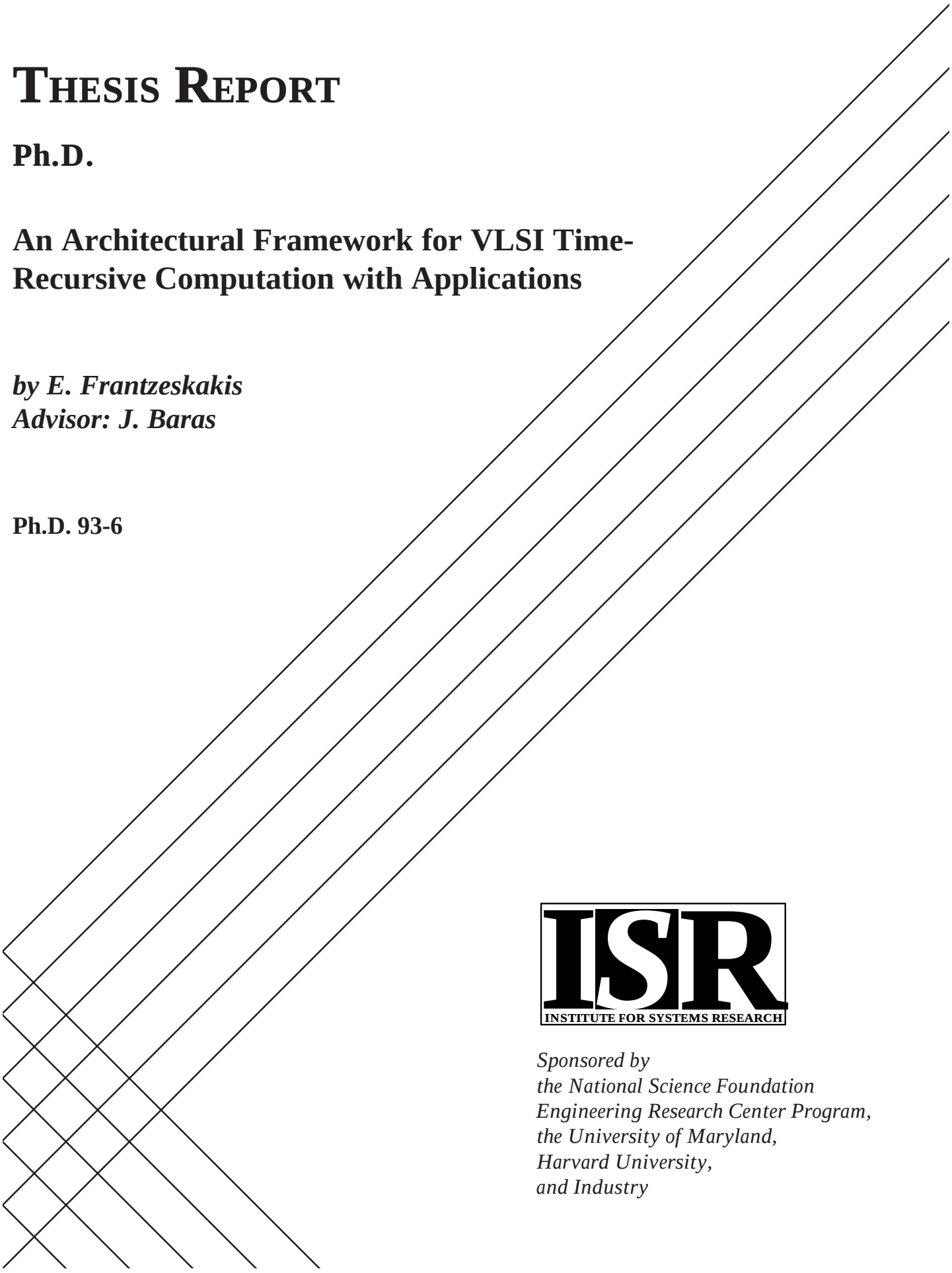
THESIS REPORT

Ph.D.

An Architectural Framework for VLSI Time-Recursive Computation with Applications

by E. Frantzeskakis
Advisor: J. Baras

Ph.D. 93-6



Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

ABSTRACT

Title of Dissertation: **An Architectural Framework for VLSI Time-Recursive Computation with Applications**

Emmanuel Frantzeskakis, Doctor of Philosophy, 1993

Dissertation directed by: Professor John Baras
Department of Electrical Engineering

The time-recursive computation model has been proven as a particularly useful tool in audio, video, radar and sonar real-time data processing architectures. Unlike the FFT based architectures, the time-recursive ones require only local communication, they imply linear implementation cost and they operate in a single-input multiple-output (SIMO) manner. This is appropriate for the above applications since the data are supplied serially. Also, the time-recursive architectures are modular and regular and they allow high degree of parallelism; thus they are very appropriate for VLSI implementation.

In this dissertation, we establish an architectural framework for parallel time-recursive computation. We consider a class of linear operators (or signal transformers) that are characterized by discrete time, time invariant, compactly supported, but otherwise arbitrary kernel functions. We specify the properties of linear operators that can be implemented efficiently in a time-recursive way. Based on these properties, we develop a systematic routine that produces a time-recursive architectural implementation for a given operator. We demonstrate the use and effectiveness of this routine by means of specific examples, namely the Discrete Cosine Transform (DCT), the Discrete Fourier Transform (DFT) and the Discrete Wavelet Transform (DWT).

By using this architectural framework we obtain novel architectures for the uniform-DFT QMF bank, the cosine modulated QMF bank, the 1-D and 2-D Modulated Lapped Transform (MLT), as well as an Extended Lapped Transform (ELT). Furthermore, the architectural implementation of the Cepstral Transform and a Short Time Fourier Transform are considered based on the time-recursive architecture of the DFT. All of the above designs are modular, regular, with local communication and linear cost in operator counts. In particular, the 1-D MLT requires $2N + 3$ multipliers, $3N + 3$ adders and $N - 1$ rotation circuits, where N denotes the data block size. The 2-D MLT requires 3 1-D MLT circuits and no matrix transposition. The ELT has basis length equal to $4N$ and it requires $3N + 4$ multipliers, $4N + 4$ adders and $N + 2$ rotation circuits. These results are expected to have a significant impact on real-time audio and video data compression, in frequency domain adaptive filtering and in spectrum analysis.

An Architectural Framework for VLSI Time-Recursive Computation with Applications

by

Emmanuel Frantzeskakis

Dissertation submitted to the Faculty of The Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1993

Advisory Committee:

Professor John Baras, Chairman/ Advisor
Assistant Professor K.J. Ray Liu
Professor Joseph Jájá
Associate Professor Steven Tretter
Associate Professor Nariman Farvardin
Associate Professor Ahmet Oruc

DEDICATION

To my wife Ioanna, my son Nicholas
and my parents Nicholas and Vassiliki.

ACKNOWLEDGMENTS

I would like to express my gratitude to my thesis advisor Dr. John Baras for providing an abundance of opportunity to explore diverse areas of research and for his encouragement, guidance and inspiration throughout the course of my graduate studies. Especially, I appreciate the academic freedom I was given. I am indebted to Dr. Ray Liu who gave me the opportunity and inspiration to broaden my interests and research in many ways and served as a constant source of encouragement. I thank the other members of my advisory committee Dr. J.Jájá, Dr. N.Farvardin, Dr. S.Tretter and Dr. A.Oruc for the unforgetable experience I have had collaborating with them.

A number of fellow students also merit special acknowledgment for serving as sounding boards for ideas, offering suggestions, comments and support over the years. I thank Bernie Frankpitt, Yan Zuang, Zeev Berman, and certainly my old colleague Babis Karathanasis.

I would like to thank the Institute for Systems Research and the Electrical Engineering Department for the excellent resources and the stimulating research environment.

I wish to thank my loving wive Ioanna for her devotion, love and continuous assistance especially at the most difficult moments, that allowed me to pursue this work with pleasure. A special acknowledgment goes to my joyful son Nicholas, who is as old as this thesis. Also, I would like to thank my parents Nicholas and Vassiliki and my sister Pepi for their unlimited confidence and support since my early school years. I would like to take this opportunity to express my gratitude for two of the most inspiring teachers I have had the chance to

meet Ms Irini Iconomacou and Mr Dimitri Karandreas. Before I close, I certainly need to acknowledge the fellow greek students at UMCP and the Digenis association for making the past few years more enjoyable.

