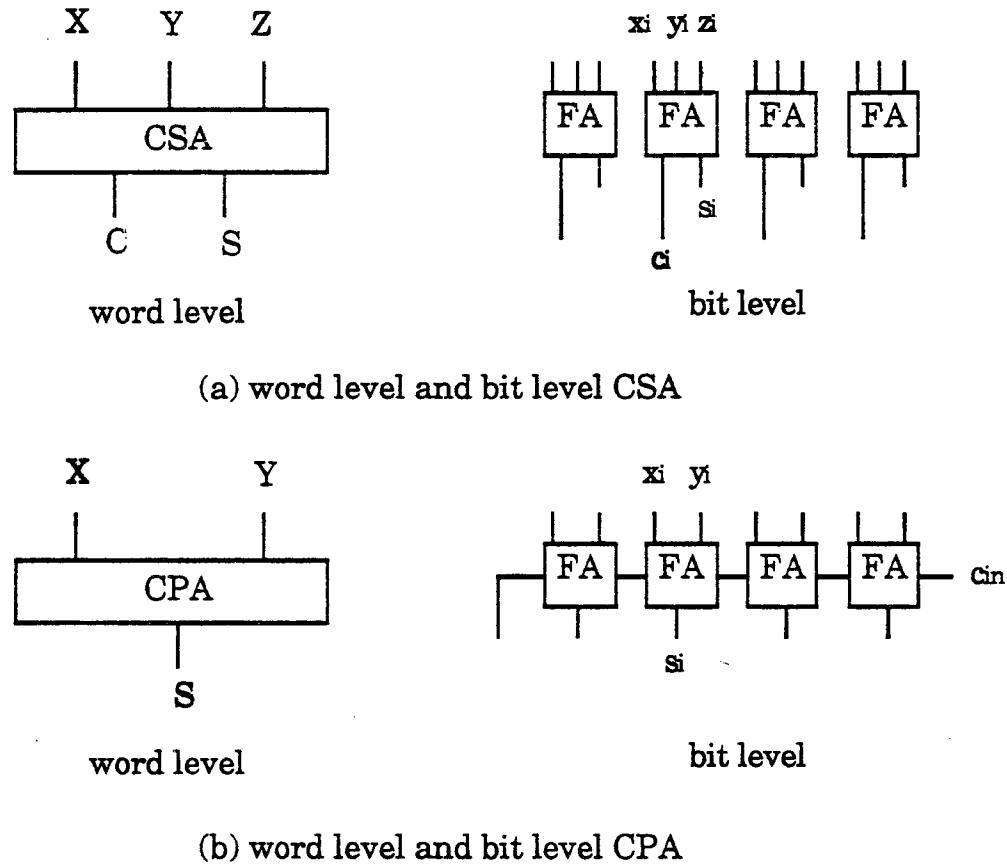


Fig.5.2 Carry-save adder(CSA) and carry-propagation adder(CPA). FA = Full Adder.

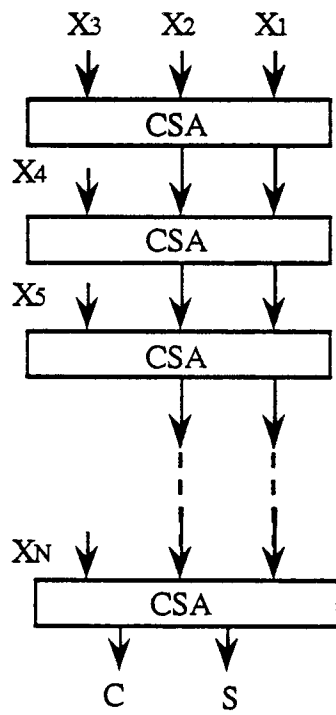


Fig.5.3 Direct CSA array for summing N numbers

CSA array is very regular so that the design is easy. Another advantage is that the data flow is unidirectional, which allows to insert pipeline in the array much evenly while keeping a regular structure. In a CPA array the data flow propagates in a diagonal way. The pipeline registers along the diagonal can not be either evenly distributed or as regular and compact as the CSA array.

Note that the summing time of  $N$  operands using direct CSA array is proportional to  $N$ . A technique termed Wallace-tree allows to further speed up the summing.

### 5.3.2 Wallace-tree and its compact layout

Wallace-tree [Wal61] exploits the concurrency in the summing of multiple operands. The building block is still the CSA. However the CSA's are organized in a tree like manner. We always center our attention on summing  $N$  numbers to two outputs. A  $N-2$  W-tree will be used to do the work. A 9-2 tree is shown in Fig.5.4 to give an idea. The computing time is only 4 tfa, instead of 7 tfa using direct CSA array. In general, an  $N-2$  W-tree requires a delay of  $O(\log N)$  full adders. The precise minimum number of layers can be determined by the following formula [Was82]:

$$k = E[\log_{3/2} N/2] + 1 ; \text{ for } N > 3. \quad (5.7)$$

where  $E[Q]$  means the integer part of a real  $Q$ . The Table 5.2 gives a number of  $N$ 's and their corresponding  $k$ 's.

Although the Wallace-tree looks very attractive, it is rarely implemented because its structure leads to difficult routing and exaggerated area requirement. Thus a general belief is that the W-tree is unimplementable and out of practical use [Vui83, Mon88].

In the following we propose a compact yet regular design which may cover the most useful  $N$ . Let us consider the 9-2 tree design. We take at first the bits of the same weight  $2^{-k}$  of all the 9 operands into account and sum them up. Further more, we should not forget the carry from the lower weight bit while summing. Then a bit-slice W-tree can be established as shown in Fig.5.5. The wordlevel tree is the regular assemblage of the bit-slice tree.

Table.5.2 Number of levels for an N-2 Wallace-tree

N	k
3	1
4	2
$5 \leq N \leq 6$	3
$7 \leq N \leq 9$	4
$10 \leq N \leq 13$	5
$14 \leq N \leq 19$	6
$20 \leq N \leq 28$	7
$29 \leq N \leq 42$	8
$43 \leq N \leq 63$	9

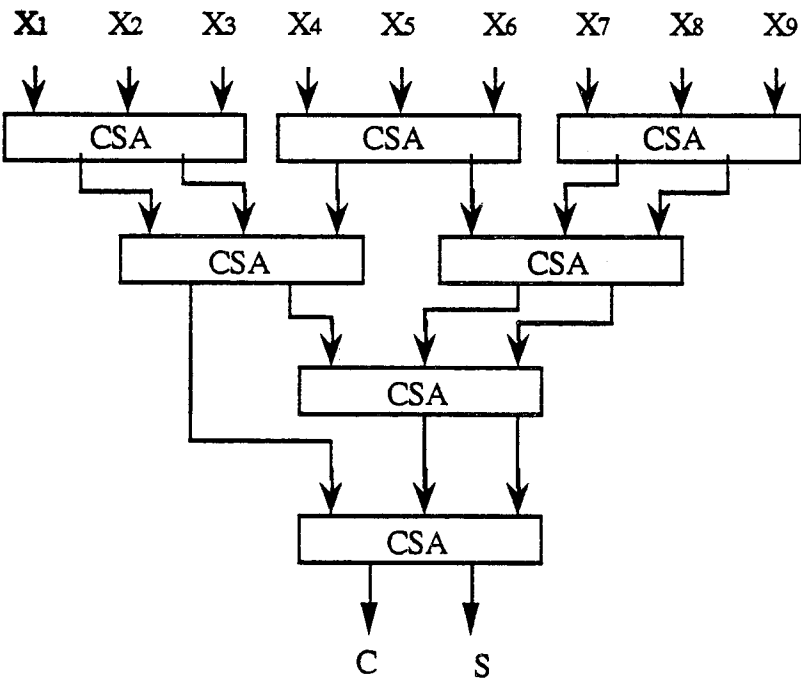


Fig.5.4 The 9-2 Wallace-tree

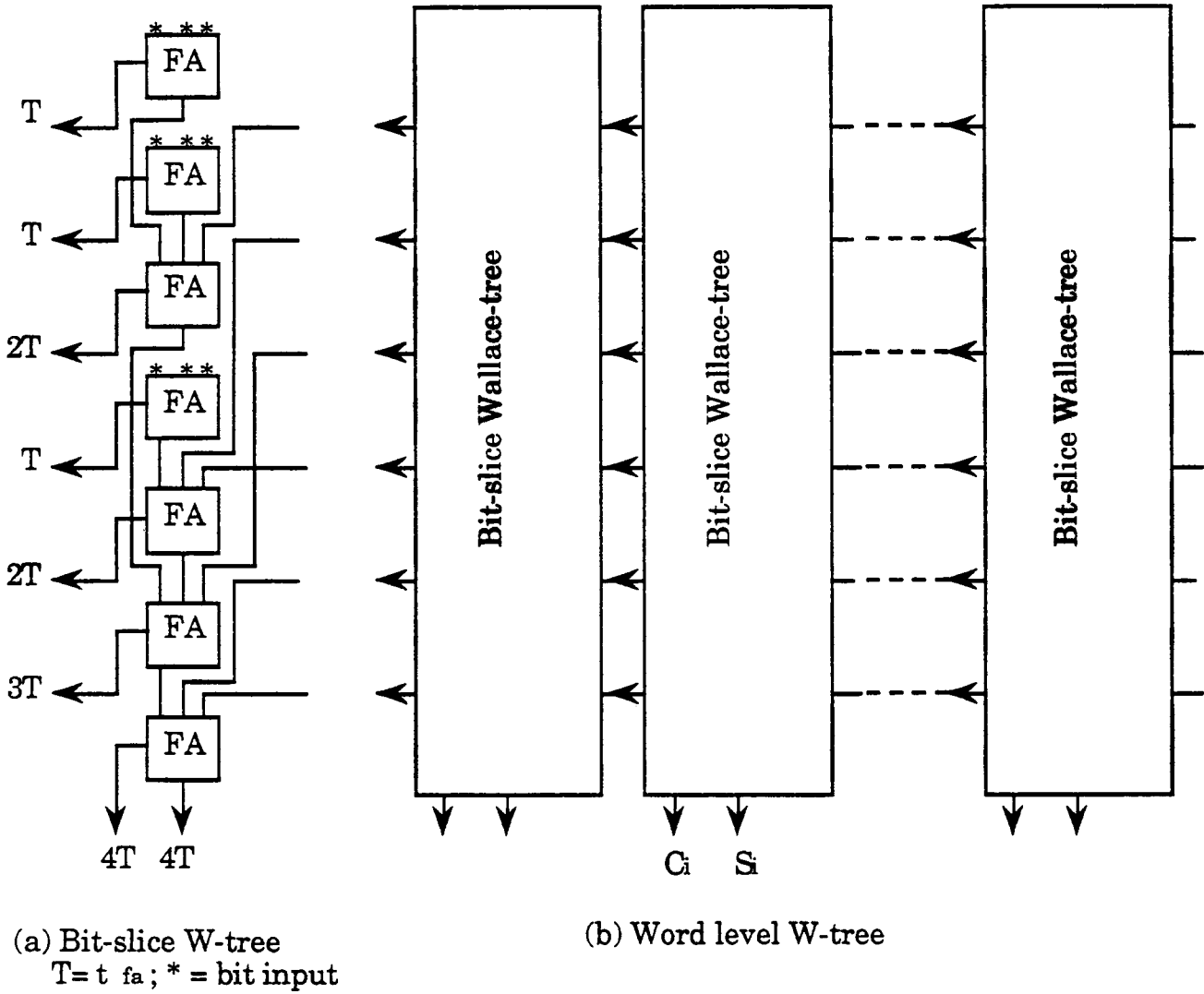


Fig.5.5 Compact and regular Wallace-tree design.

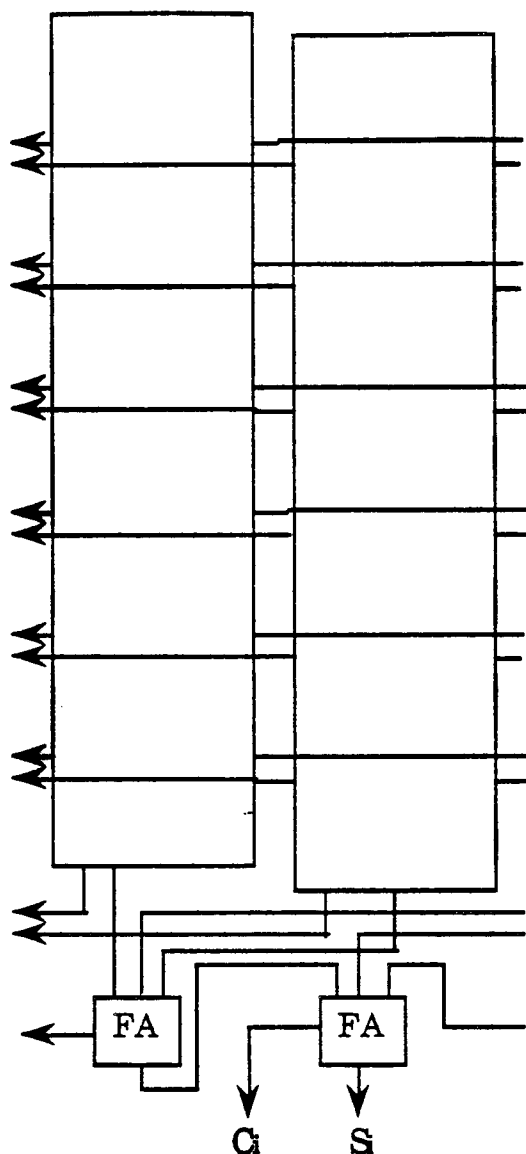


Fig.5.6 A bit-slice 18-2 W-tree using bit-slice 9-2 W-tree

For large  $N$ , the routing may become complex. We propose a solution suitable for this case. Instead of one column, we use two columns. For example, a bit-slice 18-2 tree is constructed as shown in Fig.5.6.

#### 5.4 Fast accumulator

Usually an accumulator accumulates a series of numbers one after another. It is symbolized in Fig.5.7. The time required by an accumulation, denoted by  $t_{ac}$  is equivalent to that by an addition, i.e., proportional to the number of bits of operands. An accumulation is often considered as an addition.

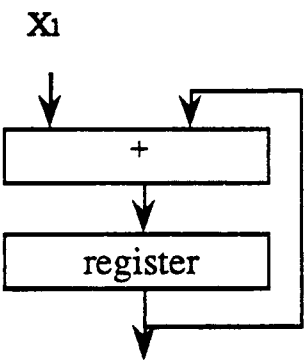


Fig.5.7 Conventional accumulator

However, some techniques allow to reduce tac considerably, as presented in the following.

5.4.1 Carry-save (uni-operand) accumulator

As mentioned before, a carry-save adder adds 3 numbers to 2 numbers in tfa time. If we route the 2 outputs, after latch, to the input of the CSA, a carry-save accumulator is formed as shown in Fig.5.8. A series of numbers are accumulated one by one as usual. We define it thus as a uni-operand accumulator. Note that tac (=tfa) is independent of the wordlength of operands.

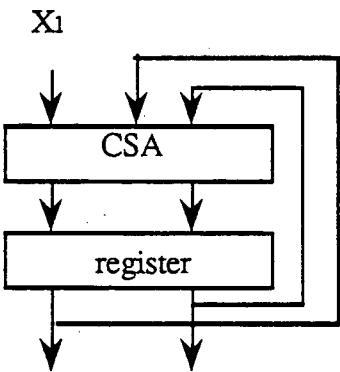


Fig.5.8 Uni-operand accumulator using CSA.

Two numbers represent the intermediate results of the accumulation. For obtaining the final result, we should sum up the two numbers. This can be easily and economically done in most cases. For example, when L numbers



are to be accumulated, Ltfa is required. We have then Ltfa time for the final addition, which may be realized using bit-serial adder.

#### 5.4.2 Bi-operand and multi-operand accumulator

The carry-save accumulator accumulates the numbers one by one. If we accumulate more than one operand in a single computing cycle, the total number of computing cycle can be reduced. In the case of a parallel multiplier-accumulator, the output of CSA array or Wallace-tree is two numbers. It is obligatory to accumulate 2 numbers each cycle.

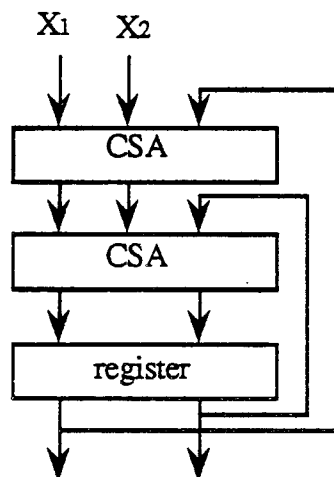
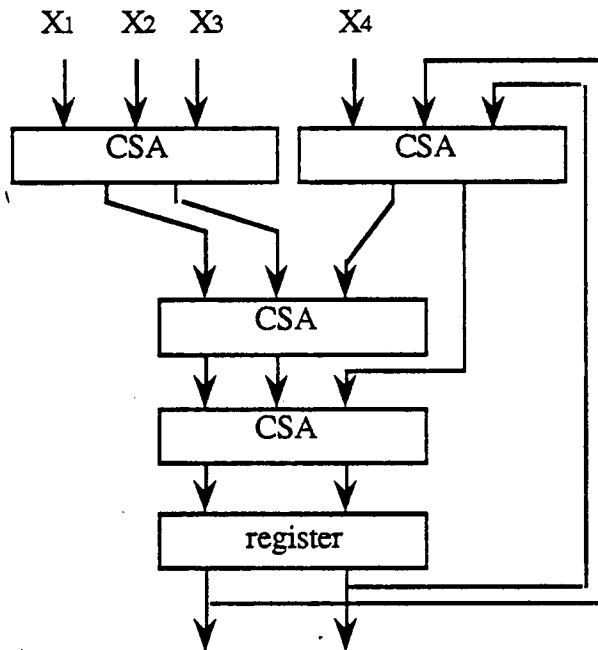


Fig.5.9 Bi-operand accumulator.

Anderson et al. [And67] proposed a scheme allowing to accumulate multiple operands per cycle. A scheme of interest is that of bi-operand accumulator (see Fig.5.9). This accumulator accomplishes one accumulation in 2 tfa time. Since the output of all operations using carry-save arithmetic is two numbers, the bi-operand accumulator is mostly used [Li87,San88].



**Fig.5.10 Quadri-operand accumulator.**

By the same token, we can design a quadri-operand accumulator as shown in Fig.5.10. This accumulator accumulates 4 numbers in only  $3t_{fa}$ , using a 6-2 Wallace-tree. This scheme can be generalized to an N-operand accumulator by using an  $(N+2)$ -2 W-tree and routing the 2 outputs to the top of the tree [Was82]. When N is large, the routing may become difficult and impractical. A good tradeoff is to use an N-2 Wallace-tree plus a bi-operand accumulator which avoids the long routing [Was82]. Another variant of interest is based on two  $N/2$ -2 W-trees plus a quadri-operand accumulator. This scheme allows double column design as in Fig.5.6. All the three alternate N-operand accumulator designs are shown in Fig.5.11. The first one is optimal in terms of delay but may be difficult to design. The other 2 schemes are suboptimal but very easy to design. Since the minimum number of CSA delays is not very sensitive to N (see Table.5.1), the third alternative may become optimal for certain values of N. For example when  $N=12$ , both the first and the third schemes have 7 CSA delays.

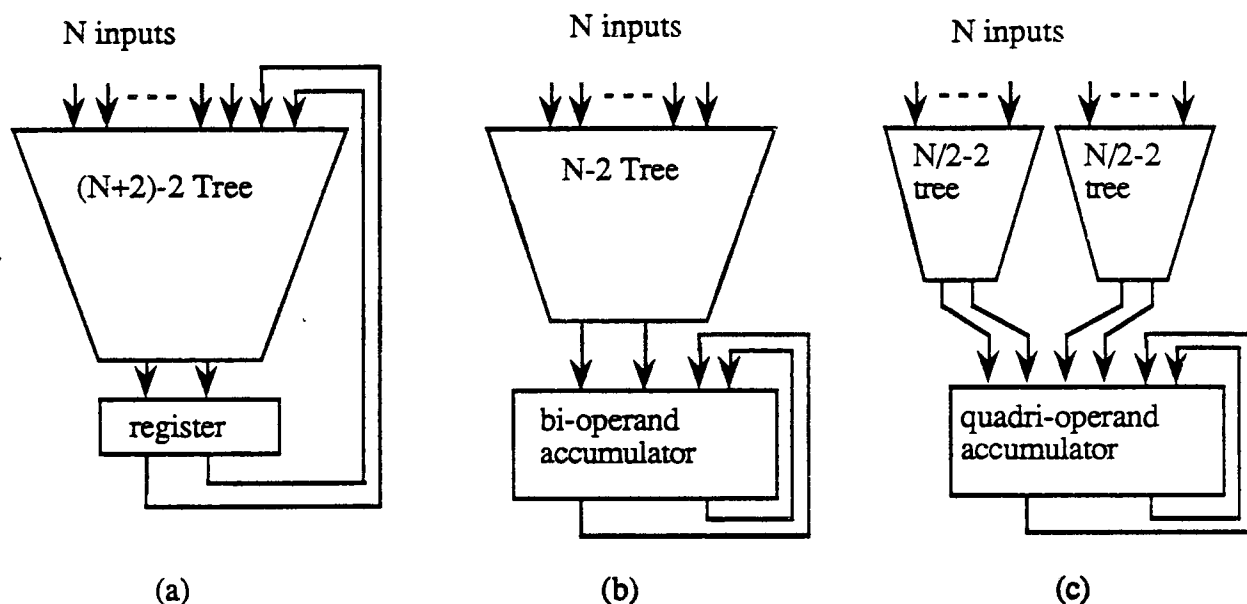


Fig.5.11 Three alternatives of N-operand accumulator.

Some other schemes can be developed in the same manner: using  $m$   $N/m-2$  W-trees plus a  $2m$ -operand accumulator. We may choose appropriate  $m$  to meet the best trade-off between the accumulation time and the complexity of design.

In fact, all multiplier-accumulators belong to the class of multi-operand accumulator. A multiplier alone can be also implemented using small multioperand accumulator recursion. For example when designing a  $64 \times 64$  parallel multiplier, the modified Booth's encoding results in 32 partial products. Using an eight-operand accumulator, the product can be computed in 4 cycles.

## 5.5 Pipelined multiplier-accumulator and new schemes

Pipeline is a popular technique to increase considerably the processing speed. The state-of-the-art IC technology allows larger and larger scale integration. The current criterion for optimal IC design is whether the circuit minimizes the value of  $AT^2$ .  $A$  is the chip area while  $T$  is the clock cycle period. This criterion largely favors reducing  $T$  or increasing the functional frequency at cost of increasing the chip area. For example a reduction in  $T$  by 2 times may justify the increase in  $A$  by 4 times.

Hence when latency is not critical, such a criterion is always encouraging to insert pipeline registers in order to maximize the processing speed.

When pipelining the schemes presented before, the minimum  $T$  we can reach is  $2t_{fa}$ , using the bi-operand accumulator. Previous works [And67,Li87,San88], as shown in Fig.5.12, did not exceed this limit. The lower bound of  $T$  is limited by the number of CSAs in the accumulation loop.