This research was supported in part by the National Science Foundation's Engineering Research Centers Program: NSFD CDR 8803012.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER
1

Introduction

1.1 Historical Background and Motivation

Since the revolutionary publication by Cooley and Tukey [11] the fast Fourier transform (FFT) has been playing a key role in digital signal processing (DSP) in a wide range of applications, including transform coding, data filtering and spectral estimation [44, 51, 39]. Nevertheless, the increasing demand of processing huge volumes of data in real time, especially in audio, radar, sonar and video applications, suggests that the FFT-like fast algorithms (see for example [44, 59, 30, 36, 39, 48, 57]) may not be considered as the main building block in a number of modern DSP applications. The same is true for the VLSI architectures based on these algorithms. Three different reasons corroborate to this conjecture:

- Global Communication: The flow graphs of the FFT and the FFT-like fast algorithms exhibit a common structure composed of a series of $\log_2 N$ alternating butterfly interconnection and multiplier stages. The butterfly communication scheme requires global communication links if a parallel implementation is to be considered.

- Block Processing: The fast algorithms require buffering of blocks of data, and then block processing. This is not the natural way of processing for a number of applications such as audio, radar, sonar and video, where the input data are supplied in a sequential manner.

- Sliding Transform Computational Complexity: The FFT has been widely used for the transformation of windowed data, where the frequency content of the data is extracted for each displacement of a sliding window. This phenomenon appears in the transform domain adaptive filtering problem [12, 41, 43, 51]. The Discrete Fourier Transform used in such applications is referred to as the sliding DFT [51]. The FFT implementation of the sliding DFT requires the repetitive processing of neighboring data samples when the window slides.

We can observe that the problem of computing the transform coefficients has appeared in two forms: in the first, one needs to compute the transform coefficients of the input data vector $[x(t-N+1),\ x(t-N+2),\ \cdots,\ x(t)]$ for each time instant $t = 0, 1, \cdots$, where N is the size of a sliding window. In the second, one computes the above coefficients only at the time instants $t = 0, N, 2N, \cdots$. For the sake of clarity throughout this thesis, we use the name **sliding transform** to describe the former situation and we reserve the term **block transform** for the latter one.

A number of authors have proposed a time recursive approach (either as a DSP programmable algorithm or VLSI architecture) for implementing sliding transforms in the context of the transform domain adaptive filtering, [7, 10, 42, 43, 61]. Within the time-recursive approach, the 8×8 Discrete Cosine Transform is viewed as eight linear operators that function independently from each other: given a common input sequence they produce eight coefficient sequences. These frequency coefficients are evaluated at time instant $t+1$ based on their values at time instant t and the input sample with time indices $t-N+1$ and $t+1$ (see Fig. 1.1). The motivation behind the time-recursive implementation of a sliding transform is the fact that the above update computation involves $O(N)$ operations instead of $O(N \log_2 N)$ implied in the FFT based implementations. This is reflected in the asymptotic cost expressions on Table 1.1 [1].

[1]The expressions under "throughput" denote the number of operations on the critical paths of the corresponding flow graphs that need to be carried out before the next block of data can be processed; in this case, before the window slides over the next position. A thorough

Time-recursive computation of sliding transforms has also found application in spectral analysis [32, 1] and frequency domain data filtering [47].

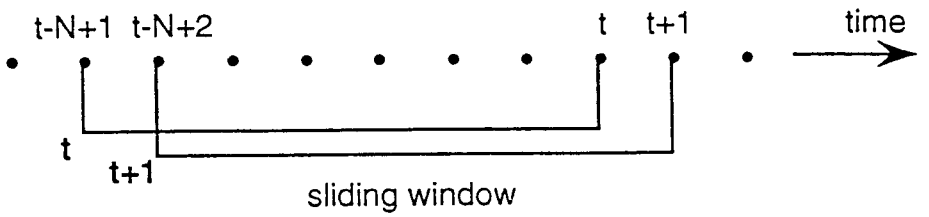


Figure 1.1: Pertain to the time-recursive computation.

	parallel architecture criteria			uniprocessor criteria	
	operators	throughput	communic.	operations per output point	latency per block
direct	$O(N^2)$	$O(N)$	local	$O(N)$	$O(N^2)$
pipelined direct	$O(N^2)$	$O(1)$	local		
FFT based	$O(N \log N)$	$O(\log N)$	global	$O(\log N)$	$O(N \log N)$
pipelined FFT	$O(N \log N)$	$O(1)$	global		
time-recursive	$O(N)$	$O(1)$	local	$O(1)$	$O(N)$

Table 1.1: Implementation cost for sliding DFT.

More recently, time-recursive designs have been successfully used in architectures for real-time computation of block transforms applied in data compression [9, 33, 34]. On Table 1.2, we display the asymptotic expressions of cost metrics used for the comparison of the time-recursive implementations with the FFT-like ones. Obviously, the algorithmic implementation of the FFT on a uniprocessor outperforms the corresponding one of the time-recursive implementation. At the architectural level though, the constant factors hidden by the asymptotic expressions are affected by the implementation details, thus leaving space for interesting cost trade-offs. The time-recursive implementation involves $O(N)$ operators and local communication as opposed to $O(N \log N)$ operators and global communication for a fully parallel and pipelined FFT architecture. In other words, less area and shorter internal clock cycle can by employed by

discussion on throughput requirements is pursued in Subsection 3.2.3.

the time-recursive architecture. A relevant asymptotic optimality result is formally proved in [34]. Note also that the locality property allows a bit-parallel implementation of the operators unlike the case of the architectures based on fast algorithms, where the global communication suggests a bit-serial implementation. Furthermore, the time-recursive architectures are very efficient for separable multi-dimensional data transforms. In particular, the implementation cost is linear in terms of the operators and the communication requirement remains local. The induction procedure for designing multi-dimensional architectures based on the one-dimensional ones is described in [34], while a detailed example is described in [9].

	parallel architecture criteria			uniprocessor criteria	
	operators	throughput	communic.	operations per output point	latency per block
direct	$O(N^2)$	$O(N)$	local	$O(N)$	$O(N^2)$
pipelined direct	$O(N^2)$	$O(1)$	local		
FFT based	$O(N \log N)$	$O(\log N)$	global	$O(\log N)$	$O(N \log N)$
pipelined FFT	$O(N \log N)$	$O(1)$	global		
time-recursive	$O(N)$	$O(N)$	local	$O(N)$	$O(N^2)$

Table 1.2: Implementation cost for DFT.

The subtle point in the time-recursive architectural implementation of the block transforms hinges on the fact that (as we will see in Chapter 2) the operators need to evaluate one result per time unit, while an operator in the fully parallel and pipelined FFT needs to produce one result every N time units (or equivalently, perform $1/N$ operation per time unit). The time unit is the time that lapses between two adjacent input data. Apparently, this is the reason that has discouraged the research and use of time-recursive computation for block transform implementation until recently [9, 33]. This situation has been changed due to the advances in VLSI technology that penalize more the global communication than the requirement for short internal clock cycle. As a side effect, the speed of a (VLSI implemented) operator can match the input data rate, by adjusting the length of the clock cycle. As long as this constraint is satisfied for a real-time application the area minimization becomes the only

concern in the design. Under this light, the success of the time-recursive VLSI circuits in computing block transforms and the promise they show are mainly justified, apart from the modularity, regularity and scalability of the resulted designs, by virtue of the area optimality property [34] and the communication locality property.

1.2 Problem Identification and Contribution

We have seen in Section 1.1 that a number of time-recursive expressions have been used for implementing diverse computations in different application areas. Also, we have briefly explained why time-recursive computation can be expected to play a more dominant role in high throughput real-time applications in the near future. It is therefore very desirable to know what kind of computations can be carried out in an efficient manner by a time-recursive formulation, so that we can explore the full potential of the time-recursive computation model. Furthermore, one will question: given a computation in this class, does it translate to a unique architecture ? If not, what are the alternatives and which is the best one ? Can we derive this best architecture in an easy step-by-step, routine, or even automated way ? Providing answers to the above questions, exploiting the implications in a number of real-time computation applications and revealing the common infrastructure of such computations in diverse areas constitutes the subject of the present thesis.