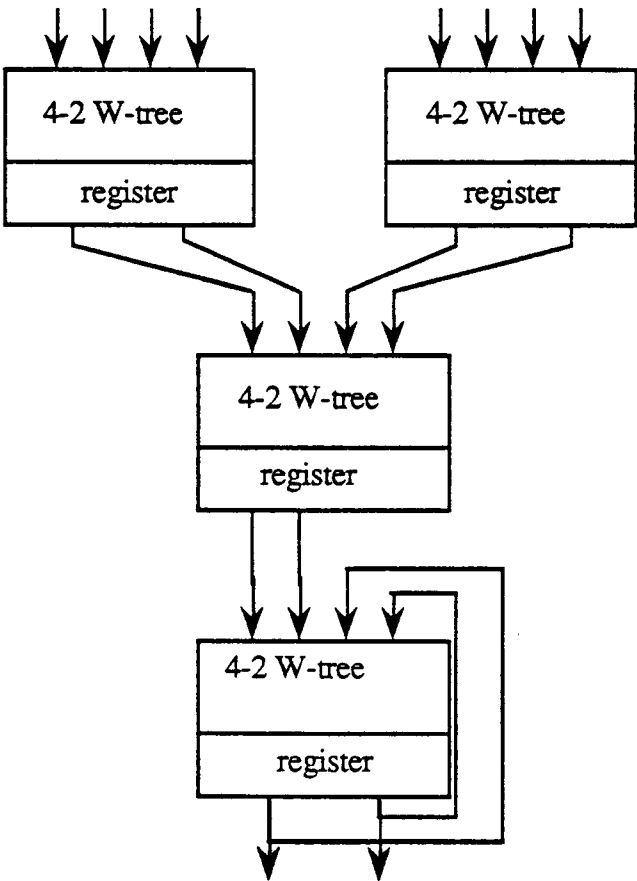


Fig.5.12 Pipelined 8-operand accumulator.

Further improvement of the lower bound of  $T$  depends on reducing the number of CSAs in the accumulation loop. We propose some new schemes having only one CSA in the loop.

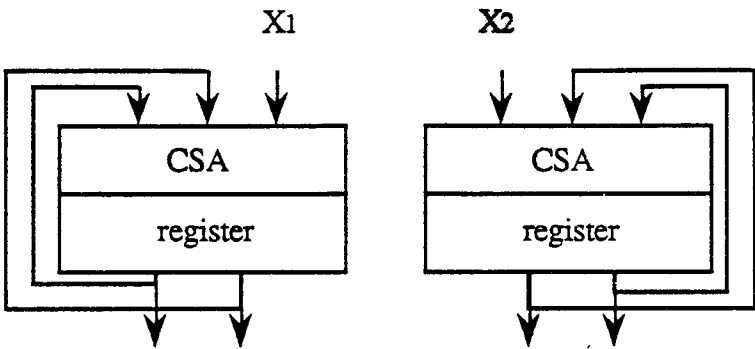


Fig.5.13 Pipelined bi-operand accumulator with 4 outputs.

A rather direct approach makes use of two uni-operand accumulators as shown in Fig.5.13. This bi-operand accumulator accumulates a series of numbers into 4 numbers. The minimum cycle time is reduced to  $t_{fa}$ . Theoretically, an N-operand accumulator can be constructed using N uni-operand accumulators so that the minimum accumulation cycle time is lowered to  $t_{fa}$ . But it results in more complex final adder design. Then it is often of interest to use a CSA array or W-tree to sum up N operands to 2 numbers followed by a bi-operand accumulator.

An alternative to that in Fig.5.13 is shown in Fig.5.14. It can accumulate as fast as that in Fig.5.13 while the result is represented by only three numbers. The final adder can be thus simplified.

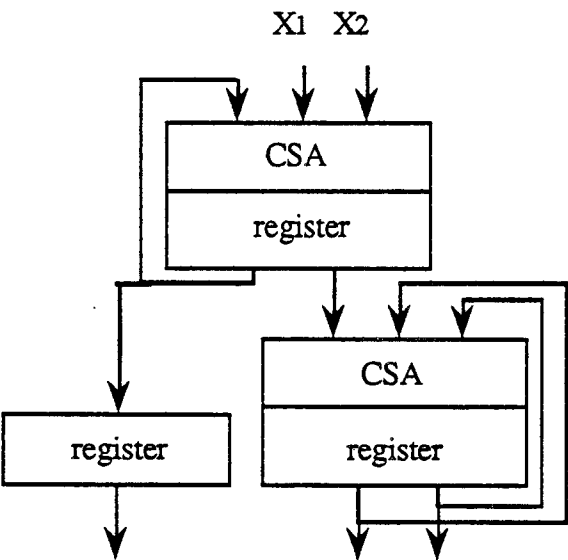


Fig.5.14 Pipelined bi-operand accumulator with 3 outputs.

## 5.6 Remarks

This chapter has been concerned with the arithmetic aspects in designing fast multiplier-accumulators, including: reduction of the number of partial products by the modified Booth's encoding; carry-save adder array and Wallace-tree; fast accumulators. We have proposed a compact yet regular design to implement Wallace-tree, overcoming the general belief that the W-tree is not practical [Vui83,Mon88]. New accumulators are presented which allow to realize the fastest accumulation, while bit-level pipeline is introduced [Hat88].

Wallace-tree and CSA array represent the two ends of the spectrum of N-operand summing operators: one requires the least delay with the most irregularity for design while the other does the contrary. There are many other types of summing tree meeting different delay/regularity tradeoffs, for example the 4-2 tree [Li88,San88] and the balanced delay tree [Zur86,Mon88]. The competitiveness of these trees benefits largely from their regular design. Since our technique allows to improve the design of Wallace-tree, we believe that the Wallace-tree is always a good candidate for summing operator design.

## Chapter 6

### **New results on the distributed arithmetic implementation of FIR filters**

The distributed arithmetic is an efficient bit-serial technique to implement FIR filters [Cro73,Pel74,Bur77] or scalar products. Its principle is to precompute the  $2^L$  possible sums of coefficients of a filter and to store them in a  $2^L$ -word ROM, given  $L$  the number of filter coefficients. At the output of ROM, a shift-accumulator adds  $B$  times partial sums to compute an output of the filter where  $B$  is the number of bits of each input.

This chapter describes several new structures which improve significantly the computing speed and hardware requirement when implementing an FIR filter by distributed arithmetic. These structures are characterized by the following aspects:

- fast accumulation;
- without ROM;
- massive use of carry-save adders and Wallace-tree;
- regular architecture allowing easy pipeline until bit-level;
- reduced number of computing cycles per output.

Usually, one accumulation is equivalent to one addition. By applying fast accumulation technique presented in the Chapter 5, we can largely reduce the time of one accumulation and to perform one addition per output instead of  $B$  additions.

The ROM occupies very large chip area and is often the bottleneck of computing speed. By decomposing the ROM into smaller ones, we reduce the hardware requirement and increase the speed. All the outputs of the small ROMs need to be summed. In the extreme case of ROM decomposition, only the coefficients are stored. The computing part is totally composed of parallel adders. The adder-based architectures are particularly investigated in this chapter. All the adders are the carry-save ones, instead of the carry-

propagation adders. Wallace-tree, an efficient technique usually used in designing hardware multiplier, is applied in these architectures.

We will show also how to pipeline, as well as to exploit concurrency in these architectures. The pipeline may be massively and easily used in the presented architectures until bit-level. The resulting architectures keep a great regularity which facilitates implementations.

The conventional scheme computes one output per  $B$  cycles. Then we propose to apply the modified Booth's encoding (MBE) to reduce the number of computing cycles per output to  $B/2$  ( $B$  even) or  $(B+1)/2$  ( $B$  odd) while requiring only a few additional hardware. The filtering speed can be thus doubled. When the coefficients are fixed for a dedicated application, further simplification can be achieved in a systematic manner.

Linear phase FIR filters have symmetric coefficients. Such symmetry can result in important reduction of hardware requirement. But straightforward implementation needs one more computing cycle, i.e.,  $(B+1)$  cycles per output. A new encoding technique is introduced to eliminate the extra cycle, requiring  $B$  cycles per output as usual. We will also apply the pipeline to the design of symmetric FIR filters.

Although the distributed arithmetic is originally a bit-serial technique, it allows bit-parallel implementation as well [Li88]. The parallel implementation will be dealt with. The new encoding technique is very attractive for parallel design of linear phase FIR filters and shows better performance than other alternatives.

## 6.1 The principle of distributed arithmetic

In this section, we give a brief review of the principle of distributed arithmetic implementation of FIR filters.

The FIR filtering is to compute the following operation :

$$y_n = \sum_{i=0}^{L-1} h_i x_{n-i} \quad n = 0, 1, \dots, \infty \quad (6.1)$$



where  $\{h_i\}$  are filter's coefficients and  $\{x_i\}$  the inputs. Suppose input signal samples are coded in 2's complement of  $B$  bits:

$$x_j = -x_{j0} + \sum_{k=1}^{B-1} x_{jk} 2^{-k}; \quad x_{jk} \in \{0, 1\} \quad (6.2)$$

this gives

$$\begin{aligned} y_n &= \sum_{i=0}^{L-1} h_i (-x_{n-i0} + \sum_{k=1}^{B-1} x_{n-ik} 2^{-k}) \\ &= - \sum_{i=0}^{L-1} h_i x_{n-i0} + \sum_{k=1}^{B-1} \left( \sum_{i=0}^{L-1} h_i x_{n-ik} \right) 2^{-k} \end{aligned} \quad (6.3)$$

Since  $\{x_{n-ik}\}$  are binary numbers, a partial sum  $P_k$  of weight  $2^{-k}$

$$P_k(h_0, h_1, \dots, h_{L-1}) = \sum_{i=0}^{L-1} h_i x_{n-ik}; \quad k = 0, 1, \dots, B-1 \quad (6.4)$$

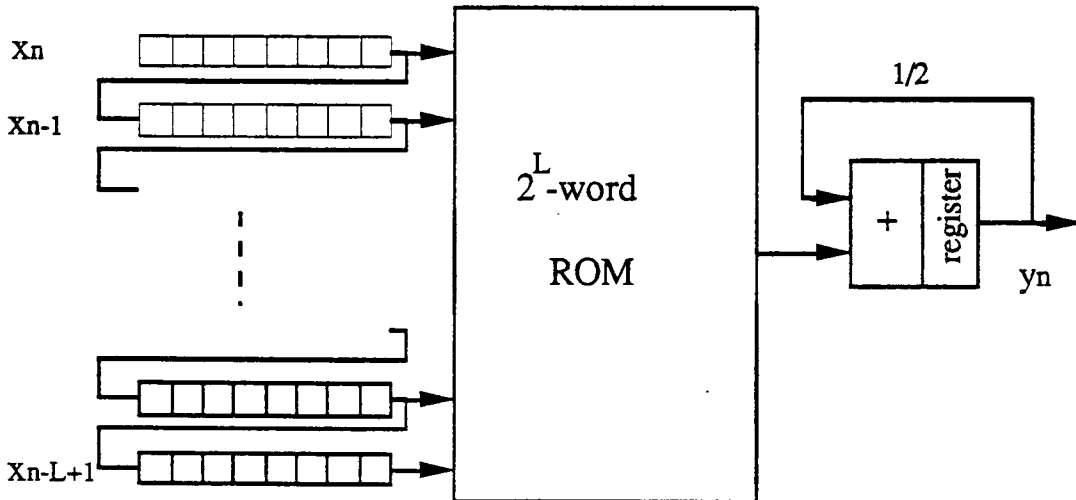


Fig.6.1 FIR filter structure by distributed arithmetic

is a function of  $L$  binary variables  $\{x_{n-i,k} ; i = 0,1,\dots,L-1\}$ . If the  $2^L$  possible values of the function are stored appropriately in a lookup table or in a ROM, by addressing the ROM with the  $L$  binary variables, we get the correspondent partial sum. A FIR filter can be thus implemented as shown in Fig.6.1. The addressing of the ROM is performed from the least significant bit (LSB) to the most significant bit (MSB) of each input sample (we can do it also from MSBs to LSBs). The partial sums of different weights are shifted and accumulated following eq.(6.3). One filter output is thus produced every  $B$  cycles. While the MSBs are addressing the ROM at the  $B$ th cycle, the output of ROM should be complemented by 2 before accumulation according to the first term in eq.(6.3).

If  $L$  is large, the  $2^L$ -word ROM would not only cost too much hardware but also limit the computing speed. An interesting way is to divide it into two  $2^{L/2}$ -word ROMs and to accumulate the outputs of two ROMs to form a partial sum [Bur77]. By analogy, each  $2^{L/2}$ -word ROM can be further divided into smaller  $2^{L/4}$ -word ROMs. After each dividing one more adder is needed to combine the outputs of two smaller ROMs.

## 6.2 New structures of general FIR filters

This section presents some new general FIR structures while linear phase FIR filter structures will be found in section 6.3.

### 6.2.1 Fast accumulation

The results in the last chapter apply directly to increasing the computing speed.

Using the uni-operand accumulator, we can improve at once the scheme in Fig.6.1. While the ROM is divided into two smaller ROMs, we should accumulate the partial sums two by two. Then a fast bi-operand accumulator can be applied to this case, as depicted in Fig.6.2.

The further decomposition of ROM is still feasible in order to decrease the chip area and to increase the speed. Suppose the ROM is decomposed into  $N$  smaller  $2^{L/N}$ -word ROMs. Then  $N$  partial sums should be accumulated together.  $N$ -operand accumulator, as presented in the last chapter, can be used.

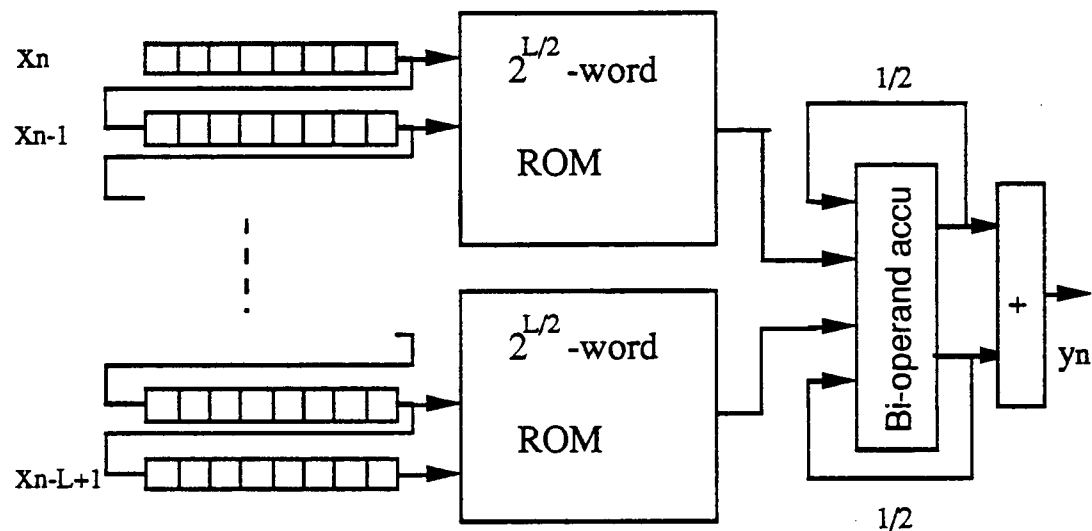
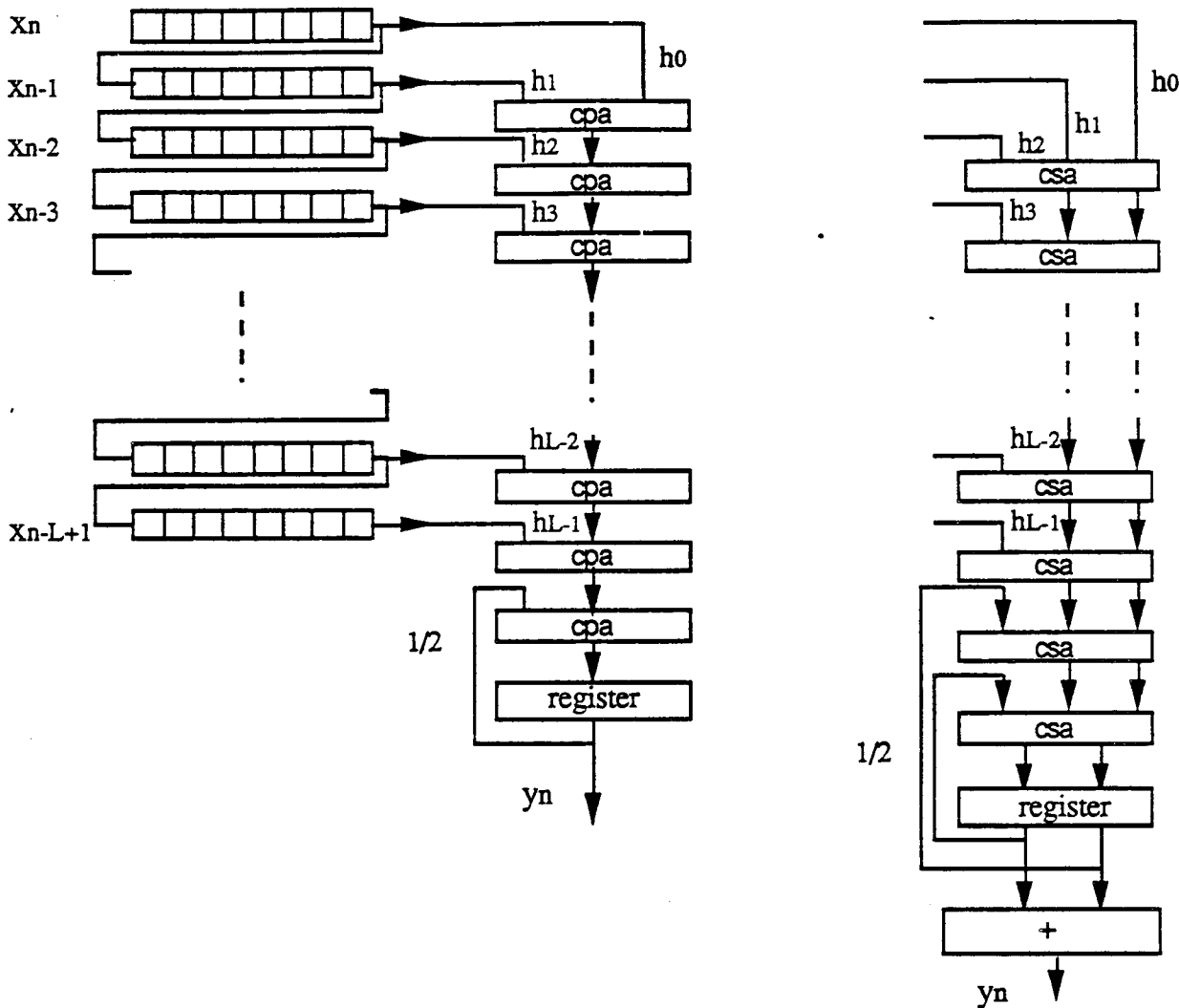


Fig.6.2 ROM dividing and fast accumulation

All these fast accumulators result in two numbers after  $B$  accumulations. Hence only one additon is needed to sum up the two numbers every  $B$  cycles. The final adder can be simplified using bit-serial approach.

6.2.2 Adder-based structures

In the extreme case of ROM dividing, only the coefficients are stored. All the computing part is composed of adders. A rather simple and regular structure (Fig.6.3a) using carry-propagation adder (CPA) is proposed in [Duh88]. By using carry-save adder (CSA), a more efficient structure can be obtained, as shown in Fig.6.3b. In this configuration one accumulation takes time of  $Lt_{fa}$  ( $L$  is the filter's order) which is proportional to  $L$ .



(a) using carry-propagation adders;      (b) using carry-save adders.

Fig.6.3 Adder-based FIR filter structures

We can regard the problem as an  $L$ -operand accumulation. Then we can use a fast  $L$ -operand accumulator as the computing part. Wallace-tree is applicable to the problem, especially in case  $L$  is not too large. When  $L$  is very large, the design of  $W$ -tree becomes difficult.

In order to increase the speed, we should exploit the concurrency. Pipelining can be easily introduced to the structure in Fig.6.3b. Each pipeline in a carry-save array needs two latches of word-length to store two outputs while only one bit register is needed to insert one stage of pipeline in the data array.

An extra advantage of pipeline is that a long filter is cut into several shorter sections. Then we can use smaller W-trees which is easier to design. An example using 14-to-2 Wallace-tree is given for a 48-tap filter structure in Fig.6.4. The time required by one accumulation is  $6 t_{fa}$ .

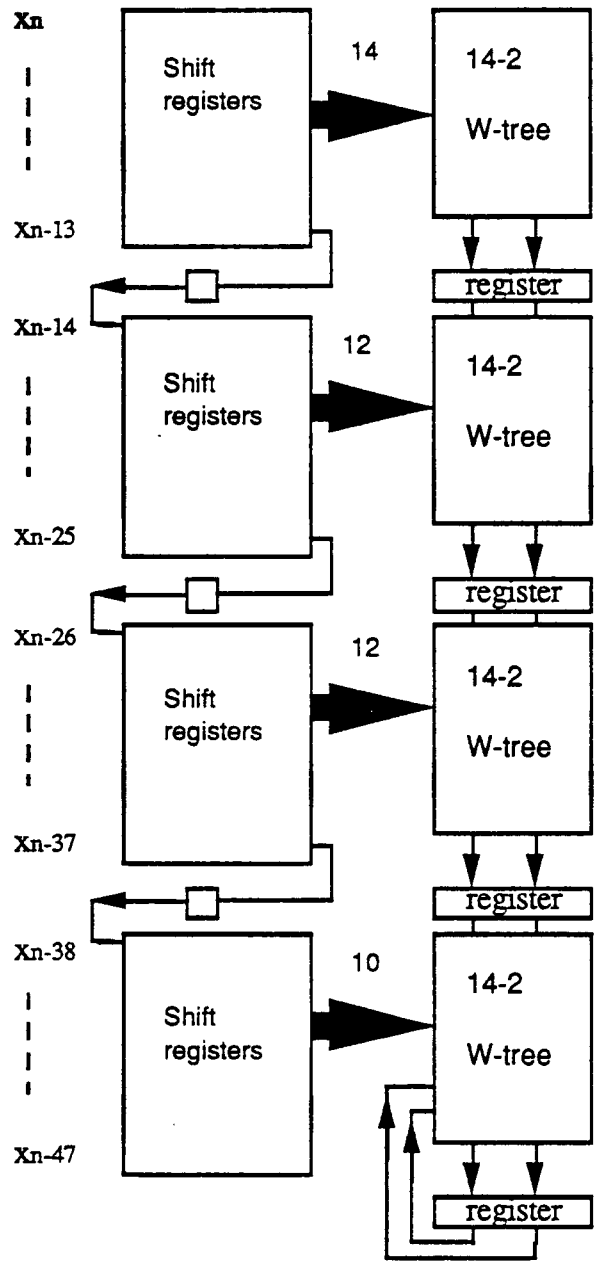


Fig.6.4 Pipeplined structure with Wallace-trees. (filter length=48)

An alternative to the scheme in Fig.6.4 is to insert pipeline every 6 taps while using direct carry-save array. Although the same speed can be reached, it requires more hardware and results in more latency.

While the latency is not critic, we can use the fully pipelined CSA array. A fully pipelined structure is given in Fig.6.5. The resulting maximum throughput of the filter will be  $f_{max}=1/(Bt_{fa})$ , given B the number of bits of an input.