More specifically, we establish an architectural framework for parallel time-recursive computation. We consider a class of linear operators that is characterized by discrete time, time invariant, compactly supported, but otherwise arbitrary kernel functions. We introduce a **shift property** and show that this property underlies the time-recursive realization of the above operators. We show that the shift property dictates the parameter values of a parametric architectural structure, the **lattice structure**, that is a generalization of the structure that has appeared in the literature with the same name (see [9, 33, 42]). We show that the information carried by the shift property is equivalent to the

information carried by the parameters of a linear **difference equation**. The latter is a generalization of the order-2 difference equation introduced in [34] and it dictates an IIR structure for the time-recursive architecture. We introduce a **periodicity property** and show how it affects the choice among the lattice and the IIR architecture, as well as the complexity of the architecture itself. We show that an arbitrary mapping operator can be implemented in a time-recursive way and we provide an optimal implementation (in the class of time-recursive implementations). We conclude that given a mapping operator the efficiency of the time-recursive architecture depends on the eigenfunctions of an appropriate Linear Time Invariant (LTI) system.

Based on the above analysis, we obtain the flow graph of a design procedure that routinely produces the time-recursive architecture for a specified mapping operator. This design procedure can also be viewed as the back bone of a CAD tool that can take a high level description of a computation (for example an algebraic formula) and produce VLSI layout for the appropriate time-recursive architecture. The efficiency and potential of this design procedure is demonstrated by means of specific design examples: the Discrete Cosine Transform (DCT) [34], the Discrete Fourier Transform (DFT) [32] and the Discrete Wavelet Transform (DWT) [2]. Furthermore, the architecture design of the Short Time Fourier Transform with Hanning windowing is demonstrated. Among the above, the inverse DCT we propose, the DWT and the inverse DWT are novel designs.

A brief description of four more advanced designs follows:

1. **Cepstral Transform:** The size-N complex Cepstral transform $y(t), t = 0, 1, \cdots$ of a real valued sequence $x(t), t = 0, 1, \cdots$ is defined as [46]

$$y(t) = IDFT\{\log(DFT\{x(t)\})\}, \quad t = 0, 1, \cdots,$$

where IDFT denotes the inverse DFT. The architectural implementation involves two fast Fourier transforms and a bank of logarithm circuits. We propose two architectures for the Cepstral transform, in which we implement DFT and IDFT in a time recursive way. We are not concerned about

the implementation details of the logarithm circuits, nevertheless we are concerned about the speed of this circuit. In the first architecture, we assume that logarithm circuits which can perform one operation per time unit are available. The cost of the resulted time-recursive architecture is $3N-1$ adders, $N-2$ rotation circuits, and N "fast" logarithm circuits. In the second one, we assume that the logarithm circuits can perform one operation every N time units. In this case, the resulted cost is $6N-1$ adders, $3N-6$ rotation circuits and N "slow" logarithm circuits. Both designs may employ very efficient rotation circuits [52].

2. **N-Band Modulated Quadrature Mirror Filter (QMF) bank:** There are two common versions of the modulated QMF bank [54]: the **uniform-DFT QMF** bank and the **cosine modulated QMF bank**. They both find applications in audio data processing and they enjoy substantial advantages over alternative designs of both Perfect Reconstruction (PR) QMF banks and pseudo QMF banks [54]. In particular, they are easier both to design and to implement. The cosine modulated QMF bank has the additional advantage that given a real input sequence the produced output is also real. These filter banks are implemented by using the polyphase components of a prototype filter [54, 39] followed by a transform matrix. The latter is the $N \times N$ IDFT matrix for the case of the uniform-DFT QMF and a $N \times 2N$ matrix of cosines for the case of the cosine modulated QMF. These matrices are typically implemented by fast algorithm techniques that require global communication. Here, we propose the substitution of the fast algorithm implementation by using a time-recursive approach. The resulting circuits involve only local communication, while the cost of the modulation matrices becomes linear.

3. **One-Dimensional Lapped Transforms:** The Modulated Lapped Transform (MLT) operates on data segments of length $2N$, $x(t+n-2N+1), n=0,1,\cdots,2N-1$ and it produces N output coefficients $X_k(t), k=$

$0, 1, \cdots, N-1$ as follows [36]:

$$X_k(t) = c_k\sqrt{\frac{2}{N}}\sum_{n=0}^{2N-1}\sin\frac{\pi}{2N}\left(n+\frac{1}{2}\right)\cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right)\right] x(t+n-2N+1),$$

where $t = 0, 1, \cdots$ and $c_k = (-1)^{(k+2)/2}$ for even k and $c_k = (-1)^{(k-1)/2}$ for odd k. The MLT is a special case of PR cosine modulated QMF bank, where the polyphase components mentioned above are FIR filters of length equal to 2. The MLT has been proved very useful in audio data coding because it alleviates the blocking effect [36]. Here, we propose a time-recursive implementation of the MLT. The implied cost is $2N + 3$ multipliers, $3N + 3$ adders and $N - 1$ rotation circuits.

Furthermore, we design a circuit for an Extended Lapped Transform (ELT) with basis function length equal to $4N$ [38]. This ELT is equivalent to a PR cosine modulated QMF bank with polyphase components of length 4. The implementation cost is $3N + 4$ multipliers, $4N + 4$ adders and $N + 2$ rotation circuits.

We would like to highlight the importance of these results by mentioning that the MLT and ELT have been incorporated by the ISO-MPEG and ASPEC standards for audio coding with the name Modified DCT (MDCT).

4. **Two-dimensional MLT:** The 2D MLT is used for image data coding. Compared to 2D DCT, at the same coding rate it implies considerably less blocking effect and slightly higher Signal to Noise Ratio [39]. Here, we propose the time-recursive implementation of the 2D MLT, for which we need three one-dimensional MLT circuits, with an overhead of shift registers and linear number of adders. The resulting design is modular, regular, scalable and it requires local communication.

All of the above designs are original and are the best known to date in terms of operator counts.

Although throughout the thesis we focus on architectural implementations for either sliding transforms or block transforms, the designs we present are useful for algorithmic implementations of sliding transforms too.

1.3 Thesis Organization

In this thesis, we establish a framework for parallel time-recursive computation, we explain the subtle points by means of specific examples and we propose novel designs for some important real-time computations.

In Chapter 2, we expose the main ideas of the time-recursive computation, we derive the various representations of a time-recursive computation and we generate the basic architectural structures that will serve as the building modules in later chapters.

In Chapter 3, we introduce a generic design procedure for time-recursive architectures and explain its use by designing architectures for the Discrete Fourier Transform (DFT), the Discrete Cosine Transform (DCT) and a dyadic Discrete Wavelet Transform (DWT). Furthermore, based on the DFT architecture, we propose a novel design for the Cepstral Transform.

In Chapter 4, we consider a number of problems related to modulated QMF banks: introducing the time-recursive computation in the uniform-DFT QMF bank and the cosine modulated bank, implementing the Short Time Fourier Transform (STFT) with a non-rectangular window, the one-dimensional Modulated Lapped Transform (MLT) and an Extended Lapped Transform (ELT).

In Chapter 5, we derive the architecture of the two-dimensional MLT.

In Chapter 6, we summarize the results of this dissertation and we suggest some directions for future research.

In Appendix A, we give the proofs of some lemmas that are stated in the course of our presentation.

CHAPTER
2

Architectural Framework

In this Chapter, we exploit the structure underlying time-recursive computation and we use this knowledge to construct the tools necessary for a systematic design of time-recursive architectures.

In Section 2.1, we introduce some terminology that will be used throughout the thesis. Also, for easy reference, we highlight some principles from linear systems theory. In Section 2.2, we study the time-recursive algorithmic structures and their properties. In Section 2.3, we focus on the architectural implementation of time-recursive architectures. In Section 2.4, we briefly discuss the special features pertinent to block data transforms. We conclude with Section 2.5.