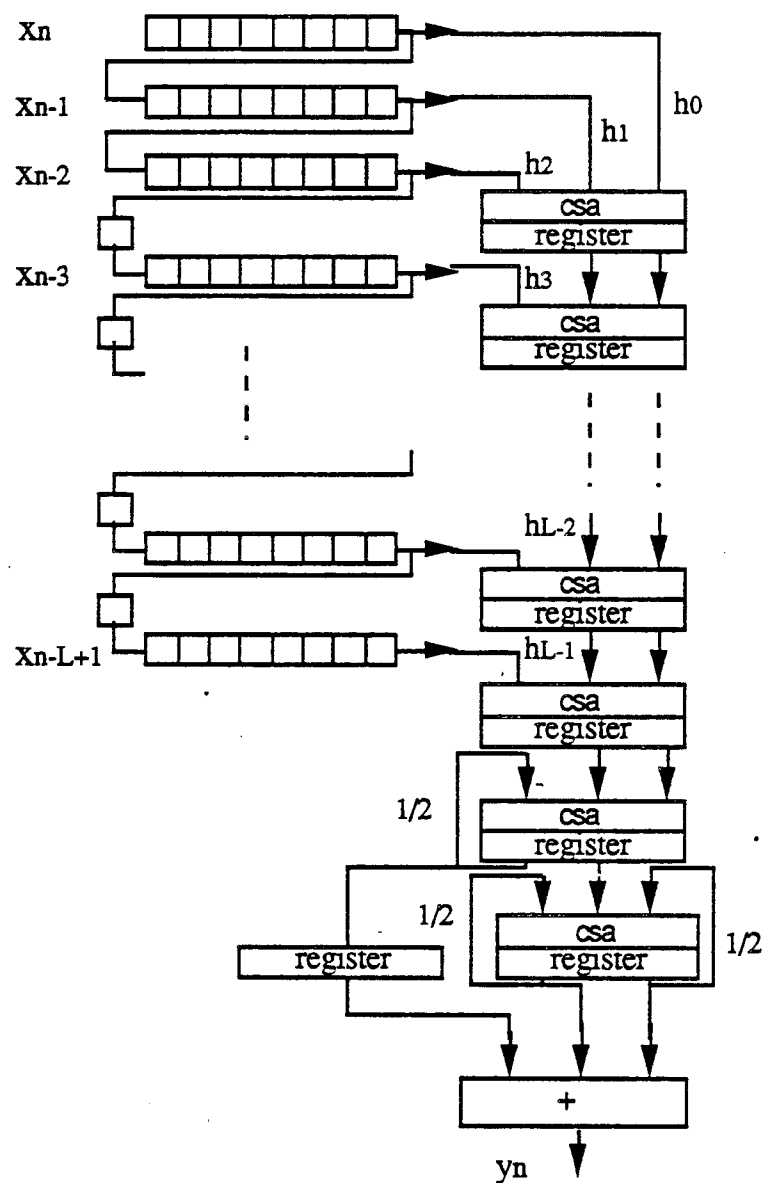


Fig.6.5 Fully pipelined structure

A simple rearrangement may result in twice the speed in case without pipeline or halve the latency in case with pipeline (Fig.6.6). We can still increase the speed or reduce the latency by further dividing the CSA array. Then more efforts should be devoted to the design.

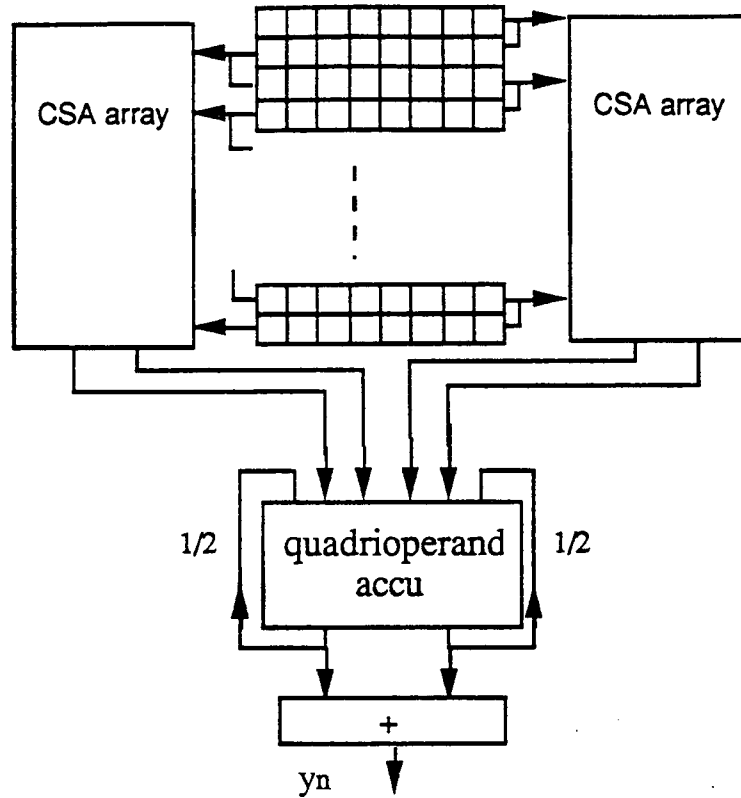


Fig.6.6 Concurrent filter structure.

### 6.2.3 Using the modified Booth's encoding

The modified Booth's encoding (MBE) is a widely-used technique in designing fast multiplier. It allows to halve the number of partial products and the number of computing cycles per output, thus 2 times faster than conventional configuration. Details can be found in the previous chapter. Here we are interested in applying it to filter design.

Then eq.(6.3) can be rewritten as follows under the MBE:

$$\begin{aligned}
 y_n &= \sum_{i=0}^{L-1} h_i 2^{-1} \sum_{k=0, k \text{ even}}^{B-1} (-2x_{n-i, k} + x_{n-i, k+1} + x_{n-i, k+2}) 4^{-k} \\
 &= 2^{-1} \sum_{k=0, k \text{ even}}^{B-1} 4^{-k} \sum_{i=0}^{L-1} (-2x_{n-i, k} + x_{n-i, k+1} + x_{n-i, k+2}) h_i
 \end{aligned} \tag{6.5}$$

Then, the number of accumulations is halved and the speed can be increased 2 times. For a B-bit number, we obtain B/2 partial sums for B even or (B+1)/2 for B odd. The structure based on the MBE is drawn in Fig.6.7.

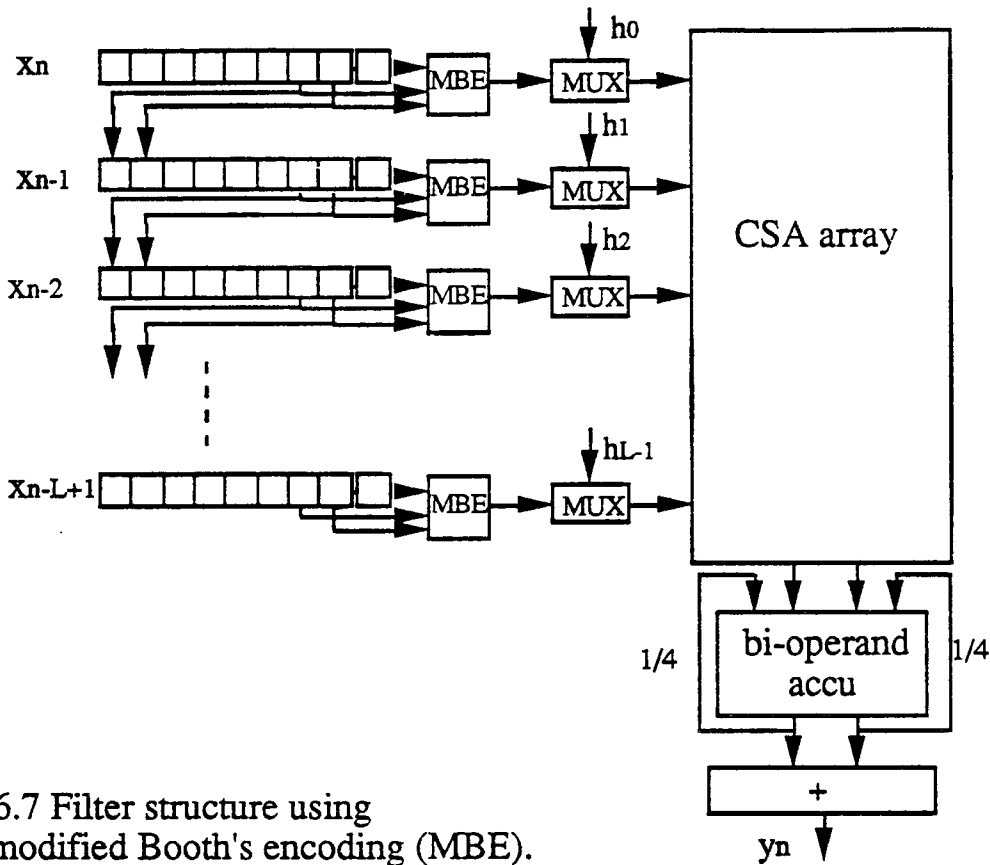


Fig.6.7 Filter structure using the modified Booth's encoding (MBE).

For a dedicated design, the filter's coefficients are constant. The scheme can be much simplified using the technique presented in [Mou89b]. By considering four possible combinations of two consecutive bits, we can write eq.(5.6) under four forms:

$$\begin{aligned}
 p_j(0, 0) &= C \\
 p_j(0, 1) &= (SN) \oplus C \\
 p_j(1, 0) &= (\bar{S}N) \oplus C \\
 p_j(1, 1) &= N \oplus C
 \end{aligned} \tag{6.6}$$

Since S, N and C are the functions of three consecutive bits of X, the following equation may simplify the logic design:

$$\begin{aligned}
 p_j(0, 0) &= x_i \\
 p_j(0, 1) &= x_i(x_{i+1} + x_{i+2}) + x_{i+1}x_{i+2} \\
 p_j(1, 0) &= x_i \oplus x_{i+1} \oplus x_{i+2} \\
 p_j(1, 1) &= \bar{x}_i(x_{i+1} + x_{i+2})
 \end{aligned} \tag{6.7}$$



The resulting configuration is without partial product selection as shown in Fig.6.8. It can be a good candidate for both mask-programmable design and silicon compilation of custom VLSI FIR filters.

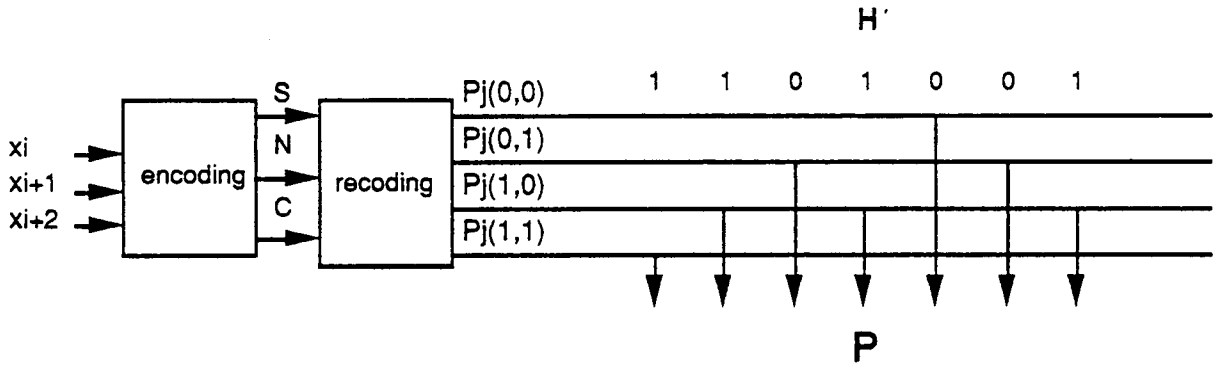


Fig.6.8 Partial product generation when  $H$  is constant. (e.g.  $H=1.01001$ )

### 6.3 Symmetric FIR filters

Linear phase or symmetric filters are the most important ones of FIR filter family. This section centers on their structures.

It is well known that linear phase FIR filters have symmetric coefficients. That means (suppose  $L$  even):

$$h_i = h_{L-i-1}; \quad i = 0, 1, \dots, L/2 - 1 \quad (6.8)$$

Then eq.(6.1) becomes a length- $L/2$  inner product, instead of a length- $L$  one, plus  $L/2$  additions.

$$y_n = \sum_{i=0}^{L/2-1} (x_{n-i} + x_{n+i-L+1}) h_i \quad (6.9)$$

#### 6.3.1 Direct implementation

Since our approach is bit-serial, the  $L/2$  additions can be performed by  $L/2$  serial adders respectively. The inner product is still computed as in general case using ROM [Pel76] as well as the techniques shown before: carry-save adders, W-tree and fast accumulation.

A serial adder is nothing but a full adder plus a carry register. While the additions are computed bit-serially, one more bit register should be added to each input data register as the most significant one. It is initialised by sign extension. Then we need one more computing cycle, i.e.,  $(B+1)$  cycles to compute one output. Fig.6.9 depicts a scheme for such implementation.

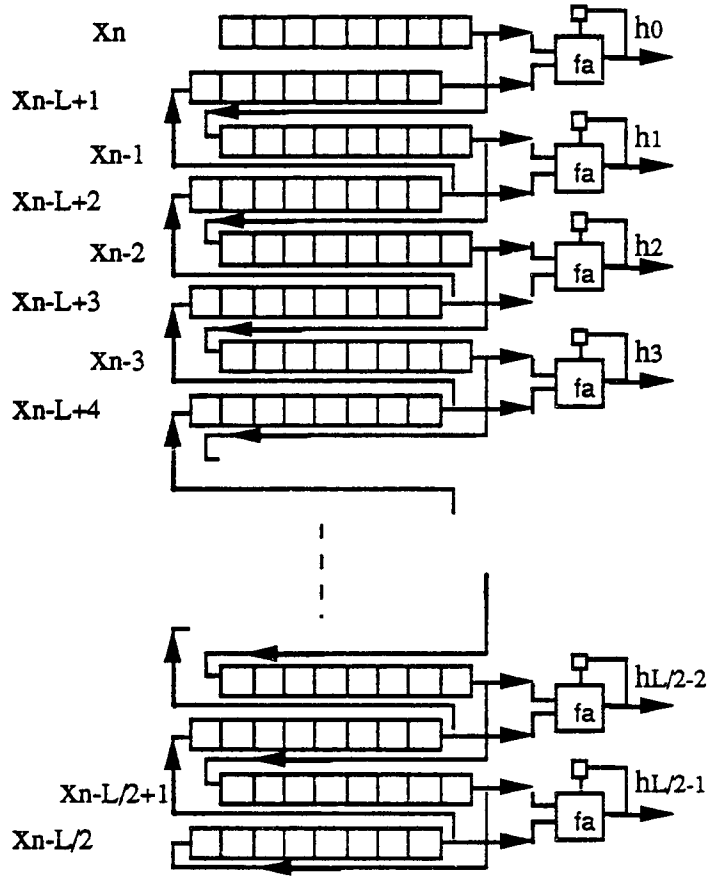


Fig.6.9 Direct implementation of symmetric filter.  
The computing part is omitted.(fa=full adder)

### 6.3.2 Using a new encoding

In bit level,  $(X_1+X_2)H$  is written as follows:

$$\begin{aligned}
 & (X_1 + X_2)H \\
 &= -(x_{1,0} + x_{2,0})H + \sum_{i=1}^{B-1} (x_{1,i} + x_{2,i})H 2^{-i}
 \end{aligned} \tag{6.10}$$

$(x_{1,i}+x_{2,i})$  take one of the three values:  $\{0,1,2\}$ . When  $X$  multiplies  $H$ , three partial products are possible:  $\{0, H, 2H\}$ .

The generation of an effective partial product  $P(i)=(x_{1,i}+x_{2,i})H$  depends on the value of  $(x_{1,i}+x_{2,i})$ . So, we can encode the three values of  $(x_{1,i}+x_{2,i})$  into two control signals: Shift  $S(i)$ , and Null  $N(i)$ . The truth table is given in Table.6.1.

Table.6.1. Truth table of control signal generation for the new encoding.

$x_{1,i}$	$x_{2,i}$	$S(i)$ Shift	$N(i)$ Null
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

From Table.6.1, we have the following relations:

$$\begin{aligned} S &= x_{1,i}x_{2,i} \\ N &= x_{1,i}+x_{2,i} \end{aligned} \quad (6.11)$$

For each bit of the effective partial product, it is determined as below:

$$p_j(h_j, h_{j+1}) = (\bar{S}h_j + Sh_{j+1})N \quad (6.12)$$

This encoding results in  $B$  partial products as usual. Hence it requires  $B$  cycles to compute one output, one cycle less than former schemes at the cost of slightly more complex circuitry. The new encoding scheme is depicted in Fig.6.10.

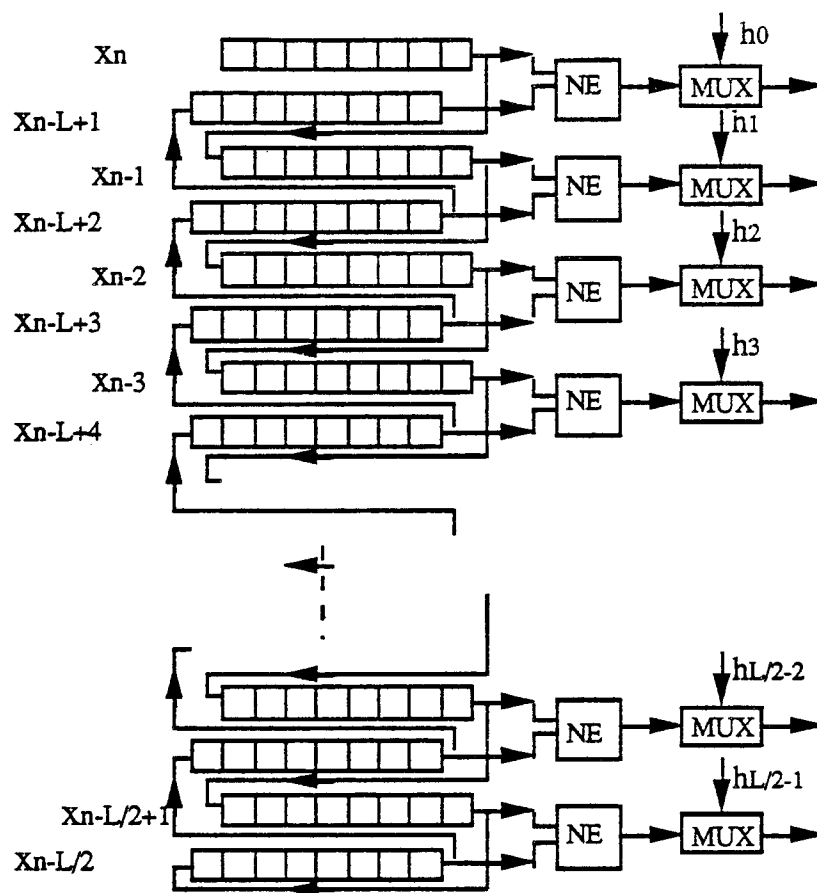


Fig.6.10 Symmetric filter using the new encoding (NE).  
The computing part is omitted.

When the filter's coefficients are constant in a dedicated design, further simplification can be reached in a systematic manner as that for the modified Booth's encoding as shown in Fig.6.8. We give below the corresponding relations:

$$\begin{aligned}
 p_j(0, 0) &= 0 \\
 p_j(0, 1) &= x_{1,i} x_{2,i} \\
 p_j(1, 0) &= x_{1,i} \oplus x_{2,i} = (x_{1,i} + x_{2,i}) \overline{x_{1,i} x_{2,i}} \\
 p_j(1, 1) &= (x_{1,i} + x_{2,i})
 \end{aligned} \tag{6.13}$$

One control signal can be saved since it is equivalent to zero (the ground).

The same technique can be also extended to design antisymmetric filters with  $h_i = -h_{L-1-i}$ . In such case, the three possible partial products are  $\{-H, 0, H\}$  and two control signals should be Complement and Null.

6.3.3 Pipelining

It is of interest to pipeline the symmetric FIR filter structure. It is trivial to pipeline the adder array. As far as pipelining the input data is concerned, a small surprise is that no additional memory is needed. We can do that by simple shift. An example is shown to pipeline a 12-tap symmetric filter (Fig.6.11). Suppose each sample is coded in 8 bits.

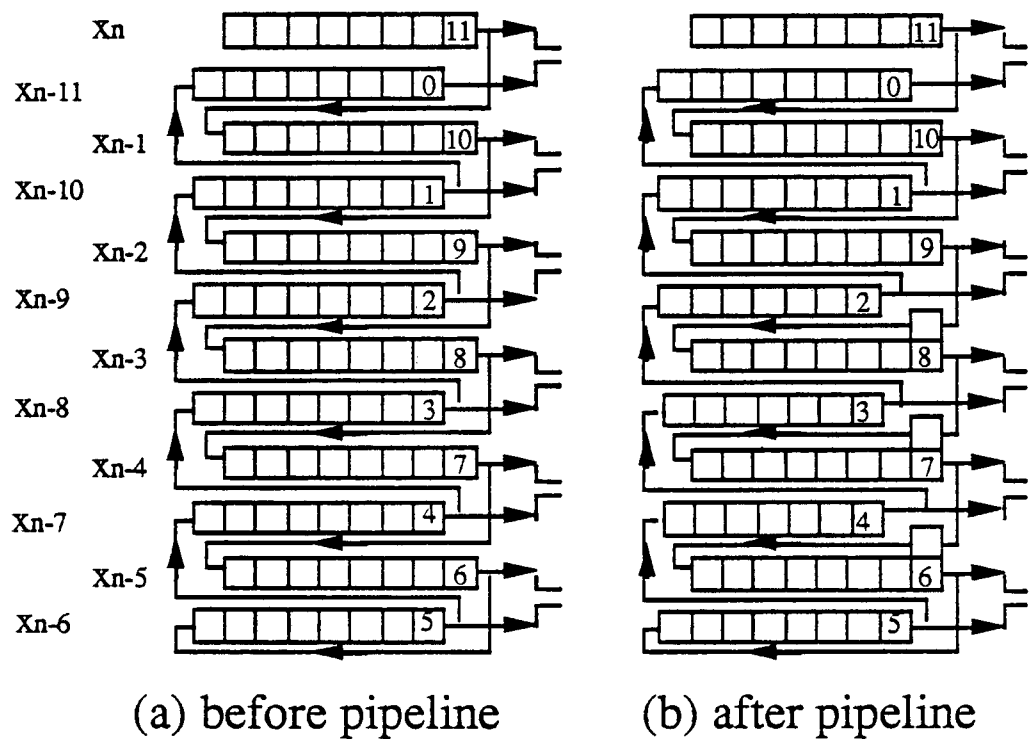


Fig.6.11 Pipelining the shift registers in symmetric filter. Fully pipelined computing part in (b) is omitted.

While without pipeline, we should encode the bit pairs (11,0), (10,1)...(6,5). The positions of bits are indicated in Fig.6.11a.

Since the partial products generated by bit pairs (11,0), (10,1) and (9,2) will be inputs of the carry save adder at the top row, the positions of these bit pairs remain unchanged. The pipeline is only applied to the partial products generated by bit pairs (8,3), (7,4) and (6,5) respectively. It is done by simple shifting bits 8 and 3 by one register along the data flow direction, bits 7 and 4 by two registers and bits 6 and 5 by three registers. After these shifts the wiring

becomes difficult. A regular arrangement is presented in Fig.6.11b to overcome this difficulty, resulting in easy wiring.

## 6.4 Bit-parallel implementation

The modern IC technology allows to integrate more and more transistors. Then bit-parallel implementation of FIR filters becomes feasible. The distributed arithmetic technique can be easily adapted to such implementation. One example can be found in [Li88].

### 6.4.1 Bit-slice approach

The distributed arithmetic technique is originally a bit-serial approach. If we suppose the input is coded in 1 bit, the filter structure can be simplified by using  $L$  registers of one bit, without touching the computing part.

The eq.(6.3) can be written as

$$y_n = -y_{n,0} + \sum_{k=1}^{B-1} y_{n,k} 2^{-k}$$

with

$$y_{n,k} = \sum_{i=0}^{L-1} h_i x_{n-i,k} \quad (6.14)$$

By taking  $y_{n,k}$  as the output of a 1-bit filter, we reconstruct the output of the initial filter through adding the outputs of the  $B$  1-bit filters with appropriate shift (see Fig.6.12). Since all the 1-bit filters are identical, we can repeat the same design.

Using the modified Booth's encoding, the subfilters are no longer 1-bit filters but a little more complex. However, we need only  $B/2$  or  $(B+1)/2$  subfilters. About 50% of hardware (chip area, transistors, etc) can be saved.

### 6.4.2 Symmetric FIR filters

It is impossible to apply the bit-slice approach to the scheme using a serial adder, because we can not split the initial filter into subfilters. In contrast, the scheme using the new encoding allows perfectly the splitting into

1-bit filters, so that we can take advantage of the symmetry in coefficients for bit parallel implementation. Hence it leads also to a saving of 50% of hardware.

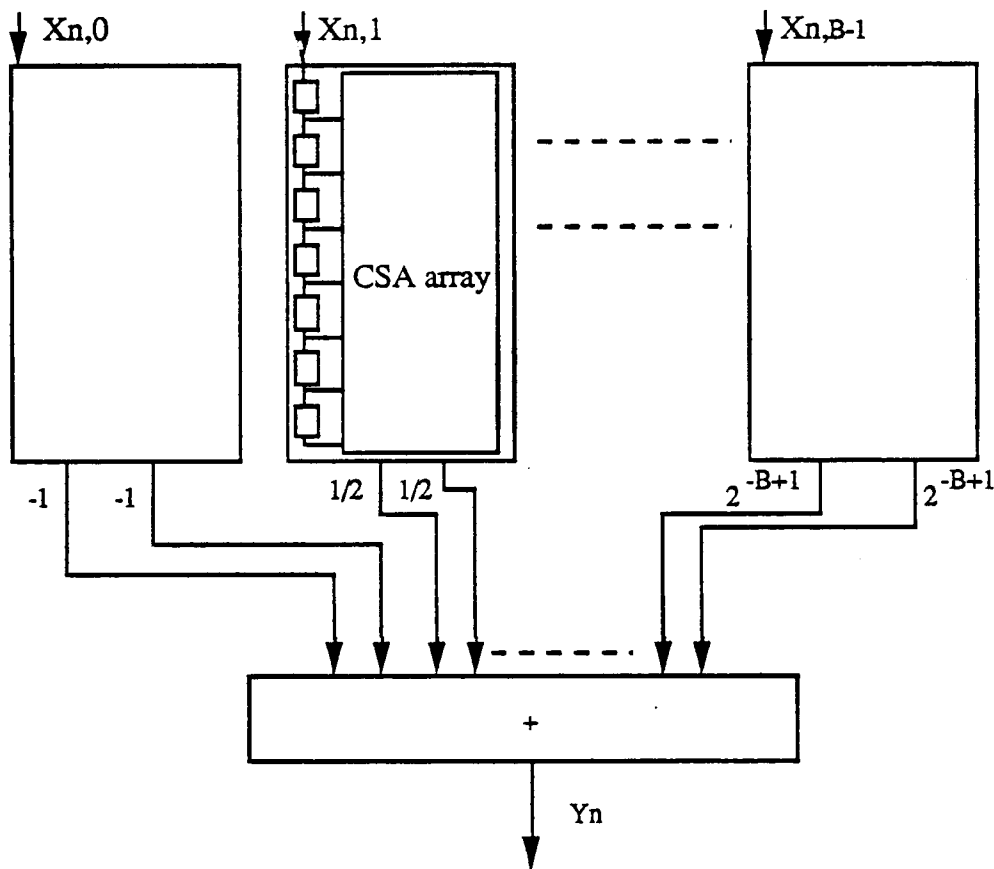


Fig.6.12 Bit-parallel filter structure by distributed arithmetic

There are two ways of achieving 50% of saving above. Firstly, the MBE applies to all kinds of filters with or without linear phase. Its application will prevent from taking advantage of the symmetry. Secondly, the new encoding is very suitable for benefiting from the symmetry. Which is better for a bit parallel symmetric filter design? Our conclusion is that the second is simpler since it enjoys an easier encoding than the MBE.

6.5 Remarks

We have presented several new FIR filter structures by the distributed arithmetic principle, both for bit-serial and bit-parallel implementation.

The time required by an accumulation is drastically reduced due to the fast accumulation. The distributed arithmetic structure is often characterized by using ROM. But the ROM occupies a large area if realized in VLSI and represents a bottleneck to speed. By extremely dividing the ROM we obtain an adder-based structure. This structure is mainly investigated. Carry-save, Wallace-tree and the modified Booth's encoding are shown to increase the speed and to reduce the number of computing cycles. We have shown also how to exploit the concurrency and to pipeline the structure.

Symmetric FIR filter structures are also studied. A new encoding is proposed to take advantage of the symmetry, which allows efficient bit-serial implementation as well as bit-parallel one.

A systematic simplification is proposed for custom VLSI design. The resulting configuration is very suitable for mask programming and silicon compilation.

Although the study is in the context of FIR filters, many presented techniques are applicable to the design of inner product computer and adaptive filters.



## Chapter 7

### Conclusion and perspectives

#### 7.1 Conclusion

This thesis has been concerned with the algorithmic and architectural aspects of implementing the finite impulse response (FIR) filter either in software or in VLSI.

In the study of fast filtering algorithms, we have dealt with the reduction of arithmetic complexity or the number of arithmetic operations while taking into account the interaction between computer architectures and algorithms.

A unified approach to derive all the FIR filtering algorithm has been proposed in chapter 2. It consists of formulation as polynomial products; interpolation; filtering; reconstruction by the chinese remainder theorem; overlap. By choosing carefully the decimating rate in the first step (formulation as polynomial products) and the interpolation points in the second step, we can derive all the variants of FIR filtering algorithms including the conventional ones. In the context of pseudocircularity, we have presented the FIR filtering algorithms as the diagonalization of a pseudocirculant matrix by rectangular matrices and proved the transposability and the equality of arithmetic complexity in both direct and transposed forms. Three most representatives of conventional algorithms (Stockham, Agarwal-Burrus, Winograd) have been reviewed in the light of the approach, and some new possibilities have been presented in chapter 3. Next a class of algorithms of interest is studied in detail in chapter 4. These algorithms not only reduce the number of arithmetic operations but also maintain the multiply-accumulate structure which can be easily implemented in many ways.

The second part is dedicated to the architecture of VLSI implementation of FIR filters. Our attention has been centered on the computer arithmetic, design regularity and high speed implementation.

Firstly, we have reviewed the fast arithmetic and concurrent structure in designing multiplier-accumulators in chapter 5. A number of innovations has been made on the modified Booth's encoding with a constant multiplicand, compact Wallace-tree layout and bit-level pipelined structure. New structures for implementing FIR filter in VLSI by distributed arithmetic are proposed in chapter 6. Our strength has been on the ROMless, carry-save adder based structure. The modified Booth's encoding are applied for reducing the number of computing cycles per output. Concurrency is largely explored using Wallace-tree and pipeline until bit level. Symmetric FIR filter structures are particularly studied, resulting in important savings in hardware requirement. A special encoding technique is suggested for computing operations like  $(X \pm Y)H$  which are essential for symmetric filters. This encoding is not only suitable for efficient bit-serial implementations but shows high potential for bit-parallel ones.

## 7.2 Perspectives

Some directions for future research which seem important to us are proposed below.

### 7.2.1 Multiprocessor implementation of FIR filters

Recently multiprocessor implementations of FIR filters were studied [Mar86,Hay86]. It is shown that a length- $L$  FIR filter can be implemented using  $N \times N$  processors in a SIMD (single-instruction-multiple-data) manner. Each processor computes a length- $L/N$  filter while  $N$  outputs of the filter are computed together. It is not surprising that the maximal throughput can be increased by a factor of  $N \times N$ , since one output is computed in a time proportional to  $L/(N \times N)$ .

However, the number of processors can be reduced using the algorithms in chapter 4. For example, an implementation in [Hay86] is composed of 4 processors while each of them computes a length- $L/2$  filter. One processor can be saved using the radix-2 algorithm. Only 3 processors are needed while the throughput is always increased by a factor of 4. Higher radix algorithm may result in more savings.

### 7.2.2 Application of fast FIR filtering algorithms to LMS adaptive filters

The least-mean-square (LMS) adaptive filter [Wid60] plays an important role in many applications. Block LMS algorithms have been proposed by Clark, Mitra and Parker [Cla81] to accelerate the computation while modifying the original functions. It adapts the filter coefficients every  $N$  steps instead of one adaptation per input while  $N$  is often greater than the filter's order. Then the product of a Toeplitz matrix and a vector can be computed by fast Fourier transform. It is claimed by the authors that similar performance is observed to that of the LMS adaptive filter.

We can easily extend our FIR filtering algorithms to compute the BLMS for  $N \geq 2$  without transform, which is suitable for implementation on DSP's.

However the comparison in [Cla81] was conducted using uncorrelated input noise. It is not sure if the claim is valid under other conditions. Nevertheless no evidence is reported on the superiority of BLMS on the LMS. Recent study [Ben89] shows that the LMS filter outperforms largely the BLMS under some particular noise of interest.

Then the application of BLMS seems limited if the same performance as the LMS algorithm is required.

However, the LMS adaptive filter can be computed in a both fast and exact manner [Duh89], yet by block processing. In fact the BLMS is an approximation to the LMS while ignoring the high order ( $\geq 2$ ) autocorrelations of input signal. Such an approximation can be generalized to ignore higher order ( $\geq n$ ) autocorrelations, resulting in an arithmetic complexity/efficiency trade-off.

The complex LMS filter [Wid75] is required in the processing of high-frequency narrow-band signals. The complex FIR filtering algorithms in chapter 3 are evidently applicable to the computation of the complex LMS filter using the technique in [Duh89]. In case the block length equals 2, 56% of reduction in arithmetic complexity can be expected.

### **7.2.3 Silicon compiler for FIR filter implementation**

Because the FIR filter is widely used in digital signal processing, it is desirable to generate an FIR filter for a given specification using high level language without redesigning all the basic cells. It is very important since the prevailing custom designs are often required for high performance circuits.

We present some considerations for a compiler using the architectures in chapter 6 as a model. We center our attention on the compilation of symmetric FIR filters. The following parameters should be taken into account:

-input wordlength  $b$ : It determines the length of shift registers;

-coefficient wordlength  $n$ : It determines the width of carry-save adders while some guard bits should be anticipated in function of the absolute sum of all the positive coefficients or of all the negative ones.

-degree of pipeline: It depends on the required throughput. Using bit-level pipeline, the maximal throughput is  $1/(b \cdot t_{fa})$ .  $t_{fa}$  is the time required by a full adder. For  $b=8$  and  $t_{fa}=2\text{ns}$  (using 1 micron CMOS),  $f_{max} = 62.5\text{ MHz}$  with system clock frequency = 500 MHz.

-latency: When the latency is critical, some techniques can be used to reduce the delay of system while maintaining high speed: such as combining the small Wallace-tree and pipeline (see Fig.6.4) or dividing the CSA array (see Fig.6.6).

-programmability: For constant coefficients further reduction in material is achieved by using the equation (6.13) and the technique in Fig.6.8.

-bit-serial or bit-parallel or in-between: The compiler should be able to increase the throughput of the filter by treating more than one bit per clock period. The task can be relieved by using the bit-slice design (see Fig.6.12). By doing so, it is also possible to lower the proportion between system clock frequency and the input sampling rate.

## Appendix

### Fixed-point error analysis of short-length FIR filtering algorithms

Digital representation is just an approximation of values in reality. In digital computers, the values are approximated by a finite number of binary digits. The arithmetic operators therein can manipulate only operands of finite number of bits. The results of more bits than the length of internal register should be reduced. Such reduction is in general called quantization.

From a point of view of implementation, it is important to know the characteristics of the quantization errors. In general, there are three sources of quantization errors: A/D noise, quantization of coefficients and quantization in arithmetic operations. The A/D noise depends on only the transfer function of the digital filter. It is already well discussed in the literature [Chan73]. Hence, we will study only the effect of the other two sources of errors.

In the following, we first present our assumption on the computing environment, the properties of noises and the manner of normalization on the the filter's coefficients.

Then examples are given to establish the error models for the analysis in the algorithms. Both direct and transposed forms of algorithms are studied and compared. Further optimization is also pointed out.

Finally, we present the study on the effect of coefficient quantization in these algorithms and compared it to the direct implementation.

#### 1. Major assumptions

As mentioned before, the short length FIR filtering algorithms are suitable for many implementations. A typical application is on the widely used Digital Signal Processors (DSP's). Then our analysis will be based on the characteristics of typical DSP's with 2's complement arithmetic. Suppose the wordlength of internal registers is  $b$ . Consequently, both input and coefficients

have  $b$  bits, as well as the output. Moreover most DSP's have an accumulator of at least  $2b$  bits. Several garde bits are often provided to prevent from overflow [Lee88]. So we can consider that a multiply can be done in full precision. A scalar product by successive multiply-accumulation is also considered as in full precision if only the  $b$  most important bits of the  $2b$  bits in the accumulator are retained by quantization.

There exist three kinds of quantization in those DSP's [Mey89]: truncation, rounding and convergent rounding, where the third one performs rounding to nearest even number (i.e.  $2.5 \rightarrow 2$  and  $3.5 \rightarrow 4$ ). The convergent rounding is a hardware-implemented feature of Motorola's DSP 56000 signal processor.

In Table.1, the mean value  $\mu$  and the variance  $\sigma^2$  of the error due to different quantization methods are listed for:

- a) scaling with  $1/2$  (one bit shift to the right).
- b) wordlength reduction of  $2b$  bits to  $b$  bits.

Table.1 Mean and variance of the error for different quantization methods with 2's complement arithmetic.  $Q=2^{-(b-1)}$ .

	a)		b)	
	$\mu/Q$	$\sigma^2/Q$	$\mu/Q$	$\sigma^2/Q$
Truncation	-1/4	1/16	-1/2	1/12
rounding	1/4	1/16	0	1/12
convergent rounding	0	1/8	0	1/12

All the following quantizations are supposed to be rounding. This will not lost generality for other kinds of quantization error.

It is assumed that the quantization errors at different locations or at the same location but at different instants are independent of each other. All errors are uncorrelated with the signal. Then the error variance at the output can be calculated separately for each error sources and the results superposed.

We suppose that the filter transfer function is normalized so that

$$\sum_{i=0}^{L-1} |h_i| = 1 \quad (1)$$

The dynamics of input  $X(z)$  is limited to 1, namely,  $|x_i| \leq 1$ . By consequence, we have the dynamics of output  $|y_i| \leq 1$ .

## 2. Quantization error in direct FIR filter computation

Since DSP's provide at least a 2b-bit accumulator, it is not necessary to quantize every result of data-coefficient multiplication. Only one quantization is performed at the output of the filter as shown in Fig.1. The mean and variance of error found in Table 1. For example, if rounding is performed, we obtain  $\mu=0$  and  $\sigma^2=\sigma_d^2=Q^2/12$ . The subscript d means direct computation.

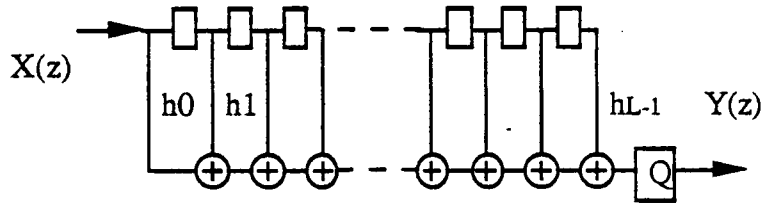


Fig.1 Quantization in direct FIR filter computation

As for coefficient quantization error, we assume the transfer function  $H(z)$  is replaced by  $H(z)+N(z)$ . It is easy to show that the corresponding error at output is  $X(z)N(z)$ . This term is presented for the following comparison. The analysis in detail on this aspect can be found in [Chan73].

## 3. Illustrative example for error analysis in short-length FIR filtering algorithms

We give as an illustrative example an error analysis on the direct and transposed radix-2 algorithms. The same methods can be generalized to analyze other algorithms.

### 3.1 Analysis for direct radix-2 algorithm

We first establish an error model, as shown in Fig.2, for the analysis.  $e_0$  is an error due to scaling by  $1/2$ . Since all the successive multiplications are supposed to be done in full precision, the only quantizations occur at the outputs of the subfilters  $H_0(z)$ ,  $H_0(z)+H_1(z)$ ,  $H_1(z)$ , which are the

responsible of errors  $e_1$ ,  $e_2$  and  $e_3$  respectively. Let  $\hat{e}_3$  denotes  $z^{-2}e_3$ , i.e., the error in computing two previous outputs. It is also assumed that  $\hat{e}_3$  and  $e_3$  are uncorrelated and independent of each other. There are not scaling after the output addition since the output dynamics is limited to 1.

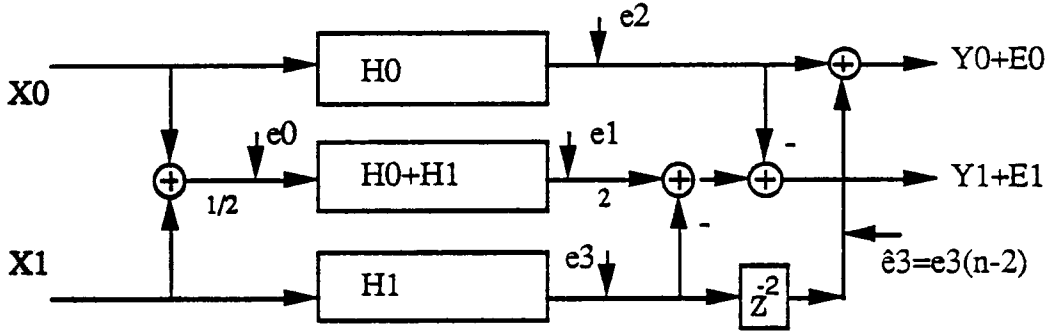


Fig.2 Quantization model for direct radix-2 algorithm

It is then easy to show

$$E_0 = e_2 + \hat{e}_3$$

$$E_1 = 2e_0 |H_0(z^2) + H_1(z^2)| + 2e_1 + e_2 + e_3 \quad (3)$$

Since  $|H_0(z^2) + H_1(z^2)| \leq 1$  after normalization, then  $E_1 \leq 2e_0 + 2e_1 + e_2 + e_3$ .

The mean and variance of the error are respectively:

$$\mu = E[(E_0 + E_1)/2] \leq E[e_0 + e_1 + e_2 + e_3/2 + \hat{e}_3/2] = Q/4$$

$$\begin{aligned} \sigma^2 &\leq \left( \frac{1}{16} + 2 \times \frac{1}{12} + \frac{1}{4} \frac{1}{12} + \frac{1}{4} \frac{1}{12} \right) Q^2 \\ &= \frac{13}{4} \frac{1}{12} Q^2 = \frac{13}{4} \sigma_d^2 \end{aligned}$$

### 3.2 Analysis for transposed radix-2 algorithm

Transposing the signal flowgraph in Fig.2 does not change the transfer function but the procedure of computing as well as the location of quantization as shown in Fig.3.

$e_0$  and  $e_1$  are roundoff errors due to the scaling by 1/2.  $e_2$ ,  $e_3$  and  $e_4$  are the roundoff errors at the outputs of the subfilters. At the output, we get:

$$E_0 = 2(e_0 |H_0(z^2)| + e_2) + e_4$$

$$E_1 = 2(e_1 |H_1(z^2)| + e_3) + e_4$$



Noting  $|H_0(z^2)| + |H_1(z^2)| \leq 1$  and  $|H_0(z^2)|^2 + |H_1(z^2)|^2 \leq 1$ , we obtain then the mean and variance of errors at the output:

$$\mu = E[(E_0 + E_1)/2] = E[e_0 |H_0(z^2)| + e_1 |H_1(z^2)| + e_2 + e_3 + e_4] \leq Q/4$$

$$\sigma^2 \leq \left(\frac{1}{16} + 3 \times \frac{1}{12}\right) Q^2 = \frac{15}{4} \frac{1}{12} Q^2 = \frac{15}{4} \sigma_d^2$$

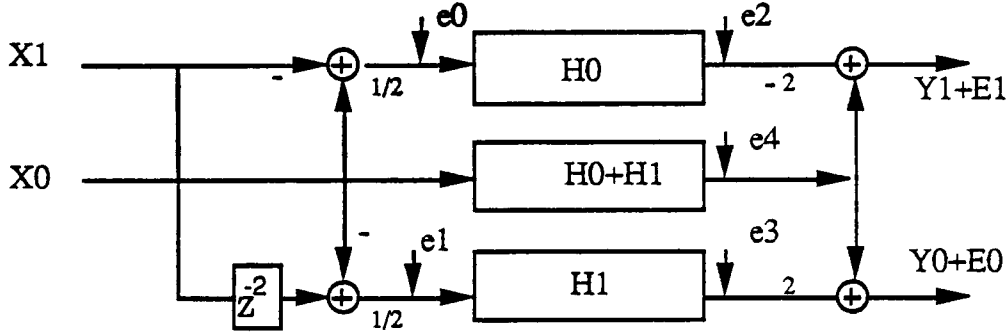


Fig.3 Quantization model for transposed radix-2 algorithm

### 3.3 Consideration for reduction of errors

Both direct and transposed radix-2 algorithms produce more errors than conventional implementation. This is the price we should pay for reducing the number of operations. Moreover the transposed algorithm generates slightly more errors than direct one does. However by carefully examining the procedure of computation and the amplitude of the subfilters' transfer function, we can further reduce the quantization error.

Firstly, in the direct radix-2 algorithm as shown in fig.2, we can compute the filters  $H_0(z^2)$  and  $H_1(z^2)$  before  $H_0(z^2) + H_1(z^2)$ . The outputs of  $H_0(z^2)$  and  $H_1(z^2)$  are quantized and saved in b-bit registers while the output of  $H_0(z^2) + H_1(z^2)$  in the 2b-bit accumulator. Hence the quantization can be performed after the scaling by 2 and after the addition with the other two terms. Then  $E_1$  is modified as:

$$E_1 = 2e_0 |H_0(z^2) + H_1(z^2)| + e_1 + e_2 + e_3$$

The variance is changed accordingly:

$$\begin{aligned} \sigma^2 &\leq \left(\frac{1}{16} + \frac{1}{12} + \frac{1}{4} \frac{1}{12} + \frac{1}{4} \frac{1}{12} + \frac{1}{4} \frac{1}{12}\right) Q^2 \\ &= \frac{10}{4} \frac{1}{12} Q^2 = \frac{5}{2} \sigma_d^2 \end{aligned}$$

Similar consideration is given to the computation of the transposed algorithm. We first compute the subfilter  $H_0(z^2)+H_1(z^2)$  and save the output in a b-bit register after quantization. For computing  $y_0$ , we can calculate the output of  $H_0(z^2)$  and quantize it after scaling since it is located in the 2b-bit accumulator. By adding it with the output of  $H_0(z^2)+H_1(z^2)$ , we get  $y_0+E_0$ . We can compute  $y_1$  in the same way. Then

$$E_0 = 2e_0 | H_0(z^2) | + e_2 + e_4$$

$$E_1 = 2e_1 | H_1(z^2) | + e_3 + e_4$$

Then the variance becomes:

$$\sigma^2 \leq \left( \frac{1}{16} + \frac{1}{4} \frac{1}{12} + \frac{1}{4} \frac{1}{12} + \frac{1}{12} \right) Q^2 = \frac{9}{4} \frac{1}{12} Q^2 = \frac{9}{4} \sigma_d^2$$

After optimization, we find that the transposed algorithm is slightly better than the direct one (9/4 compared to 5/2). Compared to the conventional implementation, one more bit allocation is largely sufficient for reducing the error to the same level.