2.1 Preliminaries

2.1.1 Definitions

In many signal processing applications the key computation consists of a **mapping operator** $[h_0\ h_1\ \cdots\ h_{N-1}] : x(\cdot) \rightarrow X(\cdot)$, which operates on the semi-infinite sequence of scalar data $x(\cdot)$ and produces the sequence $X(\cdot)$ as follows:

$$X(t) = \sum_{n=0}^{N-1} h_n x(t+n-N+1), \quad t = 0, 1, \cdots. \tag{2.1.1}$$

Note that all FIR filters can be considered as this type of computation. This is also true for a number of data transforms. For example, the k^{th} frequency component of the N-point Discrete Fourier Transform (DFT) is obtained for

$h_n = e^{-\frac{2\pi j}{N}kn}$.

We can specify a mapping operator $[h_0\ h_1\ \cdots\ h_{N-1}]$ with a function $f(\cdot)$, for which the values at the points $0, 1, \cdots, N-1$ are the prescribed coefficients: $h_n = f(n),\ n = 0, 1, \cdots, N-1$. In the sequel, we will use the term **kernel function** or simply **kernel** for this function $f(\cdot)$. For example, the kernel $f(n) = 1$ is associated to the mapping operator $[h_n = 1, n = 0, 1, \cdots, N-1]$ and similarly, the kernel $f(n) = e^{\alpha n}$ is associated to the operator $[e^{\alpha n}, n = 0, 1, \cdots, N-1]$. Furthermore, we will call **kernel group** a vector of kernel functions $f_0(\cdot), f_1(\cdot), \cdots, f_{M-1}$

$$\mathbf{f}(\cdot) = [f_0(\cdot)\ f_1(\cdot)\ \cdots\ f_{M-1}(\cdot)]^T.$$

A time-recursive implementation of a mapping operator $[h_n\ h_1\ \cdots\ h_{N-1}]$ is the one that is based on an update computation of the type

$$X(t+1) = \mathcal{U}(X(t), x(t-N+1), x(t+1)).$$

For example, if we have $[h_n = 1, n = 0, 1, \cdots, N-1]$, $X(t)$ will be the sum of the last N values in the input stream. The time-recursive algorithmic implementation of this operator will simply be the computation

$$X(t+1) = X(t) + x(t+1) - x(t-N+1).$$

2.1.2 Linear Time-Invariant Systems

In this Section, we make a brief review of some basic concepts from the linear system theory. This will be helpful for a better understanding of the material in Chapter 2. For a comprehensive presentation of these concepts the reader may refer to [27].

A single-input single-output discrete-time Linear Time Invariant (LTI) system of order M can be described by the state space equations

$$\begin{aligned}\mathbf{x}(n+1) &= \mathbf{A}\mathbf{x}(n) + \mathbf{b}u(n)\\ y(n) &= \mathbf{c}\mathbf{x}(n),\end{aligned}$$

where $\mathbf{x}(\cdot)$ is the length-M **state vector** and $u(\cdot), y(\cdot)$ are the input and output variables respectively. Also, $\mathbf{A}$ is the $M \times M$ **system matrix**, while $\mathbf{b}$ is a length-M column vector and $\mathbf{c}$ is a length-M row vector. $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ is an **order-M state space description** of the LTI system and it provides information for both the input/output behavior and the internal structure of the system.

The quantities

$$h_n = \mathbf{c}\mathbf{A}^n\mathbf{b}, \tag{2.1.2}$$

$n = 0, 1, \cdots$, are the **Markov parameters** of the linear system and they are sufficient to describe completely the input/output (i/o) behavior of this system. Consequently, providing an i/o description for a system is equivalent to providing the Markov parameters, while the state space description for this system is non-unique.

Two different order-M state space descriptions that yield a common i/o behavior are related with a **similarity transformation** specified as follows

$$\{\mathbf{A}, \mathbf{b}, \mathbf{c}\} \longrightarrow \{\mathbf{T}^{-1}\mathbf{A}\mathbf{T},\ \mathbf{T}^{-1}\mathbf{b}, \mathbf{c}\mathbf{T}\},$$

where $\mathbf{T}$ is an invertible $M \times M$ matrix. One can easily verify that the transformed state space description has the same Markov parameters with the original one.

There is a number of standard state space description forms. Two of the most commonly used ones are:

1. **Controller canonical form**: $\mathbf{A}$ and $\mathbf{b}$ are in the form

$$\mathbf{A} = \begin{bmatrix} -\alpha_1 & -\alpha_2 & \cdots & -\alpha_{M-1} & -\alpha_M \\ 1 & 0 & \cdots & & 0 \\ 0 & 1 & & & \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & & 0 & 1 & 0 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{2.1.3}$$

while $\alpha_1, \alpha_2, \cdots, \alpha_M$ are non-zero parameters. The output vector $\mathbf{c}$ does

not have any specific structure.

2. **Modal canonical form**: Let $\beta_i e^{\pm\alpha_i}$, $i = 0, 1, \cdots, I$ be the eigenvalues of the system matrix $\mathbf{A}$ in polar representation. Note that for real eigenvalues we have $\alpha_i = 0$. A linear system is in modal canonical form if the system matrix $\mathbf{A}$ is block diagonal with block elements in the form

$$\beta_i \begin{bmatrix} \cos\alpha_i & \sin\alpha_i \\ \cos\alpha_i & -\sin\alpha_i \end{bmatrix}.$$

The above reduces to β_i if $\alpha_i = 0$.

Given the Markov parameters $h_0, h_1, \cdots$, of an LTI system a state space description $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ of order M specifies a **minimal order realization** if (2.1.2) holds with $n = 0, 1, \cdots$ and furthermore (2.1.2) does not hold for any state space realization of order $M - 1$.

Given the Markov parameters $h_0, h_1, \cdots, h_{N-1}$, of an LTI system, a state space description $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ of order M specifies a **minimal order partial realization** if (2.1.2) holds with $n = 0, 1, \cdots, N-1$ and furthermore it does not hold for any state space description of order $M - 1$. The problem of finding a minimal order partial realization is addressed in detail in [29]. An illustrative example is discussed in [27, pp.491-492].

2.2 Design of a Time-Recursive Algorithm

2.2.1 The Shift Property

In the course of our study we will see that all mapping operators specified in (2.1.1) can be implemented in a time-recursive way. Nevertheless, the implementation cost not always justifies the time-recursive computation. In the following Lemma, we specify a family of kernels and kernel groups that can be implemented time-recursively in a way that will be determined shortly.

Shift Property: A kernel group $\mathbf{f}(\cdot) = [f_0(\cdot)\ f_1(\cdot)\ \cdots\ f_{M-1}(\cdot)]^T$, satisfies the

shift property (SP) if it satisfies the (matrix) difference equation

$$\mathbf{f}(n-1) = \mathbf{R}\mathbf{f}(n), \quad n = 1, 2, \cdots, N, \tag{2.2.1}$$

with a specified final condition $\mathbf{f}(N)$, where $\mathbf{R}$ is a constant matrix of size $M \times M$. Furthermore, we will say that a kernel function $\phi(\cdot)$ satisfies SP if there is a kernel group $\mathbf{f}(\cdot)$ that satisfies SP and $\phi(\cdot)$ is an element of $\mathbf{f}(\cdot)$. □

Lemma 2.1 A time-recursive implementation of a kernel group $\mathbf{f}(\cdot)$ is feasible if this kernel group satisfies the shift property.

Proof: (2.2.1) gives:

$$f_p(n-1) = \sum_{q=0}^{M-1} r_{pq} f_q(n), \quad n = 1, 2, \cdots, N, \quad p = 0, 1, \cdots, M-1,$$

where $r_{pq}, p, q = 0, 1, \cdots, M-1$ are the elements of the matrix $\mathbf{R}$. Let

$$X_p(t) = \sum_{n=0}^{N-1} f_p(n) x(t+n-N+1), \quad p = 0, 1, \cdots, M-1 \tag{2.2.2}$$

Suppose this is available at the time instant $t+1$. For the quantities $X_p(t+1)$, $p = 0, 1, \cdots, M-1$ we have:

$$X_p(t+1) = \sum_{n=0}^{N-1} x(t+n+1-N+1) f_p(n) = \sum_{n=1}^{N} x(t+n-N+1) f_p(n-1)$$

$$= \sum_{n=1}^{N} x(t+n-N+1) \sum_{q=0}^{M-1} r_{pq} f_q(n) = \sum_{q=0}^{M-1} r_{pq} \left(\sum_{n=1}^{N} x(t+n-N+1) f_q(n) \right)$$

and therefore we obtain the algorithm:

$$X_p(t+1) = \sum_{q=0}^{M-1} r_{pq} \left[X_q(t) - x(t-N+1) f_q(0) + x(t+1) f_q(N) \right], \tag{2.2.3}$$

where $p = 0, 1, \cdots, M-1$. If we assume knowledge of the boundary values $\{f_q(0), f_q(N),\ q = 0, 1, \cdots, M-1\}$, the algorithm specified in (2.2.3) will become the update computation we were after. (2.2.1) implies that knowledge of $\mathbf{f}(N)$

yields $\mathbf{f}(0)$. Furthermore, note that if $\mathbf{R}$ is nonsingular, knowledge of $\mathbf{f}(0)$ yields $\mathbf{f}(N)$. □

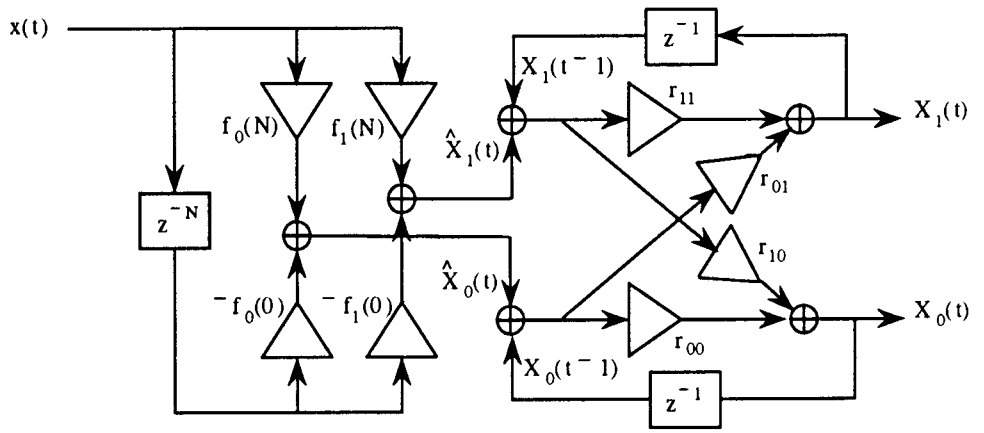


Figure 2.1: Lattice architecture for kernel group of size $M = 2$.

Corollary 2.1 A kernel group $\mathbf{f}(\cdot)$ that satisfies SP can be implemented time-recursively as follows:

1. Compute the matrix $\mathbf{R}$ by evaluating $\mathbf{f}(n-1)$ and using (2.2.1).
2. Evaluate $\mathbf{f}(n)$ at the points $n = 0$ and $n = N$.
3. At each time instant t evaluate (2.2.3). □

Note that the first two steps of the above algorithm belong to the initialization phase (off-line computation). The architectural implementation will have a lattice structure if the size of the kernel group is $M = 2$ (see Fig. 2.1). An example of this architecture appears in [33]. In an abuse of terminology, we will call **lattice architectures** the architectures that implement (2.2.3) regardless of the size of the kernel group.

2.2.2 Time-Recursive Implementations

The issue of specifying a family of kernel groups that satisfy SP is addressed by Lemma 2.2:

Lemma 2.2 The shift property is satisfied by:

1. The singleton kernel group $[cb^n]$, where b and c are non-zero free parameters.

2. The kernel group $[c_{00}b^n + c_{01}b^{-n}, c_{10}b^n + c_{11}b^{-n}]^T$, where b is a non-zero parameter and the coefficients are free parameters, such that $c_{00}c_{11} - c_{01}c_{10} \neq 0$.

3. The kernel group $\left[c_0, c_1 n, \cdots, c_{M-1}n^{M-1}\right]^T$, where the coefficients are non-zero parameters.

4. The union of two kernel groups that satisfy SP.

5. The cartesian product of two kernel groups that satisfy SP. □

The proof of this Lemma can be found in the Appendix.

Suppose now that we are given a mapping operator $[h_0\ h_1\ \cdots\ h_{N-1}]$ for which we have the following linear decomposition:

$$h_n = \alpha\phi(n) + \beta\psi(n), \quad n = 0, 1, \cdots, N-1,$$

and $\phi(\cdot)$, $\psi(\cdot)$ are kernel functions that satisfy SP. One may verify that h_n can not be proved to satisfy SP based on the above linearity property. Nevertheless, we have

$$X(t) = \sum_{n=0}^{N-1} h_n x(t+n-N+1) = \sum_{n=0}^{N-1} \left[\alpha\phi(n) + \beta\psi(n)\right] x(t+n-N+1)$$

$$= \alpha\left[\sum_{n=0}^{N-1} \phi(n)x(t+n-N+1)\right] + \beta\left[\sum_{n=0}^{N-1} \psi(n)x(t+n-N+1)\right] \quad \text{or}$$

$$X(t) = \alpha X_\phi(t) + \beta X_\psi(t),$$

where $X_\phi(t)$ and $X_\psi(t)$ have the obvious definitions. Therefore, we can obtain an efficient time-recursive implementation for the linear combination of two (or more) kernels that satisfy SP, by properly combining the output points of the implementations of the latter. The mapping operators generated by this linearity

property supplement the family of the operators that can be computed in a recursive way based on Lemma 2.2.

Now we will discuss some of the most interesting examples of the time-recursive computation, under the light of Lemma 2.2.

1. We may observe that for $b = c = 1$ the kernel group in Lemma 2.2, Statement 1 specifies the mapping operator that computes the sum of the last N values in the input stream.

2. If $b = e^{j2k\pi/N}$, and $c = 1$ the same case of kernel group specifies the k^{th} frequency component of the DFT. From Fig. 2.2, we conclude that the recursive implementation requires one complex multiplier. Consequently, no more than N complex multipliers are required for the computation of the N point DFT. This result was obtained in [47, pp.175-179] and [7]. It was rediscovered and used in the real-time architecture context in [32], where it is also pointed out that N rotation circuits [52] suffice for the computation of the DFT.

3. Another interesting special case of kernel group is obtained from Lemma 2.2, Statement 2 for

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}e^{j\frac{k\pi}{2N}} & \frac{1}{2}e^{-j\frac{k\pi}{2N}} \\ \frac{1}{2j}e^{j\frac{k\pi}{2N}} & \frac{1}{2j}e^{-j\frac{k\pi}{2N}} \end{bmatrix} \quad \text{and} \quad b = e^{j\frac{k\pi}{N}}. \tag{2.2.4}$$

   For this choice of our constants we obtain $f_0(n) = \cos\frac{k\pi}{N}(n + \frac{1}{2})$ and $f_1(n) = \sin\frac{k\pi}{N}(n + \frac{1}{2})$. One can recognize the kernels of the DCT and the DST. The algorithm for computing the DCT/DST in a time-recursive manner was proposed by [61] and was suggested for the sliding transform computation. In [33] it was realized that time-recursiveness results an efficient architectural design for the DCT/DST. It was shown that such an implementation requires less than $6N$ real multipliers.

4. The fact that the kernel group in Lemma 2.2, Statement 3 can be implemented time-recursively was independently observed in [16] and in [1],

where it is used for estimating the energy spectral density of a noisy waveform via Taylor series.

5. One can use Lemma 2.2, Statement 5 in order to design a lattice architecture for the Lapped Orthogonal Transform (LOT) [36, 38, 39, 62]. The feasibility of such a design is due to the fact that some LOT kernel functions are products of two sinusoidal kernels. The time-recursive implementation of the LOT first appeared in [34]. The implementation of two interesting special cases of the LOT, the Modulated Lapped Transform (MLT) [36] and an Extended Lapped Transform (ELT) [38] are considered in detail in Chapter 4.