We note that the scaling errors due to input additions generate always the same mean ( $Q/4$ ) and variance ( $Q^2/16$ ) in both direct and transposed algorithms. Whether this can be generalized to higher-radix algorithms should be further examined.

### 3.4 Error analysis for quantization of coefficients

After quantization, the transfer functions of subfilters are approximated by  $H_0+N_0$ ,  $H_0+H_1+N_1$ ,  $H_1+N_2$ , as assumed in [Vet88]. Then the algorithm becomes

$$Y_0 = X_0(H_0+N_0) + z^{-2}X_1(H_1+N_2)$$

$$Y_1 = (X_0+X_1)(H_0+H_1+N_1) - X_0(H_0+N_0) - X_1(H_1+N_2)$$

The computation of  $Y_0$  is equivalent to conventional implementation. But that of  $Y_1$  is different. We can further develop the computation of  $Y_1$  as:

$$Y_1 = (X_0+X_1)(H_0+H_1) - X_0H_0 - X_1H_0 + X_0(N_1-N_0) + X_1(N_1-N_2)$$

while the conventional computation is equivalent to the following:

$$Y_1 = X_0(H_1+N_2) + X_1(H_0+N_0)$$

$$=X_0H_1+X_1H_0+X_0N_2+X_1N_0$$

The error elements in two different computations are respectively:  $X_0(N_1-N_0)+X_1(N_1-N_2)$  and  $X_0N_2+X_1N_0$ . Assume that  $N_0$ ,  $N_1$  and  $N_2$  have the same variance. Then the first term has greater dynamic range than the second one. It means that the radix-2 algorithm is more sensitive to the coefficient quantization effect than the conventional computation.

#### 4. Conclusion

The quantization effects are analyzed for the short length FIR filtering algorithms. The analysis is based on the radix-2 algorithm. But the methodology can be generalized to the analysis of higher-radix algorithms without difficulty. We found that the radix-2 algorithm needs no more than one more bit to reach the same error level as conventional computation. The transposed and direct algorithms generate nearly the same error. The procedure of computation has important influence on the error level. Further reduction of error can be performed by efficient use of the 2b-bit accumulator. Using double precision computation may reduce once more the errors. The double precision is only necessary in the output additions. Then the cost in time may not be important.

An observation is that the radix-2 algorithm results in an increased sensibility to coefficient quantization effect than conventional computation.

## References

### Algorithms:

- [Aga74] R.C.Agarwal, C.S.Burrus, "Fast one-dimensional digital convolution by multi-dimensional techniques", IEEE Trans. on ASSP, Vol.ASSP-22, No.1, January 1974, pp.1-10.
- [Aga77] R.C.Agarwal, J.W.Cooley, "New algorithms for digital convolution", IEEE Trans. on ASSP, Vol.ASSP-25, No.5, October 1977, pp.392-410.
- [Bal86] P.C.Balla, A.Antoniou, S.D.Morgera, "Higher radix aperiodic convolution algorithms", IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-34, N°1, February 1986, pp.60-68.
- [Bel74] M.G.Bellanger, J.L.Daquet, "TDM-FDM Transmutiplexer: Digital Polyphase and FFT", IEEE Trans. on Communication, Vol-22, No.9, Sept. 1974, pp.1199-1204.
- [Ben89] J.Benesty, private communication.
- [Bla85] R.E.Blahut, Fast Algorithms for Signal Processing, Addison-Wesley, Reading, MA, 1985.
- [Cha73] D.S.K.Chan and L.R.Rabiner, "Analysis of quantization errors in the direct form for finite impulse response digital filters," IEEE Trans. Audio Electroacoust., Vol.AU-21, No.4, Aug. 1973, pp.354-366.
- [Cla81] G.A.Clark, S.K.Mitra, S.R.Parker, "Block implementation of adaptive digital filters", IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-29, June 1981, pp.744-752.
- [Coo65] J.W.Cooley, J.W.Tukey, "An algorithm for the machine calculation of complex Fourier series", Math. of Comput., Vol.19, pp.297-301, April 1965.
- [Cro83] R.E.Crochiere, L.R.Rabiner, Multirate Digital Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1983.
- [Duh87] P.Duhamel, M.Vetterli, "Improved Fourier and Hartley Transform Algorithms: Application to Cyclic Convolution of Real Data", IEEE Trans. on ASSP, Vol.ASSP-35, No.6, June 1987, pp.818-824.
- [Duh89] P.Duhamel, Z.J.Mou, J.Benesty, "Une présentation unifiée du filtrage rapide fournissant tous les intermédiaires entre traitements temporels et fréquentiels", Proc. of 12th GRETSI Symposium, Juan-les-Pins France, June 1989.
- [Hay86] K.Hayashi, K.K.Dhar, K.Sugahara, K.Hirano, "Disign of high speed digital filters suitable for multi-DSP implementation", IEEE Trans. Circuits Syst., Vol.33, pp.202-216, February 1986.

- [Kwa87] H.K.Kwan, M.T.Tsim, "High Speed 1-D FIR Digital Filtering Architecture using Polynomial Convolution", Proc.ICASSP 87, Dallas, USA, pp.1863-1866.
- [Lee88] E.A.Lee, "Programmable DSP Architectures: Part I", IEEE ASSP Magazine, Oct.1988, pp.4-19.
- [Mar86] T.G.Marshall, Jr, "Transform methods for developing parallel algorithms for cyclic block signal processing", Proc. Int. Conf. Commun., Toronto, Canada, June 1986, pp.288-294.
- [Mey89] R.Meyer, "Error analysis and comparison of FFT implementation structures", Proceedings of ICASSP89, May 1989, pp.888-891.
- [Mou87] Z.J.Mou, P.Duhamel, "Fast FIR Filtering: Algorithms and Implementations", Signal Processing, Dec. 1987, pp.377-384.
- [Mou88] Z.J.Mou, P.Duhamel, "A unified approach to the fast FIR filtering algorithms", Proc. IEEE ICASSP, New York, USA, April 1988, pp.1914-1917.
- [Mou89] Z.J.Mou, P.Duhamel, "Short length FIR filters and their use in fast non recursive filterings", submitted to IEEE Trans. ASSP, 1989.
- [Nus81] H.J.Nussbaumer, Fast Fourier Transform and Convolution Algorithms, Springer, Berlin/New York, 1981.
- [Por80] M.R.Portnoff, "Time-Frequency Representation of Digital Signals and Systems Based on Short-Time Fourier Analysis", IEEE Trans. on ASSP, Vol.ASSP-28, Feb.1980, pp.55-69.
- [Sod86] M.A.Soderstrand, W.K.Jenkins, G.A.Jullien, F.J.Taylor, eds., Residue Number System Arithmetic: Modern applications in digital signal processing, IEEE Press, New York, pp.1-2,1986.
- [Sor88] H.V.Sorensen, D.L.Jones, M.T.Heideman, C.S.Burrus, "Real-valued fast Fourier transform algorithms", IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-35, N°6, June 1988, pp.849-863.
- [Sto66] T.G.Stockham, "High Speed Convolution and Correlation", Proc.1966 Spring Joint Comput. Conf., AFIPS, Vol.28, 1966, pp.229-233.
- [Vai88] P.P.Vaidyanathan, S.K.Mitra, "Polyphase networks, block digital filtering, LPTV systems and alias-free QMF banks: A unified approach based on pseudocirculants," IEEE Trans. Acoust. Speech, Signal Processing, Vol. 36, N°3, March 1988, pp.381-391.
- [Vet88] M.Vetterli, "Runing FIR and IIR Filtering Using Multirate Filter Banks", IEEE Trans. on ASSP, Vol.ASSP-36, No.5, May 1988, pp.730-738.
- [Wid60] B.Widrow, M.E.Hoff, Jr., "Adaptive switching circuits", in 1960 IRE WESCON Conv. Rec., part 4, p.96-104.

- [Wid75] B.Widrow, J.McCool, M.Ball, "The complex LMS algorithm", Proceedings of IEEE, April 1975, p.719-720.
- [Win77] S.Winograd, "Some bilinear forms whose multiplicative complexity depends on the field of constants", Math. Syst. Theory, Vol.10, No.2, pp.169-180, 1977.
- [Win80] S.Winograd, "Arithmetic Complexity of Computation", CBMS-NSF Regional Conf. Series in Applied Mathematics, SIAM publications, No.33, 1980.

### Architectures:

- [And67] S.Anderson, J.Earle, R.Goldschmidt, D.Powers, "The IBM 360 model 91: Floating point execution unit", IBM J. Res. Develop., Vol.11, January 1967.
- [Boo51] A.D.Booth, "A signed binary multiplication technique", Qt. J. Mech. Appl. Math., Vol.4, 1951.
- [Bur77] C.S.Burrus, "Digital filter structures described by distributed arithmetic", IEEE Trans. Circuits Syst., Vol.24, pp.674-680, Dec. 1977.
- [But76] M.Buttner, H.W.Shusseler, "On structures for the implementation of the distributed arithmetic", Nachrichtentech. Z., Vol..29, pp.472-477, June 1976.
- [Che85] C.F.Chen, "Implementing FIR filters with distributed arithmetic", IEEE Trans. ASSP, Vol.33, pp.1318-1321, October 1985.
- [Cro73] A.Croisier, D.J.Esteban, M.E.Levilion, V.Riso, "Digital filter for PCM encoded signals", U.S. patent, No.3 777 130, Dec. 3, 1973.
- [Duh88] P.Duhamel, Z.Mou, M.Cand, "Multiplieur généralisé mettant en oeuvre un filtre numérique", French patent pending, filed in 1988.
- [Hat77] M.Hatamian, G.L.Cash, "Parallel bit-level pipelined VLSI designs for high speed signal processing", Proceedings of the IEEE, Vol.75, pp.1192-1202, No.9, Sept. 1987.
- [Li87] W.Li, J.Burr, "Parallel multiplier accumulator using 4-2 adders", U.S. patent pending, filed in 1987.
- [Li88] W.Li, J.Burr, A.Peterson, "A fully parallel VLSI implementation of distributed arithmetic", Proceedings of Int. Symp. Circuits and Systems, May, 1988, Espoo, Finland.
- [Mac61] O.L.MacSorley, "High speed arithmetic in binary computers", Proc. IRE, Vol.49, January 1961, pp.67-91.
- [Mon88] L.Montperrus, "Etude d'une famille d'additionneurs et de multiplieurs", Thèse de Doctorat de l'Université de Paris XI, November, 1988.

- [Li87] W.Li, J.Burr, "Parallel multiplier accumulator using 4-2 adders", U.S. patent pending, filed in 1987.
- [Li88] W.Li, J.Burr, A.Peterson, "A fully parallel VLSI implementation of distributed arithmetic", Proceedings of Int. Symp. Circuits and Systems, May, 1988, Espoo, Finland.
- [Mac61] O.L.MacSorley, "High speed arithmetic in binary computers", Proc. IRE, Vol.49, January 1961, pp.67-91.
- [Mon88] L.Montperrus, "Etude d'une famille d'additionneurs et de multiplieurs", Thèse de Doctorat de l'Université de Paris XI, November, 1988.
- [Mou89b] Z.J.Mou, W.Li, "Fast arithmetic for computing  $\pm(W \pm X \pm Y \pm Z)A$ ", submitted for publication, 1989.
- [Pel74] A.Peled, B.Liu, "A new hardware realization of digital filters", IEEE Trans. ASSP, Vol.22, pp.456-462, Dec.1974.
- [Pel76] A.Peled, B.Liu, Digital Signal Processing Theory, Design, and Implementation, John Wiley & Sons, New York, 1976.
- [Rub75] L.P.Rubinfield, "A proof of the modified Booth's algorithm for multiplication", IEEE Trans. on Computers, pp.1014-1015, October 1975.
- [San88] M.Santoro, M.Horowitz, "A pipelined 64x64b iterative array multiplier", ISSCC Digest of Technical papers, pp.36-37, Feb. 1988. Also in IEEE J. Solid-State Circuits, Vol.24, pp.487-493, No.2, April 1989.
- [Vui83] J.Vuillemin, "A very fast multiplication algorithm for VLSI implementation", the VLSI Journal INTEGRATION, Vol.1, pp.39-52, April, 1983.
- [Wal64] C.S.Wallace, "A suggestion for a fast multiplier", IEEE Trans. Electronic Computers, Vol.13, pp.14-17, February 1964,.
- [Was82] W.Waser, M.J.Flynn, Introduction to Arithmetic for Digital Systems Designers, Holt, Rinehart, and Winston, New-York, 1982.
- [Zur86] D.Zuras, W.H.McAllister, "Balanced delay trees and combinatorial division in VLSI", IEEE J. Solid-State Circuits, Vol.21, pp.814-819, No.5, May 1986.

Version Concise en Français



# FILTRAGE RIF RAPIDE

## ALGORITHMES ET ARCHITECTURES

### 1.Introduction

Nous commençons par rappeler différentes représentations du filtre numérique à Réponse Impulsionnelle Finie ou du filtre RIF.

Une première représentation possible est celle en produit scalaire:

$$y_n = \sum_{i=0}^{L-1} h_i x_{n-i} \quad n = 0, 1, \dots, \infty \quad (1)$$

ou en notation vectorielle:

$$y_n = [x_n \quad x_{n-1} \quad x_{n-2} \quad \dots \quad x_{n-L+2} \quad x_{n-L+1}] \cdot \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \\ h_{L-1} \end{bmatrix} \quad (2)$$

L étant l'ordre du filtre (nombre de coefficients).

La transformée en Z de l'eq.(1) fournit une deuxième représentation :

$$Y(z) = H(z)X(z) \quad (3)$$

ou

$$\begin{aligned} & y_0 + y_1 z^{-1} + y_2 z^{-2} + \dots \\ &= (h_0 + h_1 z^{-1} + h_2 z^{-2} + \dots + h_{L-1} z^{-(L-1)})(x_0 + x_1 z^{-1} + x_2 z^{-2} + \dots) \end{aligned} \quad (4)$$

On remarque que l'équation (4) est le produit d'un polynôme de degré (L-1) et d'un autre d'ordre infini.

Les graphes de fluence fournissent une troisième représentation graphique (voir Fig.1).

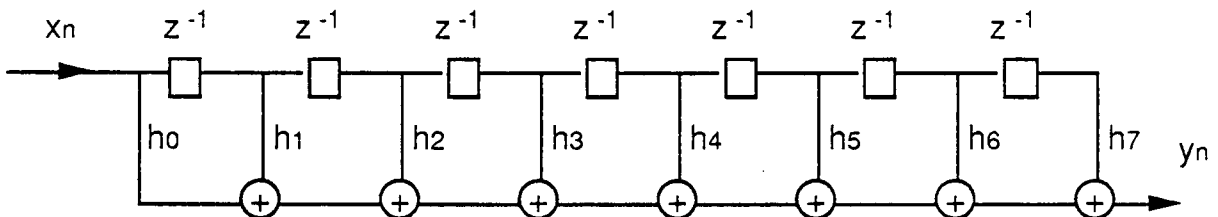


Fig.1 Le filtre numérique à réponse impulsionnelle finie (RIF).

Ces trois représentations seront utilisées tour à tour dans la suite de ce texte.

Le filtre RIF joue un rôle très important dans le traitement numérique du signal. Il possède un ensemble de propriétés intéressantes:

- 1) Il permet le filtrage à phase linéaire;
- 2) Il est toujours stable;
- 3) On peut le réaliser facilement;
- 4) Il permet d'approximer une réponse fréquentielle quelconque.

Mais son défaut majeur est la grande quantité de calcul qu'il représente. Il nécessite souvent la charge de calcul principale dans beaucoup de systèmes de traitement du signal.

Depuis la redécouverte de la transformée de Fourier rapide (FFT) [Coo65] de nombreux algorithmes permettant de réduire la charge de calcul ont été mis en évidence. L'algorithme par convolution cyclique utilisant la FFT [Sto67] est le plus connu. Des algorithmes ne faisant pas appel à la FFT ont également été proposés [Aga74, Win80], leur intérêt étant de permettre un calcul plus efficace des filtres de longueur moyenne ( $\leq 64$ ). Tous ces algorithmes ont pour objectif de réduire le nombre de multiplications dont le temps d'exécution était beaucoup plus important que celui d'une addition dans les processeurs à usage général à l'époque où ils ont été proposés.

L'évolution de la technologie semiconducteur a bouleversé le monde scientifique. Nous avons aujourd'hui des nouveaux moyens de calcul beaucoup plus puissants, plus flexibles ou bien plus spécialisés qu'il y a seulement quelques années. Cette évolution est en train de changer la façon dont nous appréhendons la complexité de calcul. Par exemple le temps d'une multiplication n'est plus dominant dans les calculateurs à usage général. D'autre part, les processeurs de traitement numérique du signal (DSP's) récemment apparus présentent un certain parallélisme: une multiplication plus une accumulation coûte seulement un temps de cycle c'est à dire le même prix qu'une multiplication ou une addition toute seule. La différence entre le critère ayant conduit à proposer les algorithmes classiques de filtrage RIF et le type de machines sur lequel ils sont maintenant le plus souvent implantés explique qu'ils ne soient véritablement efficaces ni sur ces DSP ni en circuits intégrés (VLSI).

C'est pourquoi il est nécessaire de rechercher de nouveaux algorithmes qui tiennent compte de l'architecture des processeurs. Nous avons aussi besoin de nouvelles architectures de filtre RIF qui permettent d'une part une implantation facile en VLSI et d'autre part un traitement rapide, les fréquences d'échantillonnage des signaux à filtrer ne cessant pas de croître.

Notre objectif est donc de rechercher, d'une part, de nouveaux algorithmes qui non seulement réduisent la complexité de calcul mais aussi maintiennent la structure 'multiplication-accumulation', et d'autre part, de nouvelles architectures, pour l'implantation en VLSI, pouvant servir de support à ces algorithmes, mais permettant également l'intégration efficace d'un filtre quelconque.

Dans la présentation suivante, les deux parties correspondantes 'algorithmes' et 'architectures' sont précédées d'un exemple simple permettant de se rendre compte qu'un tel objectif est réaliste.

## 2. Exemple

Supposons que nous calculons 2 sorties successives d'un filtre RIF à L coefficients, avec L pair :

$$\begin{bmatrix} y_n \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} x_n & x_{n-1} & x_{n-2} & \dots & x_{n-L+2} & x_{n-L+1} \\ x_{n-1} & x_{n-2} & x_{n-3} & \dots & x_{n-L+1} & x_{n-L} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \\ \vdots \\ h_{L-1} \end{bmatrix} \quad (5)$$

Réarrangeons l'ordre dans lesquels les calculs sont effectués dans l'eq (5), et regroupons les termes pairs et impairs des coefficients :

$$\begin{bmatrix} y_n \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} x_{n-1} & x_{n-3} & \dots & x_{n-L+1} & x_n & x_{n-2} & \dots & x_{n-L+2} \\ x_{n-2} & x_{n-4} & \dots & x_{n-L} & x_{n-1} & x_{n-3} & \dots & x_{n-L+1} \end{bmatrix} \begin{bmatrix} h_1 \\ h_3 \\ \vdots \\ \vdots \\ h_{L-1} \\ h_0 \\ h_2 \\ \vdots \\ \vdots \\ h_{L-2} \end{bmatrix} \quad (6)$$

Ce regroupement, en ce qui concerne le signal d'entrée est équivalent à un sous-échantillonnage à cadence 1/2, les deux signaux sous-échantillonnés étant traités simultanément.

Définissons:

$$\begin{aligned}
 A &= (x_{n-1} \ x_{n-3} \ \dots \ x_{n-L+1}) \\
 B &= (x_{n-2} \ x_{n-4} \ \dots \ x_{n-L}) \\
 C &= (x_n \ x_{n-2} \ \dots \ x_{n-L+2})
 \end{aligned}
 \quad
 \begin{aligned}
 H_0 &= \begin{bmatrix} h_0 \\ h_2 \\ \vdots \\ h_{L-2} \end{bmatrix} &
 H_1 &= \begin{bmatrix} h_1 \\ h_3 \\ \vdots \\ h_{L-1} \end{bmatrix}
 \end{aligned}
 \tag{7}$$

Donc, l'équation (6) devient:

$$\begin{aligned}
 \begin{bmatrix} y_n \\ y_{n-1} \end{bmatrix} &= \begin{bmatrix} A & C \\ B & A \end{bmatrix} \begin{bmatrix} H_1 \\ H_0 \end{bmatrix} \\
 &= \begin{bmatrix} A(H_0 + H_1) - (A - C)H_0 \\ A(H_0 + H_1) + (B - A)H_1 \end{bmatrix}
 \end{aligned}
 \tag{8}$$

Et l'eq. (8) montre que deux produits scalaires de longueur L peuvent être calculés à l'aide de trois produits scalaires de longueur L/2. Comme les coefficients du filtre sont fixes et connus, nous pouvons calculer  $(H_0 + H_1)$  avant le filtrage.  $(A - C)$  et  $(B - A)$  sont des additions (On considère une soustraction comme une addition.) entre 2 échantillons successifs. La plupart d'entre elles sont réutilisables pour calculer les sorties suivantes à l'exception de  $(x_{n-L+1} - x_{n-L+2})$  et  $(x_{n-L} - x_{n-L+1})$ . En d'autres termes, seules deux nouvelles additions  $(x_n - x_{n+1})$  et  $(x_{n+1} - x_{n+2})$  sur les échantillons entrées sont nécessaires pour calculer les deux sorties suivantes  $(y_{n+1}, y_{n+2})$ . Deux additions supplémentaires, nécessaires pour combiner les trois produits scalaires suivant l'équation (8), fournissent les sorties désirées.

La complexité initiale de calcul d'un filtre RIF de l'ordre L est:

1 produit scalaire de longueur L par sortie

soit  $L$  multiplication-accumulations (MACs) par sortie.

Par l'algorithme décrit dans l'équation (8), la complexité de calcul devient:

3 produits scalaires de longueur  $L/2$  + 4 additions pour 2 sorties

soit  $3L/4$  MACs + 2 additions par sortie.

Donc, cet algorithme a réduit le nombre de MACs de 25% par rapport au calcul direct.

Le graphe correspondant à l'équation (8) est donné en la figure 2: Trois filtres de longueur  $L/2$  à vitesse réduite d'un facteur 2 remplacent un filtre de longueur  $L$ . Ces trois sous-filtres attendent 2 échantillons d'entrée avant de calculer 2 sorties du filtre initial. Il s'agit donc un traitement par blocs de taille 2, come peut le montrer également l'équation (5).

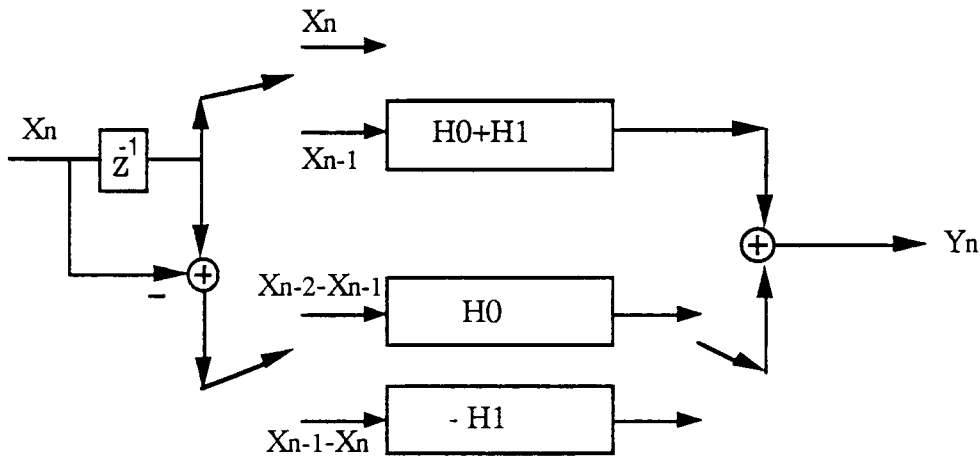


Fig.2 Un filtre RIF de longueur  $L$  par trois filtres de longueur  $L/2$

Les caracteristiques essentielles de cet exemple sont les suivantes:

- 1) Un filtre quelconque peut être calculé par des filtres de longueur plus petite et de cadence réduite; ou bien encore, il est possible d'utiliser des filtres de longueur et de vitesse identiques pour construire un filtre de longueur plus longue et de vitesse plus grande.
- 2) Une réduction de la charge de calcul a été possible sans "casser" complètement la structure de calcul habituelle des filtres RIF, structure qui est très facile à implanter.