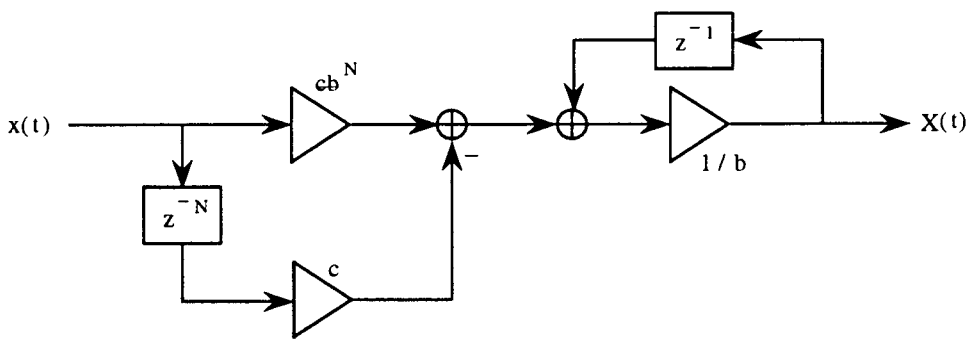


Figure 2.2: Architecture for kernel group of size $M = 1$.

2.2.3 On a Systematic Design I

Surprisingly, we can realize that:

Lemma 2.3 Every mapping operator of finite length N can be implemented in a time-recursive way. □

The proof is given in the Appendix.

A reasonable question follows from this existence result regarding the time-recursive implementation of a given mapping operator. We will proceed with a design procedure that partially answers this question. To do so we will make the assumption that a given operator can be expressed by inspection (and use

of Lemma 2.2) as a linear combination of kernel functions that satisfy SP. For example, the kernel functions of the discrete sinusoidal transforms belong in this class of operators (cf. Lemma 2.2, Statement 2).

Design Procedure

Input:

$$h_n = \sum_i c_i \phi_i(n), \tag{2.2.5}$$

where $\{\phi_i(n)\}$ is a set of kernel functions that satisfy the shift property SP and $\{c_i\}$ is a set of known constants.

Step 1.1:

Specify the kernel groups $\mathbf{f}_i(\cdot)$ to which the kernel functions $\phi_i(\cdot)$ belong. For example, if $\phi_i(n) = n^2$ then, according to Lemma 2.2, Statement 3, we get $\mathbf{f}_i(n) = [1 \; n \; n^2]^T$.

Step 1.2:

For each kernel group $\mathbf{f}_i(\cdot)$ use (2.2.1) in order to compute the matrix of parameters $\mathbf{R}_i$ and evaluate $\mathbf{f}_i(n)$ at the points $n = 0$ and $n = N$.

The outcome of this design procedure is the following algorithm:

1. Evaluate (2.2.3) in order to obtain $X_i(t)$, defined as $X_i(t) = \sum_{n=0}^{N-1} \phi_i(n) x(t+n-N+1)$.

2. Evaluate

$$X(t) = \sum_i c_i X_i(t). \tag{2.2.6}$$

The kernel group associated to the mapping operator is the union $\bigcup_i \mathbf{f}_i(\cdot)$. □

Detailed examples along the lines of this procedure are given in later Chapters.

2.2.4 Mapping Operator Decomposition

In Subsection 2.2.3 we introduced a design procedure for the time-recursive implementation of a class of mapping operators. In this class, the operator coefficients are specified in by a close form description that can be manipulated easily by inspection and compared to the kernel functions in Lemma 2.2. If this

is not feasible, an elaborate technique must be employed to obtain the linear expression required as the input of the design procedure. This is the subject of this Subsection.

One can easily verify that for a given mapping operator a number of different recursive architectural implementations exist. Given a mapping operator, we would like to obtain the optimal time-recursive implementation in terms of the architectural cost. Unfortunately, this is not an easy problem, since a variety of ad-hoc designs may exist for a specified operator. Here, we address the question of optimality with respect to the number of kernels that are used in a linear decomposition of the given mapping operator. We show that this question is equivalent to finding the minimal order partial realization of a proper linear time-invariant (LTI) system. With this approach, we do not assume any prior knowledge of the structure of the coefficients of the given operator.

Lemma 2.4 The size of the smallest kernel group that can be used to implement the mapping operator $[h_0\ h_1\ \cdots\ h_{N-1}]$ in a time-recursive way is equal to the size of the minimal order partial realization of the LTI system with the N first Markov parameters being equal to the coefficients of the specified operator.

Proof: Given a mapping operator $[h_0\ h_1\ \cdots\ h_{N-1}]$, we can have the following coefficient expansion:

$$h_n = \mathbf{c}\mathbf{A}^n\mathbf{b}, \quad n = 0, 1, \cdots, N-1, \tag{2.2.7}$$

where $\mathbf{A}$ is the system matrix of size $M \times M$ and $\mathbf{b}$, $\mathbf{c}$ are the input and output vectors respectively [27, 29]. Let

$$\mathbf{f}(n) = \mathbf{A}^n\mathbf{b} \tag{2.2.8}$$

be a kernel group of size M. Since $\mathbf{f}(n-1) = \mathbf{A}^{n-1}\mathbf{b} = \mathbf{A}^{-1}\mathbf{f}(n)$, this kernel group satisfies the shift property with

$$\mathbf{R} = \mathbf{A}^{-1} \quad \text{and} \quad \mathbf{f}(0) = \mathbf{b}. \tag{2.2.9}$$

From (2.2.7) and (2.2.8) we get the linear decomposition of the mapping operator coefficients $h_n = \mathbf{cf}(n)$. Therefore, the time-recursive implementation of the mapping operator can be based on the kernel group $\mathbf{f}(\cdot)$. In our construction, the size of the kernel group M is equal to the order of the realization $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$. Thus, minimizing the number of the decomposition kernels is equivalent to minimizing the order of the the partial realization of the LTI system, for which the first N Markov parameters are equal to the coefficients of our operator. □

The importance of Lemma 2.4 stems from the fact that the problem of the minimal partial realization has a well known solution [27, 29] (see for an example [27, pp.491-492]). By using this result in combination with Lemma 2.4 we can obtain a time-recursive algorithm for an arbitrary mapping operator based on the minimum number of kernels. The extended algorithm design procedure is described in the following Subsection.

2.2.5 On a Systematic Design II

For the time-recursive implementation of an arbitrary mapping operator $[h_0\ h_1\ \cdots\ h_{N-1}]$ three steps need to be added at the beginning of the design procedure in Subsection 2.2.3:

Design Procedure Supplement

Input:

The mapping operator $[h_0\ h_1\ \cdots\ h_{N-1}]$.

Step 0.1:

Use the algorithm for the minimal order partial realization in order to compute the quantities $\mathbf{A}, \mathbf{b}$ and $\mathbf{c}$ in (2.2.7) [27, 29].

Step 0.2:

Use the similarity transform that will yield $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ in the modal canonical form.

Step 0.3:

Calculate the close form expression for the operator coefficients. □

The expression specified in Step 0.3 can be used as the input in the design procedure described in Subsection 2.2.3.

Note that the algorithm we refer to in Step 0.1 returns a state space description of an LTI system in the controller canonical form (see Subsection 2.1.2). By transforming this system in the modal canonical form we are able to compute the close forms of the elements in matrix $\mathbf{A}^n$ (since this is a block diagonal matrix where the blocks are either rotation matrices or real scalars). Consequently, Step 0.3 can be carried out by simple algebraic manipulations.