Le chapitre suivant présente une approche permettant de dériver les algorithmes de filtrage RIF de manière systématique, et on montrera que les algorithmes rapides classiques peuvent s'obtenir également dans le même cadre.

### 3. Approche unifiée pour les algorithmes de filtrage RIF rapide

Dans cette section, on montre que les algorithmes rapides de calcul des filtres RIF peuvent se décomposer en trois étapes:

- 1) Formulation du filtrage comme un produit polynomial;
- 2) Calcul rapide de ce produit polynomial;
- 3) Recouvrement.

Nous allons présenter chacun de ces points l'un après l'autre.

#### 3.1 Formulation en produit polynomial

Reprenons l'équation (4) qui est déjà un produit polynomial. Mais l'un des deux polynômes à multiplier est d'ordre infini, ce qui rend difficile le calcul rapide du produit. En sous-échantillonnant le signal d'un facteur  $N$ , et en décimant la fonction de transfert du même facteur, nous obtenons un produit polynômial de deux polynômes en  $z^{-N}$  à coefficients eux même polynomiaux .

$$\begin{aligned} & Y_0 + Y_1 z^{-1} + \dots + Y_{N-1} z^{-N+1} \\ &= (H_0 + H_1 z^{-1} + \dots + H_{N-1} z^{-N+1})(X_0 + X_1 z^{-1} + \dots + X_{N-1} z^{-N+1}) \end{aligned} \quad (9)$$

où

$$Y_i = Y_i(z^{-N}) \quad H_i = H_i(z^{-N}) \quad X_i = X_i(z^{-N})$$

$$Y_i(z^{-N}) = \sum_{k=0}^{\infty} y_{kN+i} z^{-kN} \quad X_i(z^{-N}) = \sum_{k=0}^{\infty} x_{kN+i} z^{-kN} \quad H_i(z^{-N}) = \sum_{k=0}^{L/N-1} h_{kN+i} z^{-kN}$$

L'avantage de cette présentation est que les deux polynômes facteurs ont maintenant le même degré  $(N-1)$ . En fait, la décimation d'un facteur  $N$  permettra de travailler sur un bloc de signal de taille  $N$  indépendant de la taille du filtre et de réduire de manière correspondante la charge de calcul. En effet, ayant obtenu une formulation en produit polynomial de degré fini, nous pouvons appliquer des algorithmes rapides pour le calculer. La dérivation de

ceux-ci est en général fondé sur le théorème des restes chinois (TRC) que nous allons maintenant présenter.

### 3.2 Théorème des restes chinois

Ce théorème se trouve dans une oeuvre chinoise de mathématiques qui date de l'année 100. A titre anecdotique, la figure 3 montre le théorème en chinois.

今有物不知其數三三數之賸二五五數之賸三  
七七數之賸二問物幾何  
答曰二十三  
術曰三三數之賸二置一百四十五數  
之賸三置六十三七七數之賸二置三十  
并之得二百三十三以二百一十減之即  
得凡三三數之賸一則置七十五五數之  
賸一則置二十一七七數之賸一則置十  
五一六以上以一百五減之即得

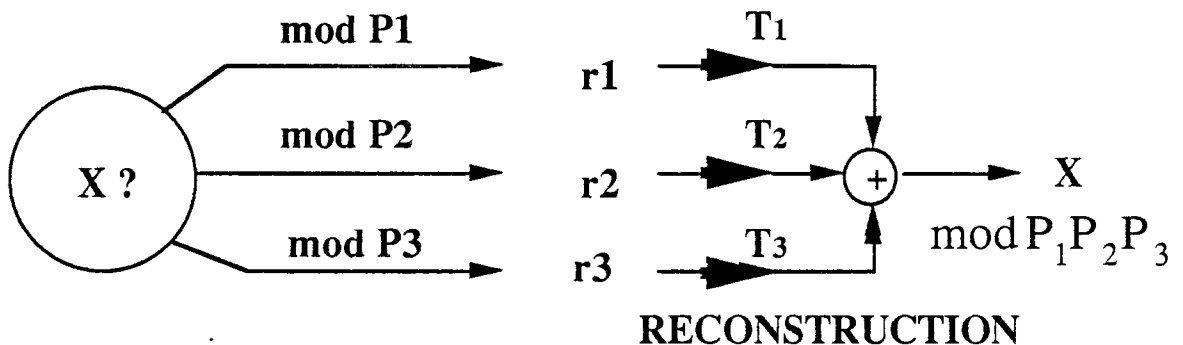
Fig.3 Le théorème des restes chinois en chinois

Une anectode liée à ce théorème raconte que le général Han Xin, l'un des fondateurs de la Dynastie Han (206 av. J.C.-220 ap. J.C.), comptait ses soldats

d'une manière particulière. Il demandait à ses soldats de se regrouper par paquets d'abord de 3, puis de 5, et enfin de 7. Au lieu de compter le nombre de paquets, il ne retenait que les restes. A partir de ceux-ci, il savait calculer le nombre total de soldats. Si l'anecdote s'avère exacte, la date de naissance du TRC est plus ancienne que l'oeuvre mathématique citée ci-dessus .

Ce théorème fait partie a priori de la théorie des nombres. Son énoncé est le suivant: "Etant donné les restes d'un nombre inconnu  $x$  modulo des nombres premiers entre eux mutuellement, nous pouvons reconstruire  $x$  modulo le produit des nombres premiers en cause". La figure 4 interprète le TRC.

### THEOREME DES RESTES CHINOIS



$$X \equiv \sum r_i T_i \mod P_1 P_2 P_3$$

$$\text{où} \quad \begin{aligned} T_i &\equiv 0 \mod P_j & i \neq j \\ &\equiv 1 \mod P_i \end{aligned}$$

Fig.4 Le théorème des restes chinois.

Le TRC s'étend aussi à l'anneau des polynômes. Son énoncé est alors strictement parallèle à celui ci-dessus, mais nous donnons ici plus de détails car c'est cette version qui nous sera la plus utile.

**Théorème des Restes Chinois:** Etant donné un anneau de polynômes modulo  $P(z)$  et

$$P(z) = \prod_{i=1}^s p_i(z) \tag{10}$$



$\{p_i(z)\}$  sont mutuellement premiers entre eux. Tout polynôme  $Q(z)$  est connu modulo  $P(z)$  en fonction de ses restes mod  $p_i(z)$  par l'équation suivante :

$$Q(z) \equiv \sum_{i=1}^s T_i(z) q_i(z) \mod P(z) \quad (11)$$

où

$$q_i(z) \equiv Q(z) \mod p_i(z) \quad ; \text{ ce sont les 'restes' .}$$

$$\begin{aligned} T_i(z) &\equiv 1 \mod p_i(z) && ; \text{ ce sont les polynômes de reconstruction.} \\ &\equiv 0 \mod p_j(z) \text{ for } j \neq i. \end{aligned}$$

Remarque: si l'ordre de  $P(z) >$  celui de  $Q(z)$ , le résultat de la reconstruction est unique et égal à  $P(z)$ .

### 3.3 Calculer un produit polynomial par le TRC

A partir de l'équation (9), soit

$$\begin{aligned} Q(z) &= R_1(z)R_2(z) \\ &= (H_0 + H_1z + \dots + H_{N-1}z^{N-1})(X_0 + X_1z + \dots + X_{N-1}z^{N-1}) \end{aligned} \quad (12)$$

En général, on choisit un polynôme  $P(z)$  complètement factorisable en termes du premier degré:

$$P_i(z) = z - a_i$$

Les  $\{P_i(z), i=0, \dots, K\}$  sont premiers entre eux si les constantes  $\{a_i\}$  sont distinctes. Une évaluation de polynome modulo  $(z - a_i)$  est équivalent à une interpolation. Et donc,

$$\begin{aligned} &Q(z) \mod (z - a_i) \\ &= Q(a_i) \\ &= R_1(a_i) R_2(a_i) \end{aligned} \quad (13)$$

Pour que  $Q(z)$  soit unique,  $2N-1$  points sont nécessaires. On choisit les  $\{a_i\}$  tels que le calcul soit simple. Les points les plus simples sont  $\{0, 1, -1, \infty\}$ .

Les  $\{R_1(a_i)\}$  sont calculés avant le filtrage. Ils sont des polynômes en  $z^{-N}$  de degré  $L/N-1$ . Les  $\{R_2(a_i)\}$  sont les combinaisons des signaux sous-échantillonnés. Ils sont à calculer. Chaque "reste"  $Q(a_i)$  est obtenu par multiplication de  $R_1(a_i)$  et  $R_2(a_i)$ . Tenant compte du fait que  $R_1(a_i)$  et  $R_2(a_i)$

sont eux même polynômes en  $z^{-N}$ , cette multiplication est équivalent à un filtrage à cadence  $1/N$ .

### 3.4 Reconstruction

Pour la reconstruction, calculons d'abord

$$T_i = \frac{P(z)}{P'(a_i)(z - a_i)} \quad \text{où} \quad P(z) = \prod P_i(z) = \prod (z - a_i) \quad (14)$$

Ensuite, appliquons le TRC:

$$\begin{aligned} Q(z) &= R_1(z) R_2(z) \\ &= (H_0 + H_1 z + \dots + H_{N-1} z^{N-1})(X_0 + X_1 z + \dots + X_{N-1} z^{N-1}) \\ &= \sum Q(a_i) T_i \\ &= C_0 + C_1 z + \dots + C_{N-1} z^{N-1} + \dots + C_{2N-2} z^{2N-2} \end{aligned} \quad (15)$$

Nous obtenons ainsi le polynôme  $Q(z)$  qui est caractérisé par ses  $2N-1$  coefficients  $\{C_i\}$ . Mais ceux-ci ne sont pas les sorties du filtre. Les sorties sont calculé à partir de  $\{C_i\}$  par une opération baptisée "recouvrement" (overlap en anglais).

### 3.5 Recouvrement

Par les équations (9) et (15), il est évident que:

$$\begin{aligned} Y_0 + Y_1 z^{-1} + \dots + Y_{N-1} z^{-N+1} \\ = C_0 + C_1 z^{-1} + \dots + C_{N-1} z^{-N+1} + \dots + C_{2N-2} z^{-2N+2} \end{aligned} \quad (16)$$

Et les sorties  $\{Y_i\}$  s'obtiennent facilement par identification:

$$\begin{aligned} Y_{N-1} &= C_{N-1} \\ Y_i &= C_i + z^{-N} C_{N+i} \quad i = 0, 1, \dots, N-2 \end{aligned} \quad (17)$$

L'équation (17) est caractéristique d'un schéma appelé habituellement en anglais "overlap-add" dans le cas de calculs par FFT. Nous donnons ci-dessous un schéma général pour les algorithmes rapides de filtrage RIF faisant apparaitre toutes les étapes nécessaires à leur construction:

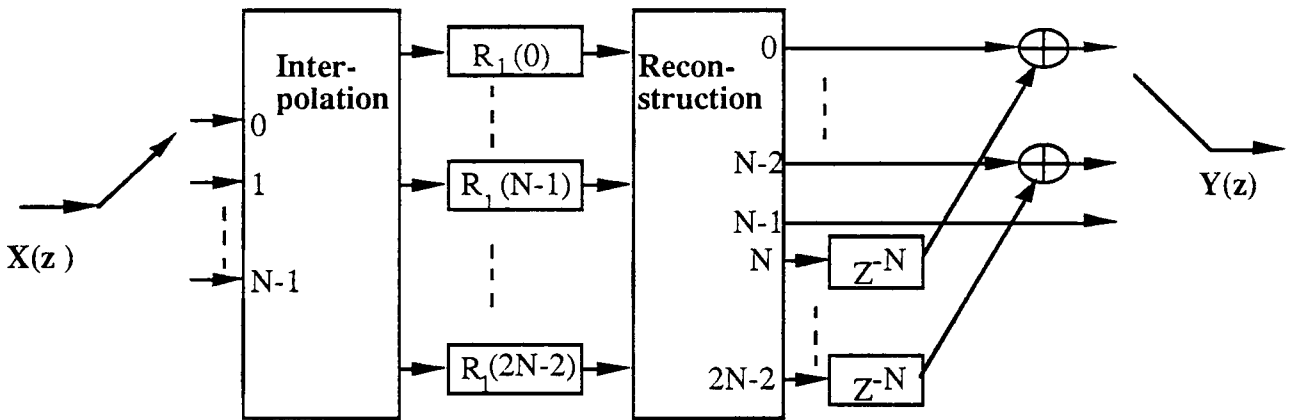


Fig.5 Le schéma général des algorithmes rapides RIF

### 3.6 Convolution pseudocyclique

Si l'on écrit de manière matricielle l'ensemble des relations entre entrée et sortie du filtre sur la longueur du bloc considéré, on obtient:

$$\begin{bmatrix} Y_{N-1} \\ Y_{N-2} \\ \vdots \\ Y_0 \end{bmatrix} = \begin{bmatrix} H_0 & H_1 & \dots & & H_{N-1} \\ z^{-N}H_{N-1} & H_0 & & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & & H_1 & \\ z^{-N}H_1 & & & z^{-N}H_{N-1} & H_0 \end{bmatrix} \begin{bmatrix} X_{N-1} \\ X_{N-2} \\ \vdots \\ X_0 \end{bmatrix} \quad (18)$$

Qui fait intervenir une matrice dite "pseudocyclique". Une autre manière d'expliquer tous les algorithmes rapides connus est de dire qu'ils 'diagonalisent' la matrice pseudocirculante, avec une éventuelle augmentation de la dimension de la matrice diagonale.

A l'aide de cette formulation, on peut montrer qu'il est possible après permutation des entrées et sorties d'obtenir des algorithmes transposés réalisant une fonction de transfert identique et nécessitant la même complexité arithmétique.

Les algorithmes transposés appartiennent alors à une autre classe connue sous le nom d' "overlap-save" dans le cadre des algorithmes classiques.

## 4. Les algorithmes classiques dans le cadre de l'approche

Nous étudions, dans le cadre de l'approche proposée ci-dessus, la dérivation des algorithmes classiques, en montrant leurs avantages et inconvénients.

### 4.1 Algorithmes par FFT

On choisit:  $N \geq L$

$$(h_0 + h_1 z^{-1} + \dots + h_{L-1} z^{-L+1})(x_0 + x_1 z^{-1} + \dots + x_{N-1} z^{-N+1}) \quad (19)$$

et les points d'interpolation sur le cercle unité

$$a_i = \exp(-j \frac{2\pi i}{K}); \quad K = N + L - 1$$

L'interpolation aux points ainsi choisis est alors équivalente à une DFT de longueur K. Souvent on choisit:  $K = N + L - 1 = 2^M$ , parce que la  $2^M$ - DFT se calcule efficacement.

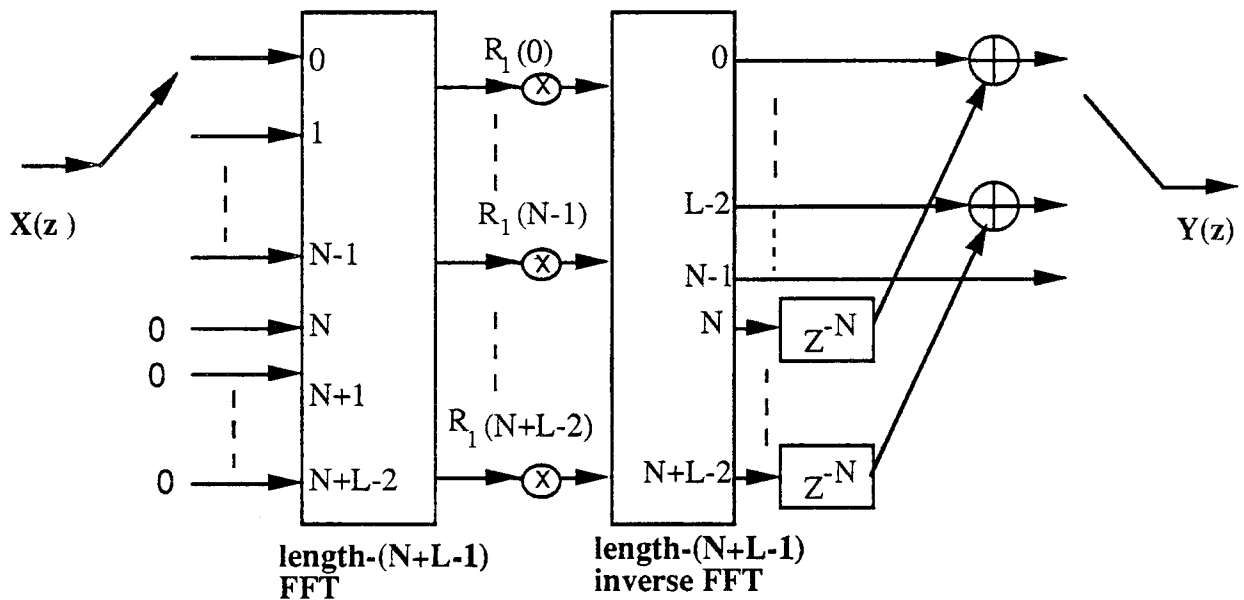


Fig.6 Les algorithmes de filtrage RIF par FFT

### 4.2 Algorithmes de Agarwal-Burrus

On choisit:  $N \geq L$

$$\begin{aligned} & Y_0 + Y_1 z + \dots + Y_{L-1} z^{L-1} \\ & = (h_0 + h_1 z + \dots + h_{L-1} z^{L-1})(X_0 + X_1 z + \dots + X_{L-1} z^{L-1}) \end{aligned} \quad (20)$$

Après regroupement par blocs

$$[(h_0 + h_1 z + \dots + h_{L/2-1} z^{L/2-1}) + z^{L/2} (h_{L/2} + \dots + h_{L-1} z^{L/2-1})] \times \\ [(X_0 + X_1 z + \dots + X_{L/2-1} z^{L/2-1}) + z^{L/2} (X_{L/2} + \dots + X_{L-1} z^{L/2-1})] \quad (21)$$

Soit

$$(G_0 + G_1 z^{L/2}) (F_0 + F_1 z^{L/2})$$

Rappelons l'algorithme pour le produit de 2 polynômes du 1er degré dans Fig.7.

$$(x_0 + x_1 z) (h_0 + h_1 z) \\ = C_0 + C_1 z + C_2 z^2$$

$$\text{points d'interpolations:} \quad a_i = \{0, 1, \infty\}$$

$$C_0: \quad x_0 h_0 \\ C_1: \quad (x_0 + x_1) (h_0 + h_1) - x_0 h_0 - x_1 h_1 \\ C_2: \quad x_1 h_1$$

Fig.7 L'algorithme pour le produit de 2 polynômes du 1er degré.

Appliquons cet algorithme comme suit:

$$(G_0 + G_1 z^{L/2}) (F_0 + F_1 z^{L/2}) \\ = G_0 F_0 + [(G_0 + G_1) (F_0 + F_1) - G_0 F_0 - G_1 F_1] z^{L/2} + G_1 F_1 z^L \quad (22)$$

Réitérons-le sur les termes suivants qui sont encore des produits de polynômes mais de degrés plus faibles:

$$G_0 F_0, (G_0 + G_1) (F_0 + F_1), G_1 F_1$$

C'est donc une interpolation récursive utilisant  $\{0, 1, \infty\}$  comme points d'interpolation.

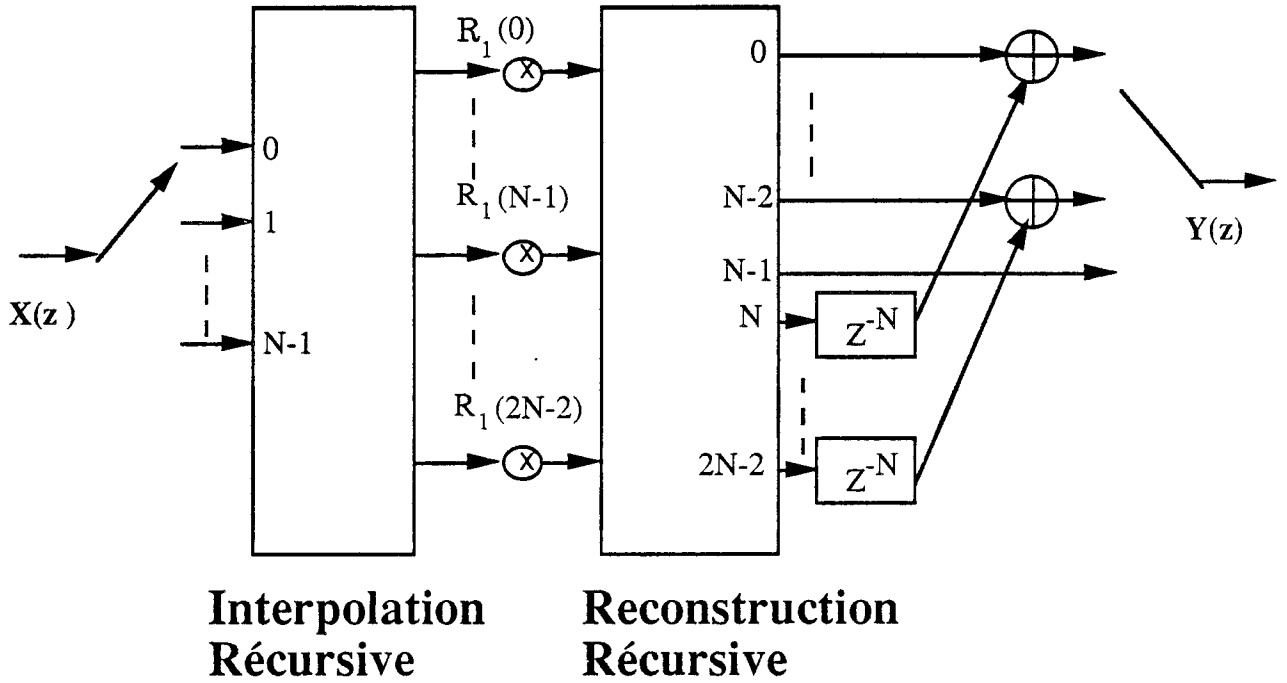


Fig.8 Les algorithmes de Agarwal-Burrus

#### 4.3 Algorithmes de Winograd

L'exemple le plus simple est le suivant où  $L=N=2$ :

$$\begin{bmatrix} y_n \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} x_{n-1} & x_n \\ x_{n-2} & x_{n-1} \end{bmatrix} \begin{bmatrix} h_1 \\ h_0 \end{bmatrix} = \begin{bmatrix} x_{n-1}(h_1 + h_0) - (x_{n-1} - x_n)h_0 \\ x_{n-1}(h_1 + h_0) + (x_{n-2} - x_{n-1})h_1 \end{bmatrix} \quad (23)$$

C'est le transposé de l'algorithme qui calcule le produit de deux polynômes du premier degré.