2.2.6 The Difference Equation Property

A fundamental property of the Markov parameters $\{h_n = \mathbf{c}\mathbf{A}^n\mathbf{b},\ n = 0, 1, \cdots\}$ of LTI systems dictates:

$$h_{n+M} + \alpha_M h_{n+M-1} + \cdots + \alpha_1 h_n = 0,$$

where $\alpha_p,\ p = 1, 2, \cdots, M$ are the constants specified in (2.1.3) [27]. Equivalently, this can be written in a difference equation format as follows:

$$h_n = \gamma_M h_{n-1} + \cdots + \gamma_1 h_{n-M}, \tag{2.2.10}$$

where

$$\gamma_p = -\alpha_p, \quad p = 1, 2, \cdots, M. \tag{2.2.11}$$

Let $\mathbf{e}_p$ be the row vector of length M, for which the p^{th} element is unity and all other elements equal zero. If vector $\mathbf{c}$ equals $\mathbf{e}_p$ then (2.2.7) implies that h_n is the p^{th} kernel function of the kernel group $\mathbf{f}(\cdot)$. Suppose now that $\mathbf{A}$ and $\mathbf{b}$ are of the form specified in (2.1.3). Then, all kernel functions in (2.2.8) satisfy the same difference equation (2.2.10). The upcoming Lemma 2.5 states that this is true even if $\mathbf{A}$ and $\mathbf{b}$ do not have any special structure. Thus, it introduces the following property of a kernel group:

Difference Equation Property : A kernel group $\mathbf{f}(\cdot) = [f_0(\cdot)\ f_1(\cdot)\ \cdots\ f_{M-1}(\cdot)]^T$, satisfies the difference equation property (DEP) if there are scalars $\gamma_p, p = 1, 2, \cdots, M$, independent of n, such that the kernel functions $f_q(\cdot),\ q =$

$0, 1, \cdots, M-1$ satisfy the following difference equation

$$f_q(n) = \gamma_1 f_q(n-1) + \cdots + \gamma_M f_q(n-M) \quad n = 1, 2, \cdots, N \tag{2.2.12}$$

with some specified initial conditions $f_q(n), n = -1, -2, \cdots, -M$. □

Lemma 2.5 A kernel group satisfies DEP if and only if it satisfies SP. □

The proof of this Lemma is given in the Appendix.

2.3 Design of a Time-Recursive Architecture

2.3.1 Lattice Architecture Design for Mapping Operators

In the previous Section, we have seen how we can specify the kernel group of minimal size that is associated to a given mapping operator and how we can implement this operator by evaluating (2.2.3) and (2.2.6). The algorithm resulted by the design procedure in Subsections 2.2.3 and 2.2.5 can be realized by the lattice architecture introduced in Section 2.1 that evaluates (2.2.3), followed by a simple weighted-sum circuit that evaluates (2.2.6). The lattice architecture that implements a kernel group of size $M = 2$ is shown in Fig. 2.1, while the one that implements a kernel group of size $M = 3$ is depicted in Fig. 2.3. We can observe that this architecture consists of M 2-tap FIR filters and a $M \times M$ weighted interconnection network with M feedback loops. The total cost of this structure is $M^2 + 2M$ multipliers and $M(M-1) + 2M = M^2 + M$ 2-input adders. The weighted-sum circuit consists of M multipliers and $M - 1$ adders. The cost of the overall implementation is given on Table 2.1.

The $M \times M$ weighted interconnection network is characterized by the matrix R specified in (2.2.9). If we follow all five steps of the design procedure described in Subsections 2.2.3 and 2.2.5 the matrix R will be block diagonal with blocks consisted of plain rotations. Consequently, we can implement the interconnection network very efficiently, with locally interconnected rotation circuits. The latter can be realized either with CORDIC processors [23] or with distributed arithmetic techniques [52]. The cost for implementing a mapping operator with

this approach is shown on Table 2.1. Furthermore, with this setup we can exploit the fact that the absolute values of the eigenvalues of a normalized paraunitary system [54] are all equal to 1 [54, 55]. The Discrete Wavelet Transform implementation presented in Chapter 3 takes advantage of this fact to reduce the number of multipliers to be implemented.

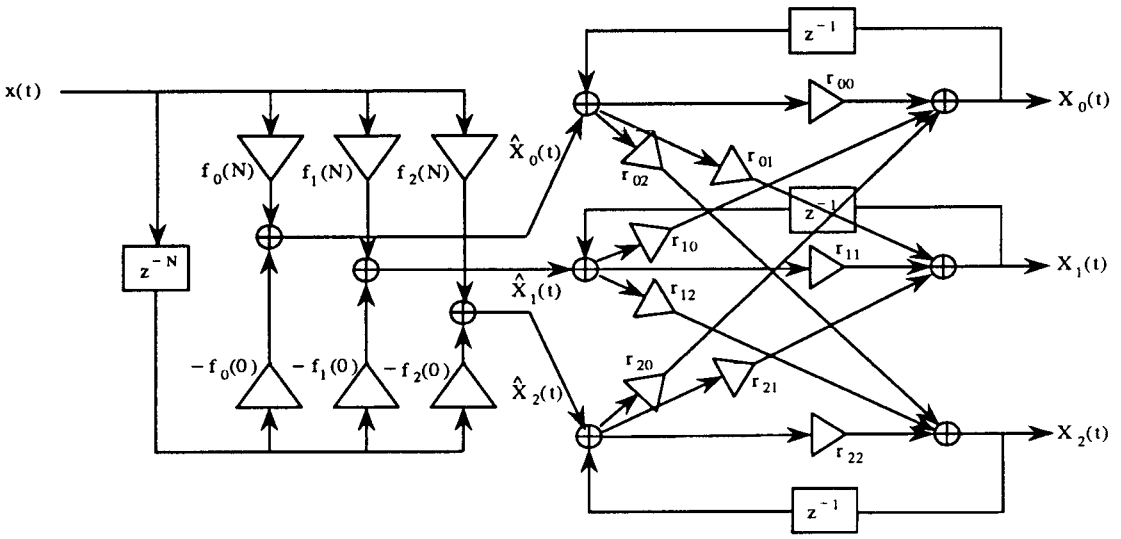


Figure 2.3: Lattice architecture for kernel group of size $M = 3$.

		multipliers	adders	rotations
Case a.	lattice architecture	$M^2 + 3M$	$M^2 + 3M - 1$	-
	lattice / modal	$2M$	$\lfloor 5M/2 + 1 \rfloor$	M
	IIR architecture	$3M$	$3M - 1$	-
Case b.	lattice architecture	$M^2 + 2M + 1$	$M^2 + 2M - 2$	-
	lattice / modal	$2M$	$\lfloor 5M/2 + 1 \rfloor$	$M/2$
	IIR architecture	$2M$	$2M$	-
Case c.	lattice architecture	$M^2 + 2M + 1$	$M^2 + 2M - 3$	-
	lattice / modal	$2M$	$\lfloor 5M/2 \rfloor$	$M/2$
	IIR architecture	$2M$	$2M - 1$	-

Table 2.1: Implementation cost of a mapping operator, based on a kernel group of size M: Case a, the operator does not satisfy the periodicity property and it is utilized by a sliding transform. Case b, the operator satisfies the periodicity property and it is utilized by a sliding transform. Case c, the operator is utilized by a block transform.

2.3.2 The Periodicity Property

A class of special case architectures with important implications to the implementation of the sinusoidal data transforms will be considered. With regard to the structure depicted in Fig. 2.3, suppose that there are two constants D_1 and D_2 such that the relation

$$\hat{X}_p(t) = D_p \hat{X}_0(t), \tag{2.3.1}$$

is true for $p = 1, 2$ and $t = 1, 2, \cdots$. Then, one can verify that the 3 2-tap filters on Fig. 2.3 can be replaced by the structure shown on Fig. 2.4.*a*. The corresponding circuit for $M = 2$ is given on Fig. 2.4.*b*.

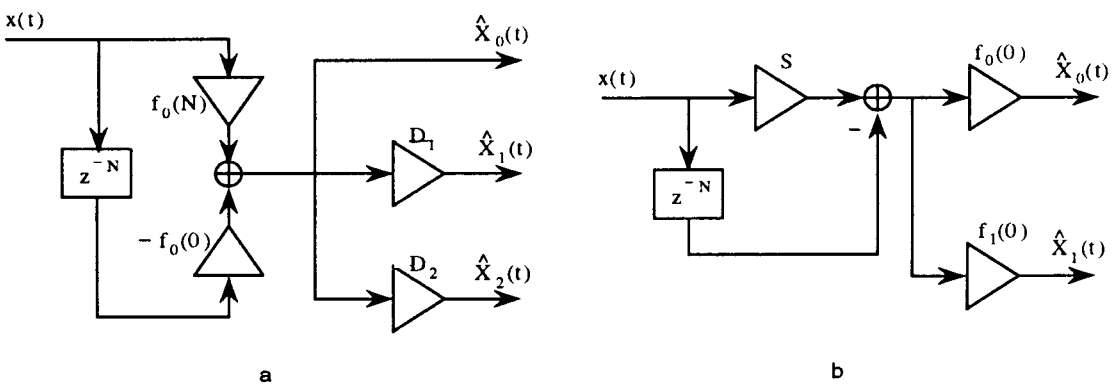


Figure 2.4: Part of lattice architecture if the periodicity property is satisfied.