Quand  $N \geq L$ , nous devons calculer le produit d'une matrice Toeplitz et d'un vecteur:

$$\begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-L+1} \end{bmatrix} = \begin{bmatrix} x_{n-L+1} & \cdot & \cdot & \cdot & x_{n-1} & x_n \\ x_{n-L} & x_{n-L+1} & \cdot & \cdot & x_{n-1} & \cdot \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n-2L+2} & \cdot & \cdot & \cdot & x_{n-L} & x_{n-L+1} \end{bmatrix} \begin{bmatrix} h_{L-1} \\ h_{L-2} \\ \vdots \\ h_0 \end{bmatrix} \quad (24)$$

En partitionnant la matrice :

$$\begin{bmatrix} E1 \\ E0 \end{bmatrix} = \begin{bmatrix} F0 & F1 \\ F2 & F0 \end{bmatrix} \begin{bmatrix} G1 \\ G0 \end{bmatrix}$$

Nous obtenons:

$$\begin{bmatrix} E1 \\ E0 \end{bmatrix} = \begin{bmatrix} F0(G1+G0) - (F0-F1)G0 \\ F0(G1+G0) + (F2-F0)G1 \end{bmatrix} \quad (25)$$

Pour réduire davantage la complexité arithmétique, on peut appliquer la même procédure récursivement sur les termes suivants:

$$F0(G1+G0), (F0-F1)G0, (F2-F0)G1$$

Les algorithmes de Winograd sont obtenus en utilisant un algorithme générateur qui est le transposé de celui utilisé dans les algorithmes de Agarwal-Burrus. Nous avons illustré ce fait par un seul exemple, mais cette remarque est générale.

#### 4.4 Remarques

Les algorithmes classiques présentés ci-dessus ont pour but de minimiser essentiellement le nombre de multiplications. Ils sont intéressants pour les processeurs à usage général, dans le cas de filtres très longs. Leur caractéristique commune est d'utiliser de grands blocs de calcul. Leur principal défaut réside dans le fait que leur structure est trop compliquée pour que leur utilisation soit véritablement efficace dans les processeurs de traitement du signal. Ils sont d'autre part difficilement implantables en VLSI, à cause précisément de l'utilisation de grands blocs de calcul. Enfin, ces grands blocs de calcul induisent un retard important de traitement, retard qui peut devenir critique pour certaines applications temps réel.

#### 5. Nouvelles possibilités

L'approche unifiée nous permet non seulement d'établir un cadre général pour tous les algorithmes existants mais aussi de dériver des nouveaux algorithmes. Nous allons présenter quelques nouvelles possibilités dans les paragraphes suivantes.

### 5.1 Algorithmes de petite longueur

On choisit  $N=2$  sans tenir compte de l'ordre du filtre. Nous obtenons:

$$\begin{aligned} Y_0(z^2) + Y_1(z^2)z &= [H_0(z^2) + H_1(z^2)z][X_0(z^2) + X_1(z^2)z] \\ &= R_1(z)R_2(z) = Q(z) \end{aligned} \quad (26)$$

Et on interpole  $Q(z)$  aux points suivants:  $\{a_i\} = \{0, 1, \infty\}$ . Donc:

$$\begin{aligned} R_1(0) &= H_0(z^2) & R_2(0) &= X_0(z^2) \\ R_1(1) &= H_0(z^2) + H_1(z^2) & R_2(1) &= X_0(z^2) + X_1(z^2) \\ R_1(\infty) &= H_1(z^2) & R_2(\infty) &= X_1(z^2) \end{aligned}$$

Calculons ensuite les "restes":

$$\begin{aligned} Q(0) &= R_1(0) R_2(0) \\ Q(1) &= R_1(1) R_2(1) \\ Q(\infty) &= R_1(\infty) R_2(\infty) \end{aligned}$$

On obtient les sorties:

$$\begin{aligned} Y_0(z^2) &= Q(0) + z^2 Q(\infty) \\ Y_1(z^2) &= Q(1) - Q(0) - Q(\infty) \end{aligned} \quad (27)$$

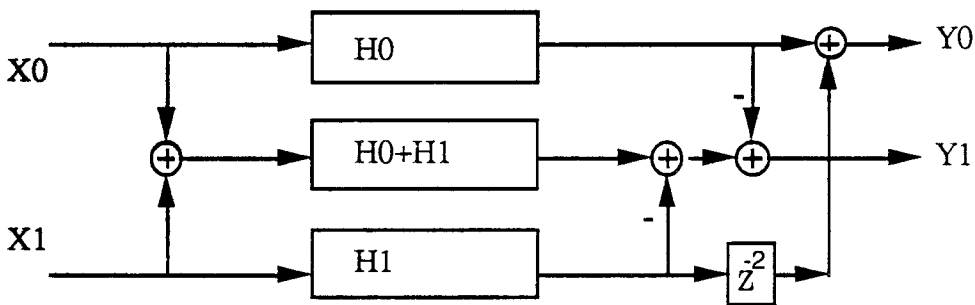


Fig.9 L'algorithme de petite longueur pour  $N=2$ .

Cet algorithme calcule 2 sorties d'un filtre de l'ordre  $L$  à la fois par trois filtres de l'ordre  $L/2$ . Le nombre d'opérations est le suivant:

$3L/4$  multiplication-accumulations (MACs) par sortie

25% de réduction est ainsi obtenu. Cet algorithme présente les avantages suivants:

1) Réduction de la complexité;



## 2) Maintien de la structure "MAC".

Quand  $N=2$ , seulement 3 points d'interpolation sont nécessaires pour dériver l'algorithme. Comme il existe 4 points d'interpolation simples (c'est à dire ne faisant pas intervenir de multiplications) :  $\{0,1,-1,\infty\}$ , on a 4 variantes possibles pour le cas où  $N=2$ .

Dans le cas où  $N=3$ , on aurait besoin de 5 points simples pour dériver un algorithme optimal. Mais les points autres que  $\{0,1,-1,\infty\}$ ,  $\pm 2$  par ex. génèrent beaucoup d'additions et augmentent la sensibilité de l'algorithme. Pour  $N>2$ , il est donc préférable de chercher des solutions sous-optimales vis à vis du nombre de multiplications (ou de sous-filtres dans ce cas précis), mais ne générant pas trop d'additions.

Voici la dérivation d'un algorithme sous-optimal de radix-3:

$$\begin{aligned}
 & Y_0(z^3) + Y_1(z^3)z + Y_2(z^3)z^2 \\
 &= [H_0(z^3) + H_1(z^3)z + H_2(z^3)z^2][X_0(z^3) + X_1(z^3)z + X_2(z^3)z^2] \\
 &= [H_0 + z(H_1 + H_2z)][X_0 + z(X_1 + X_2z)] \\
 &= [H_0 + zF][X_0 + zG]
 \end{aligned}$$

Itérons l'algorithme de longueur 2 sur  $[H_0 + zF][X_0 + zG]$ , puis sur  $[H_0 + F][X_0 + G]$  et  $FG$ . Ceci nous permet d'obtenir un algorithme de radix-3 qui n'a besoin que de 6 multiplications (filtrages de longueur  $L/3$ ), au lieu de 9 pour un algorithme direct. La complexité de calcul est donc:

$2L/3$  MACs par sortie

On obtient alors une réduction de 33% par rapport aux  $L$  MACs par sortie initialement nécessaires. Il est également possible de dériver des algorithmes pour des tailles de blocs plus grandes. Dans ce cas, la réduction du nombre de MACs est plus importante, alors que l'interpolation et la reconstruction deviennent plus complexes, c'est-à-dire, il y a plus d'opérations avant et après les sous-filtrages. L'utilité de ces algorithmes dépendra alors de la taille du filtre, puisque on obtient un gain proportionnel à la taille du filtre au prix d'un nombre d'opérations fixe.

Quand  $N$  est factorisable, soit  $N=N_1N_2$ , nous pouvons construire un algorithme de radix  $N$  à partir des algorithmes de radix  $N_1$  et  $N_2$ . Par

exemple, quand  $N = 6 = 2 \times 3$ , nous pouvons appliquer d'abord la décomposition radix-2 et ensuite radix-3 ou d'abord radix-3 et ensuite radix-2 pour obtenir un algorithme de radix-6. Le nombre de multiplications (filtrages) ne varie pas dans les algorithmes ainsi construits.

$$M = M(2) M(3) = 18$$

Cependant, le nombre d'additions dépend de l'ordre dans lequel l'itération s'effectue. Pour cet exemple,

2x3: 42 Additions

3x2: 44 Additions

Et dans le cas où  $N$  admet plusieurs facteurs:

$$N = N_1 N_2 N_3 \dots N_K$$

Nous avons défini un coefficient de qualité pour déterminer l'ordre d'itération qui rend le nombre d'additions minimum:

$$Q = (M_i - N_i) / A_i$$

où  $M_i$  le nombre de multiplications,  $N_i$  le radix et  $A_i$  le nombre d'additions.

Voici le coefficient  $Q$  pour certains  $N$ .

N	M	A	Q
1	L	L-1	1
2	3	4	0,25
3	6	10	0,3
4	9	20	0,25
5	12	40	0,175

Le 1er algorithme à appliquer est celui qui a le coefficient de qualité le plus faible.

## 5.2 Algorithmes de petite longueur pour des données complexes

Dans le corps de nombres complexes, les points d'interpolation les plus simples sont  $\{a_j\} = \{0, 1, -1, j, -j, \infty\}$ . On peut établir des algorithmes optimaux pour  $N = \{2, 3\}$ . La méthodologie permettant de dériver les algorithmes est semblable à celle décrite pour les données réelles.

Il faut cependant tenir compte de la particularité suivante du filtrage complexe: Pour tout filtrage complexe  $Y(z) = H(z)X(z)$ :

$$Y_r(z) + jY_i(z) = [H_r(z) + jH_i(z)][X_r(z) + jX_i(z)]$$

nous pouvons réduire la complexité de calcul par l'algorithme de multiplication complexe à 3 multiplications réelles :

$$\begin{aligned} Y_r(z) &= H_r(z) [X_r(z) + X_i(z)] - [H_r(z) + H_i(z)] X_i(z) \\ Y_i(z) &= H_r(z) [X_r(z) + X_i(z)] - [H_r(z) - H_i(z)] X_r(z) \end{aligned} \quad (28)$$

De telle façon, un filtrage complexe est calculé par trois filtres réels (Fig.10) au lieu de quatre.

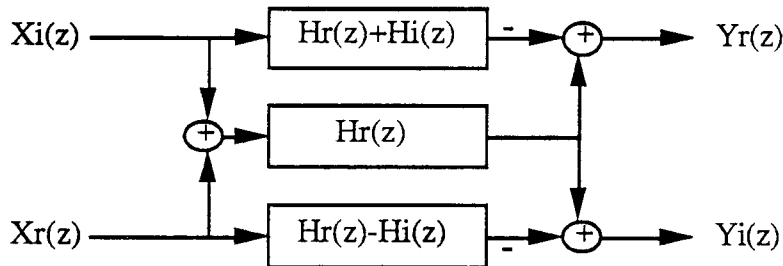


Fig.10 Un filtre complexe par 3 filtres réels

### 5.3 Algorithmes par FFT courte

Dans les algorithmes classiques, la taille de bloc  $N$  est supérieure ou égale à l'ordre du filtre  $L$ . Mais ici nous proposons des algorithmes pour  $N < L$ .

Les points d'interpolation sont :

$$\{a_l\} = \left\{ \exp\left(-j \frac{2\pi l}{2N}\right) \right\}$$

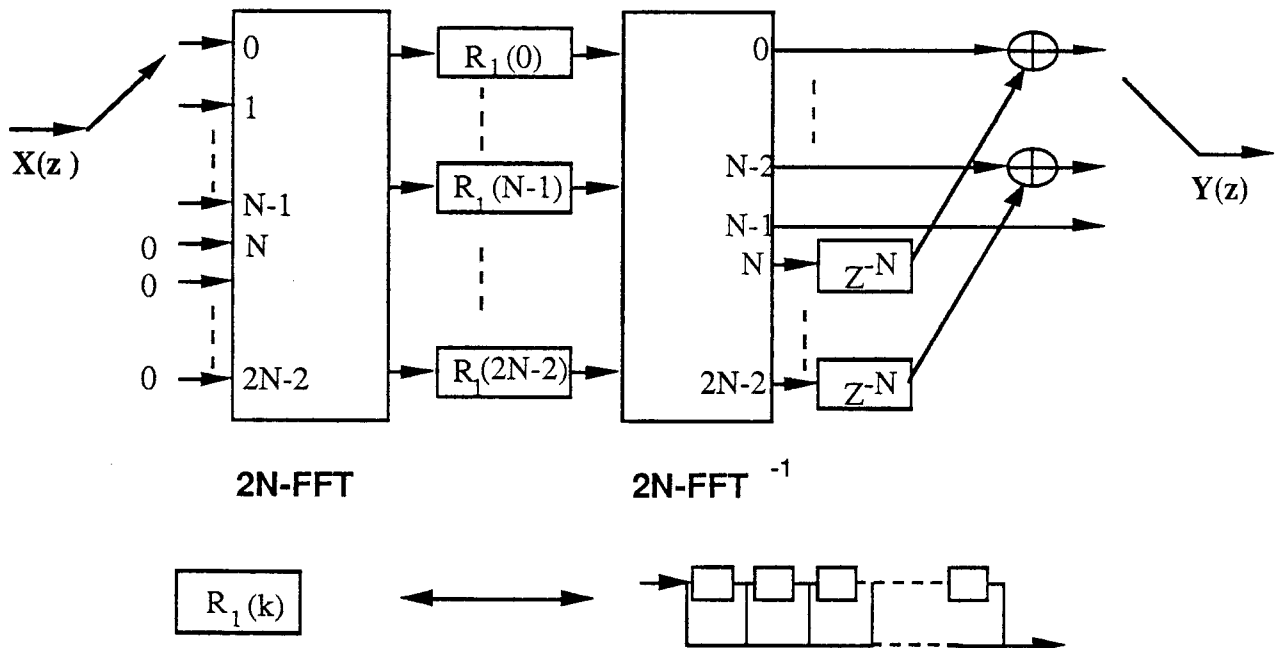


Fig.11 Le schéma des algorithmes par FFT courte

Dans le cas où  $N < L$ , les multiplications dans le domaine fréquentiel sont devenues des filtrages complexes de longueur  $L/N$ . Par exemple, pour  $N=L/2$ ,  $R_1(k)$  sont des filtres de longueur 2.

En variant le rapport entre  $N$  et  $L$ , nous pouvons dériver tous les intermédiaires entre traitements temporel et fréquentiel, qui représentent différents compromis entre complexité de calcul et structure régulière (la partie "MAC"). Notons enfin que la charge de calcul de ces algorithmes peut encore être réduite par l'utilisation d'algorithmes de petite longueur pour le filtrage RIF complexe.

On peut montrer, et il s'agit là d'un résultat étonnant, que dans certains cas, e.g.  $N = L/2$ , les algorithmes par FFT courte nécessitent moins d'opérations que ceux par FFT "longue" qui sont utilisés habituellement.

## Résumé de la partie "Algorithmes"

A. Une approche unifiée => tous les intermédiaires entre traitements temporel et fréquentiel.

B. Revue des algorithmes classiques => Leurs dérivations, leurs avantages et défauts.

C. Nouvelles classes d'algorithmes permettant :

- de maintenir l'architecture "MAC",
- de réduire le nombre d'opérations,
- d'éviter les grands blocs.

## 6. Architectures pour le filtre RIF

Dans ce chapitre, nous présentons des architectures pour implanter le filtre RIF en circuits intégrés. Le principe de base des architectures est l'arithmétique distribuée. Nous apportons des contributions à cette méthode classique en éliminant la ROM et en proposant de nouvelles structures d'accumulation rapide.

### 6.1 Principe de l'arithmétique distribuée

L'arithmétique est une technique de calcul du produit scalaire de deux vecteurs dont l'un est constant. Le filtre RIF tombe bien dans le champ d'application de cette technique parce qu'il s'écrit comme suit:

$$y_n = \sum_{i=0}^{L-1} h_i x_{n-i} \quad n=0, 1, \dots, \infty \quad (29)$$

Supposons que  $y$ ,  $h$  et  $x$  sont codés en complément en 2. Le développement de  $x$  au niveau du bit fait apparaître une présentation qui est la somme de plusieurs termes similaires pondérés par des puissances de 2 (représentant le décalage dans le calcul binaire classique):

$$\begin{aligned} y_n &= \sum_{i=0}^{L-1} h_i (-x_{n-i,0} + x_{n-i,1} 2^{-1} + \dots + x_{n-i,B-1} 2^{-B+1}) \\ &= -\sum_{i=0}^{L-1} h_i x_{n-i,0} + 2^{-1} \sum_{i=0}^{L-1} h_i x_{n-i,1} + \dots + 2^{-B+1} \sum_{i=0}^{L-1} h_i x_{n-i,B-1} \end{aligned} \quad (30)$$

Définissons une fonction comportant  $L$  variables binaires  $\{a_i\}$ :

$$P(a_0, a_1, \dots, a_{L-1}) = \sum_{i=0}^{L-1} h_i a_{L-i-1} \quad (31)$$

Dans le cas où les  $\{h_i\}$  sont constants, cette fonction peut prendre au plus  $2^L$  valeurs suivant les différentes combinaisons possibles des  $a_i$ . Si ces  $2^L$  valeurs sont stockées dans une ROM à l'adresse correspondant aux combinaisons appropriées, nous pouvons obtenir le résultat de la fonction correspondant à une combinaison d'entrée tout simplement en lisant le contenu de la ROM à cette adresse.

En utilisant cette fonction plusieurs fois, nous arrivons à calculer tous les termes de la somme dans l'eq.(30). Un accumulateur en sortie de la ROM additionne ces termes et fournit le résultat final du produit scalaire.

Le principe est illustré en Fig.12. Les registres bit-série sont enchainés pour former une ligne à retard, classique dans tous les filtres RIF.

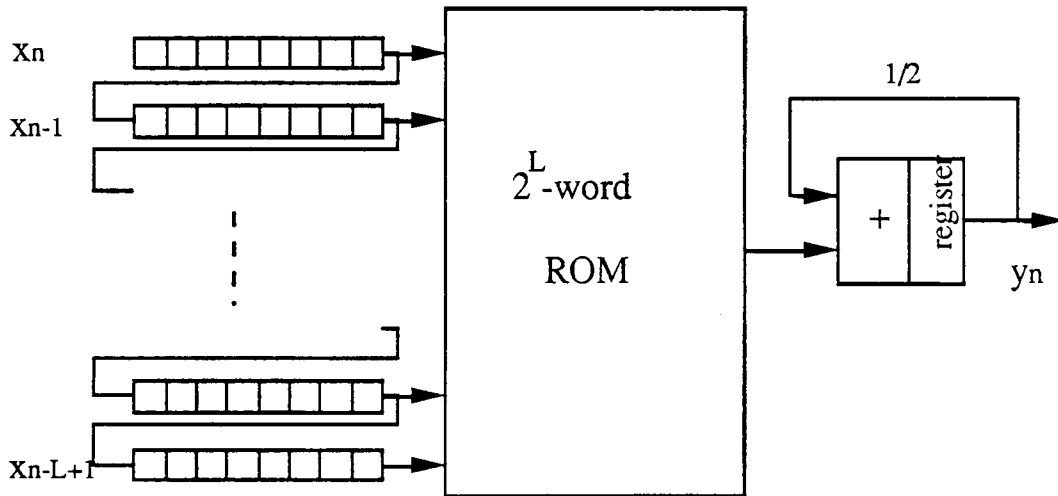


Fig.12 L'architecture du filtre RIF par l'arithmétique distribuée.

Pour une implantation en circuits intégrés, la ROM devient vite encombrante, même quand  $L$  est moyen, et donc coûte cher en termes de silicium. De plus la ROM limite la vitesse de calcul, car elle ne permet pas l'utilisation de techniques de pipelining.

Si nous calculons deux produits scalaires de longueur  $L/2$  et les additionnons ensuite, on trouve ainsi une autre façon de calculer un produit scalaire de longueur  $L$  qui s'avère plus simple et nécessite seulement deux ROMs de taille  $2^{L/2}$ . Cette solution classique (Fig.13) est nettement supérieure à la précédente à partir d'un certain degré du filtre.

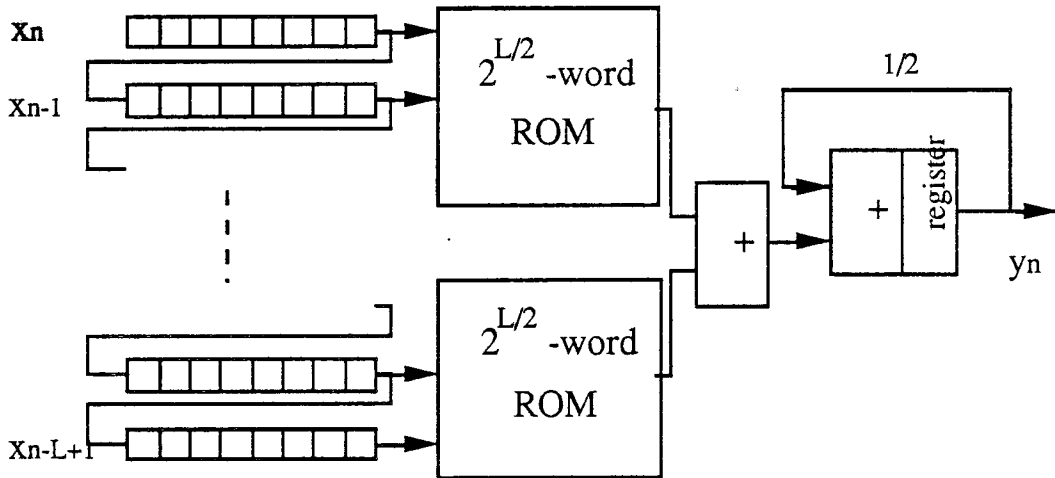


Fig.13 L'architecture utilisant des ROM de taille plus petite.

La décomposition des ROMs peut se faire récursivement jusqu'au moment où les mémoires ne contiennent plus qu'un seul coefficient. Nous obtenons ainsi une architecture par additionneurs en arbre (Fig.14) où la mémorisation est réduite à un minimum. Le découpage de la ROM en éléments plus fins permet non seulement de réduire la surface d'implantation, mais aussi d'insérer des registres de pipeline si nécessaire.

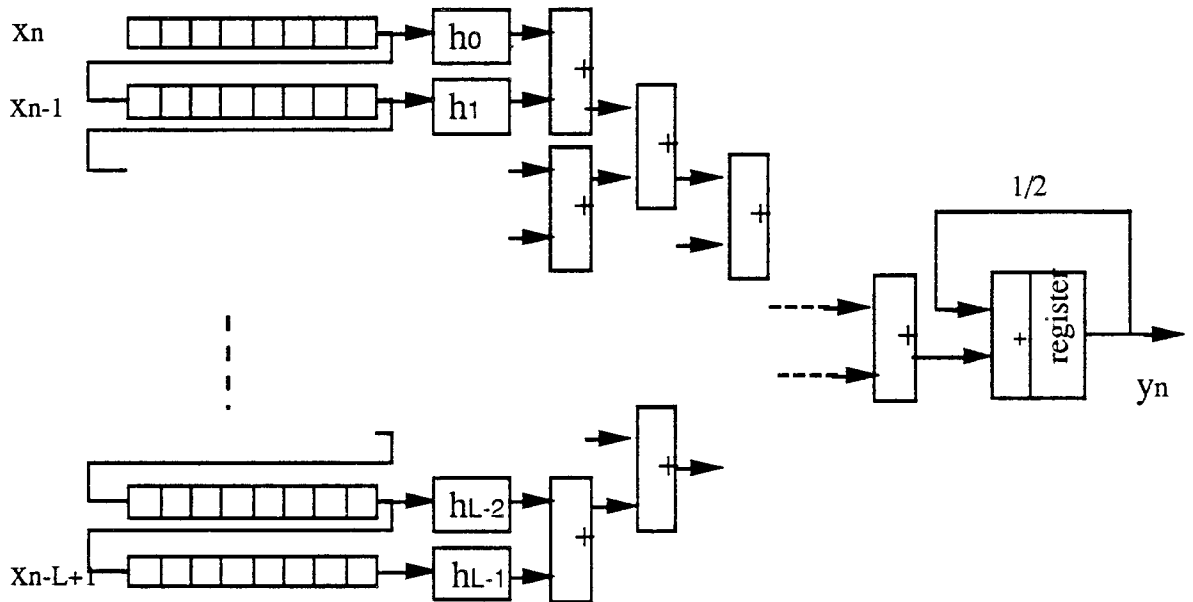


Fig.14 L'architecture par additionneurs en arbre

Une implantation directe de cette architecture n'est cependant pas si efficace que ce à quoi on pourrait s'attendre, à cause du mouvement des retenues orthogonal à celui des sommes.



Pour faciliter l'implantation, nous pouvons ramener les additionneurs à une forme linéaire qui est illustré en Fig.15. Le mouvement des retenues (carry) est horizontal alors que celui des sommes est vertical, conduisant à un mouvement du flot de données approximativement diagonal. C'est une architecture régulière. Un circuit a été réalisé à base de cette structure dont le schéma est montré en Fig.16. On peut remarquer les registres de pipelining antidiagonaux.

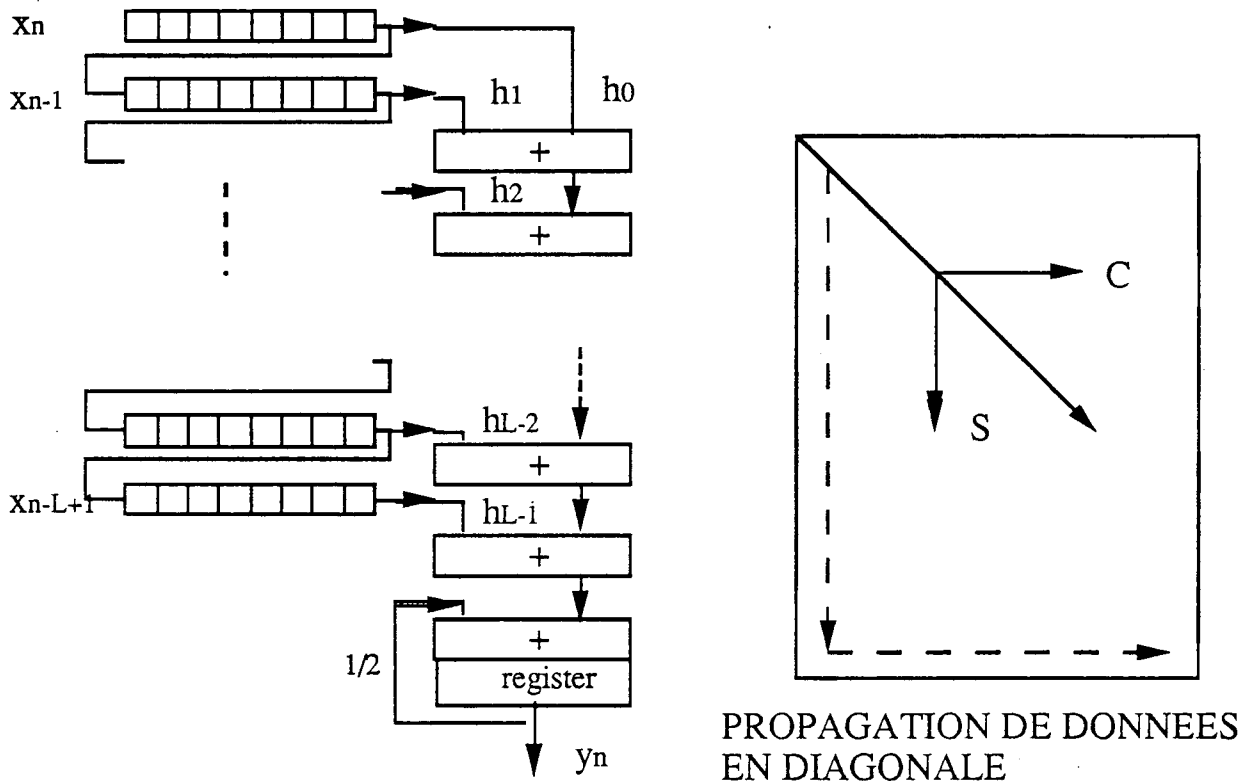


Fig.15 L'architecture par additionneurs en forme linéaire.

On peut constater que la partie 'calcul' de l'architecture accumule des opérandes multiples à chaque temps de cycle. La vitesse de calcul dépend alors fortement de l'accumulation multi-opérande choisie. Pour cette raison, nous présentons des techniques d'accumulation rapide.

## 6.2 Accumulateurs multi-opérandes rapides

D'abord, nous rappelons le concept de l'additionneur 'carry-save' (CSA). Le CSA additionne trois nombres en deux alors que l'additionneur habituel additionne deux nombres en un seul (Fig.17). Ce dernier nécessite une propagation de retenue dont le temps est proportionnel à la longueur de mots, alors que le CSA n'a pas besoin de la propagation et son temps de calcul est

indépendent de la longueur de mots. Ceci est l'origine du gain en temps de calcul apporté par le CSA.

Les accumulateurs rapides proposés sont tous à base de CSA. Nous donnons quelques exemples en Fig.18.

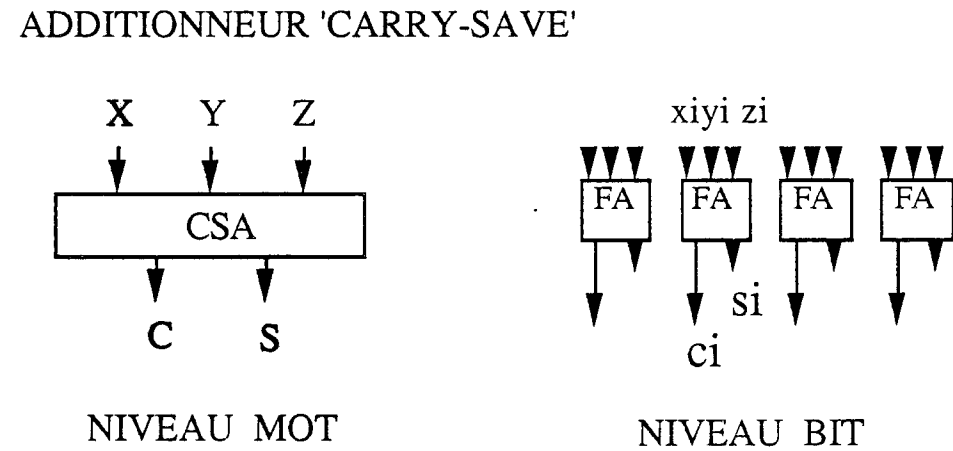
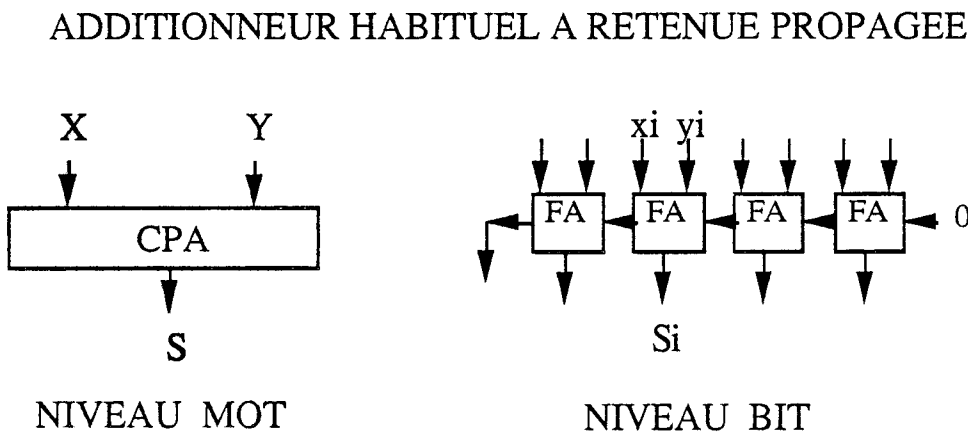
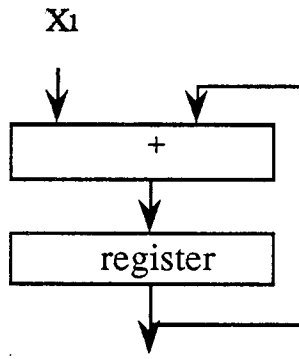
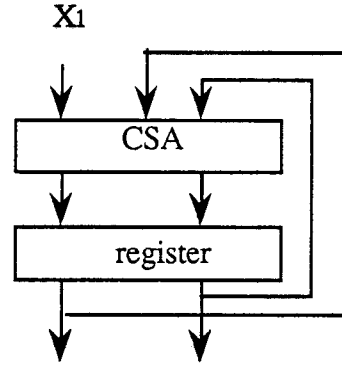


Fig.17 L'additionneur habituel et l'additionneur 'carry-save'

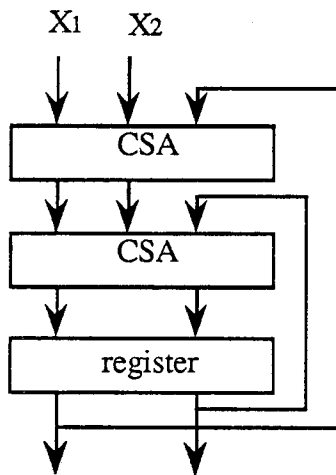
## ACCUMULATEUR HABITUEL



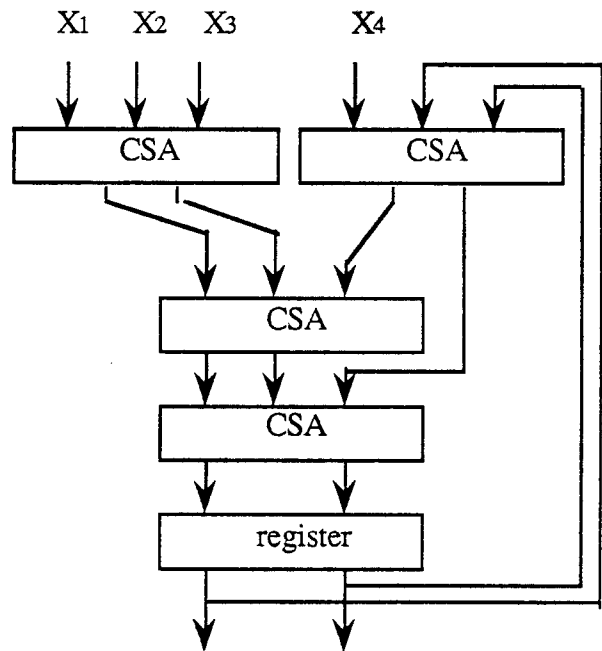
## ACCUMULATEUR 'CARRY-SAVE'



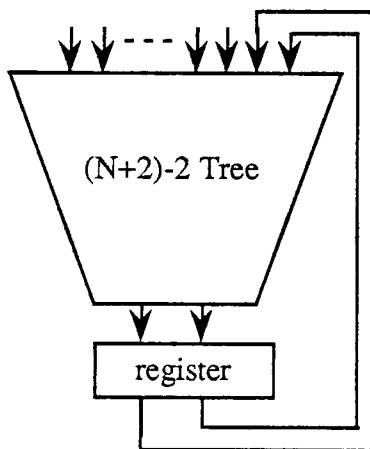
## ACCUMULATEUR 2-OPERANDE



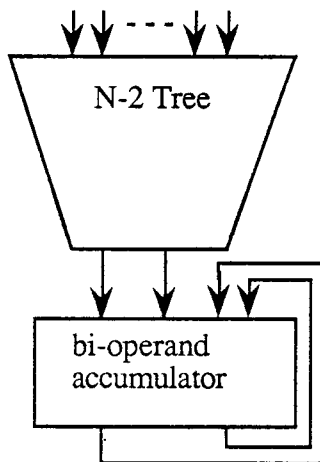
## ACCUMULATEUR 4-OPERANDE



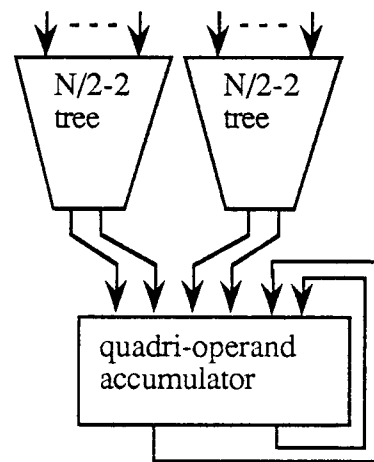
## 3 ALTERNATIVES DE L'ACCUMULATEUR MULTIOPERANDE



(a)



(b)



(c)

Fig.18 Un jeu d'accumulateurs.

Dans les accumulateurs multi-opérandes, nous devons additionner d'abord les opérandes avant de stocker et de reboucler. L'application des structures arborescentes, comme l'arbre de Wallace (Fig.19), permet d'effectuer une partie des calculs en parallèle, et donc d'additionner très rapidement les opérandes.

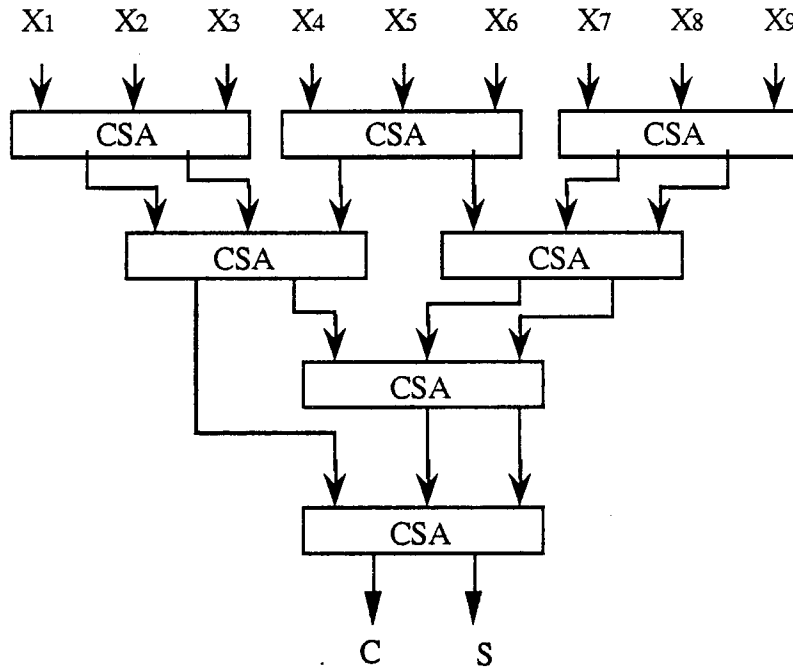


Fig.19 L'arbre de Wallace 9-2 (l'addition de 9 opérandes à 2 sorties).

Pendant longtemps, on a considéré que les structures arborescentes n'étaient pas implantables de manière très efficace en surface de silicium, malgré leur vitesse de calcul extrêmement court. Récemment, certaines méthodes ont été présentées, permettant des conceptions régulières et compactes.

### 5.3 Nouvelles architectures

Les nouvelles architectures sont caractérisée par:

- additionneur "carry-save" comme brique de base
- somme rapide par l'arbre de Wallace
- possibilité de pipeliner à un niveau arbitraire
- accumulation rapide

La fig.20 présente une architecture à base de CSA en forme linéaire. Celle-ci corrige le mouvement diagonal des données de la Fig.15 en les ramenant tous vers le bas où se trouve un accumulateur bi-opérande.

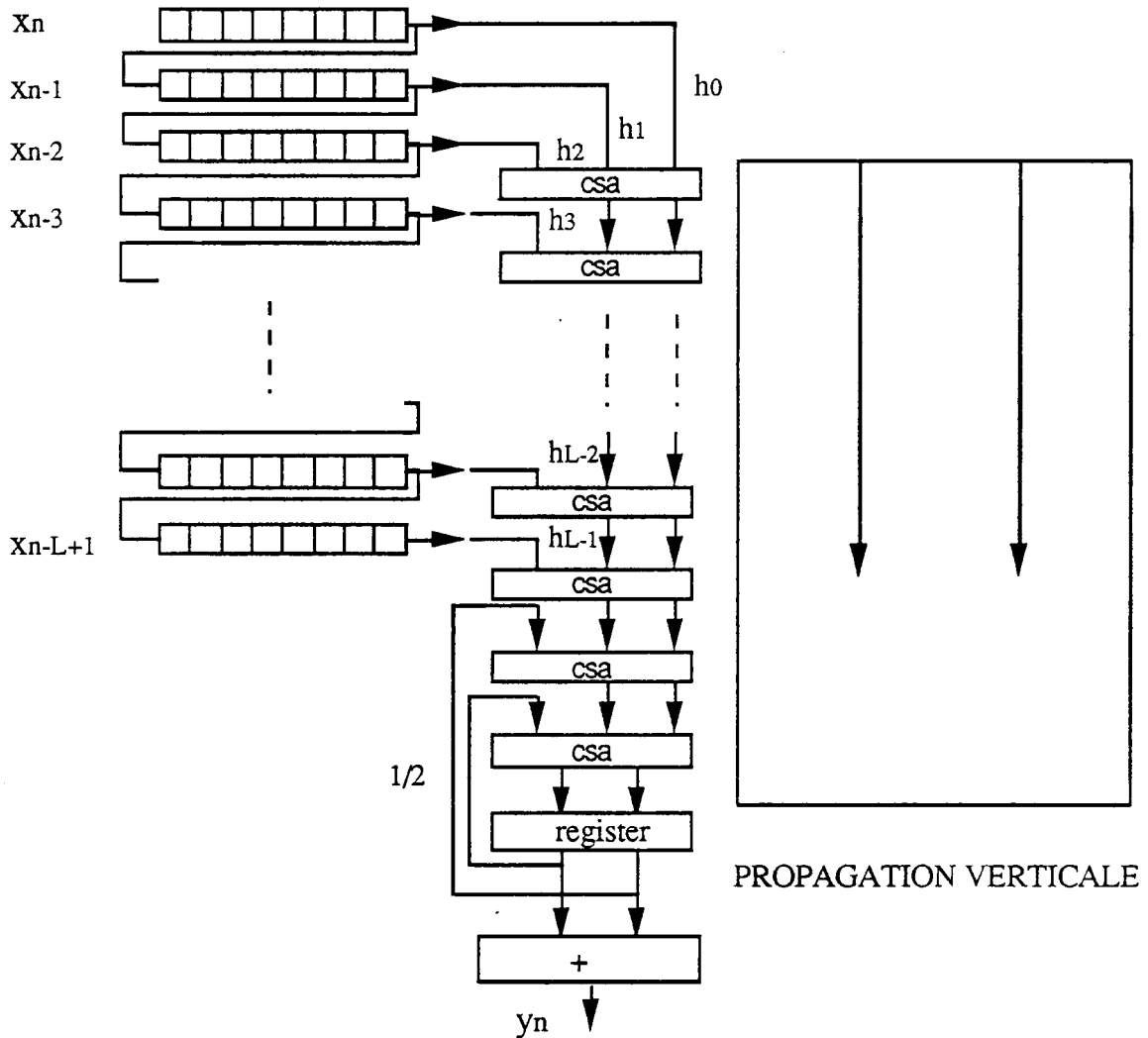


Fig.20 L'architecture par additionneurs 'carry-save' en forme linéaire

Cette architecture est très facile à pipeliner. L'architecture de la Fig.21 montre un exemple avec des étages de pipelinage et aussi des arbres de Wallace.

Il est aussi possible de pipeliner à un niveau arbitraire, éventuellement à chaque étage de CSA pour atteindre une vitesse de calcul maximale. Une autre variante de cette architecture est obtenue en traitant deux bits à la fois et en appliquant le codage de Booth modifié, ce qui permet un fonctionnement presque deux fois plus rapide, et ce avec peu de matériel supplémentaire.

Certaines autres variantes ont également été présentées, dont une architecture spécialisée pour les filtre RIF symétriques. Dans ce cas, un module commun calculant  $(X+Y)H$  est nécessaire. Un codage a été proposé pour calculer cette opération d'une façon efficace, éliminant l'addition  $(X+Y)$ . Ce codage semble cependant plus avantageux pour une conception en bit-parallèle qu'en bit-série.

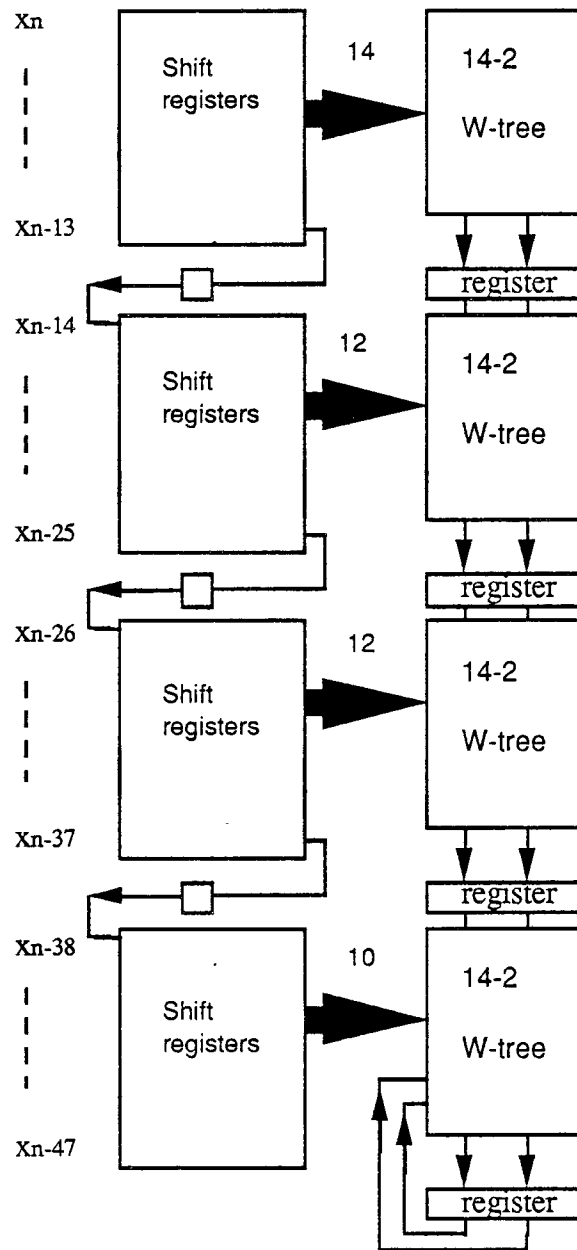


Fig.21 L'architecture pipelinée avec des arbres de Wallace.

## RESUME DE LA PARTIE "ARCHITECTURE"

A. Etude de l'arithmétique de base dans le cadre du multiplieur-accumulateur.

B. Nouvelles architectures de filtre par arithmétique distribuée ayant les caractéristiques suivantes: sans ROM; additionneur 'carry-save' comme brique de base; accumulation rapide.

C. Codage pour calculer  $(X+Y)H$  dans le filtre RIF symétrique.