In this way, $M - 1$ multipliers and an equal number of adders are saved. Obviously, the same trick can be used for a kernel group of arbitrary size. The resulted cost metrics are depicted on Table 1.2. In Lemma 2.6 that follows, we state a condition on the kernel functions that imply (2.3.1) and consequently the savings mentioned above can be obtained. We will see that this condition amounts to satisfaction of the following property:

Periodicity Property: A kernel group $\mathbf{f}(\cdot) = [f_0(\cdot)\ f_1(\cdot)\ \cdots\ f_{M-1}(\cdot)]^T$, satisfies the periodicity property (PP) if the following relation holds:

$$\frac{f_0(N)}{f_0(0)} = \frac{f_1(N)}{f_1(0)} = \cdots = \frac{f_{M-1}(N)}{f_{M-1}(0)} = \frac{1}{S} \tag{2.3.2}$$

for some non-zero constant S. □

Lemma 2.6 Given a kernel group $\mathbf{f}(\cdot)$ relation (2.3.1) holds for $p = 1, 2, \cdots, M-1$ and $t = 0, 1, \cdots$ if $\mathbf{f}(\cdot)$ satisfies the periodicity property. □

The proof is given in the Appendix.

The name "periodicity property" is justified by the following special case: Consider the kernel group specified by Statement 2 in Lemma 2.2, that is,

$$\begin{bmatrix} f_0(n) \\ f_1(n) \end{bmatrix} = \begin{bmatrix} c_{00}b^n + c_{01}b^{-n} \\ c_{10}b^n + c_{11}b^{-n} \end{bmatrix}, \tag{2.3.3}$$

where b is a non-zero parameter and the coefficients are free parameters, such that $c_{00}c_{11} - c_{01}c_{10} \neq 0$. In the Appendix we prove the following Lemma:

Lemma 2.7 If the parameter b of the kernel group (2.3.3) is of the form $b = e^{j\beta}$, then (2.3.3) satisfies the periodicity property if and only if $\beta = j\frac{k\pi}{N}$, that is, if the kernel functions are periodic with period equal to N. Furthermore, if PP is satisfied the ratio value in (2.3.2) is equal to $1/S = (-1)^k$. □

An example of kernel group that satisfies PP is $\mathbf{f}_k(n) = \left[\cos\frac{k\pi}{N}(n+\frac{1}{2})\sin\frac{k\pi}{N}(n+\frac{1}{2})\right]^T$. This is a special case of (2.3.3) for which the values of the constants are specified in (2.2.4). We can implement the pair of DCT and DST based on this kernel group. The importance of the periodicity property will be further appreciated when we see the implications it has on the IIR architecture discussed in the following Subsections.

2.3.3 IIR Architecture Based on the Shift Property

The lattice architecture we have seen in Subsection 2.3.1 constitutes a direct translation of (2.2.3) into an architectural implementation. If a transfer function approach is adopted instead, we obtain an IIR filter structure implementation for (2.1.1) [34]. In this Subsection, we show how we can specify the IIR implementation of a kernel group based on the shift property, while the IIR architecture design based on the difference equation property is the subject of the following Subsection. The IIR architecture often involves less implementation

cost in comparison to the lattice one, especially if the associated kernel group exhibits the periodicity property we have seen in the previous Subsection.

Lemma 2.8 Let $f_p(\cdot)$ be a kernel function in the kernel group $\mathbf{f}(\cdot) = [f_0(\cdot)\ f_1(\cdot)\ \cdots\ f_{M-1}(\cdot)]^T$ of size M. If $\mathbf{f}(\cdot)$ satisfies SP, the kernel function $f_p(\cdot)$ can be implemented by an IIR filter with transfer function $H_p(z)$

$$H_p(z) = \frac{b_p^0(z)}{a(z)} - z^{-N}\frac{b_p^1(z)}{a(z)}, \quad p = 0, 1, \cdots, M-1, \tag{2.3.4}$$

where $a(z)$ is a polynomial in z^{-1} of degree M and b_p^i, $i = 0, 1$ are polynomials in z^{-1} of degree $M-1$, defined as follows: $a(z) = |\mathbf{A}(z)|$, $b_p^i(z) = \left|\mathbf{B}_p^i(z)\right|$, $i = 0, 1$,

$$a(z) = \begin{bmatrix} -1 + r_{00}z^{-1} & r_{01}z^{-1} & \cdots & r_{0,P-1}z^{-1} \\ r_{10}z^{-1} & -1 + r_{11}z^{-1} & \cdots & r_{1,P-1}z^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ r_{P-1,0}z^{-1} & r_{P-1,1}z^{-1} & \cdots & -1 + r_{P-1,P-1}z^{-1} \end{bmatrix}. \tag{2.3.5}$$

$\mathbf{B}_p^i(z)$ is an $M \times M$ matrix that is formed by substituting the p^{th} column of $\mathbf{A}(z)$ with $\left[s_0^i\ s_1^i\ \cdots\ s_{M-1}^i\right]^T$ $i = 0, 1$, where

$$s_p^0 = -\sum_{q=0}^{M-1} r_{pq} f_q(0), \quad s_p^1 = -\sum_{q=0}^{M-1} r_{pq} f_q(N), \quad p = 0, 1, \cdots, M-1.$$

Note that $|\mathbf{X}|$ denotes the determinant of the matrix $\mathbf{X}$. □

The proof is given in the Appendix.

As a direct consequence of this Lemma we have:

Corollary 2.2 Let $\mathbf{f}(\cdot) = [f_0(\cdot)\ f_1(\cdot)\ \cdots\ f_{M-1}(\cdot)]^T$ be a kernel group of size M that satisfies PP. Then, the transfer function $H_p(z)$ of the linear system that models (2.2.3) is:

$$H_p(z) = \left(S - z^{-N}\right)\frac{b_p^1(z)}{a(z)}, \quad p = 0, 1, \cdots, M-1, \tag{2.3.6}$$

where $a(z)$ and $b_p^1(z)$ are specified in Lemma 2.8 and S is the constant specified in (2.3.2). □

For the sake of clarity, we will consider the special case of a kernel group of size $M = 2$ in detail. Let $H_p(z)$ be the transfer function of the linear system that models the mapping operators

$$[f_p(0)\ f_p(1)\ \cdots\ f_p(N-1)]\,,$$

for $p = 0, 1$. From (2.3.5), for $M = 2$ we get:

$$a(z) = \begin{vmatrix} -1 + r_{00}z^{-1} & r_{01}z^{-1} \\ r_{10}z^{-1} & -1 + r_{11}z^{-1} \end{vmatrix}.$$

Furthermore, we have

$$b_0^i(z) = \begin{vmatrix} s_0^i & r_{01}z^{-1} \\ s_1^i & -1 + r_{11}z^{-1} \end{vmatrix}, \quad b_1^i(z) = \begin{vmatrix} -1 + r_{00}z^{-1} & s_0^i \\ r_{10}z^{-1} & s_1^i \end{vmatrix},$$

where

$$s_p^0 = -r_{p0}f_0(0) - r_{p1}f_1(0) \quad \text{and} \quad s_p^1 = -r_{p0}f_0(N) - r_{p1}f_1(N), \quad p = 0, 1.$$

The architectural implementation resulted from (2.3.4) is shown in Fig. 2.5, while for the case where the periodicity property is satisfied, the architecture associated to (2.3.6) is depicted on Fig. 2.6. We observe that the IIR architecture consists of a feedback structure with $M = 2$ delay elements. The parameters $d_i, i = 1, 2$ and $n_{ij}, i = 0, 1, j = 0, 1, 2, 3$ are given by the following expressions:

$$\begin{array}{lll} d_1 = -r_{00} - r_{11} & n_{00} = f_0(N)r_{00} + f_1(N)r_{01} & n_{10} = f_0(0)r_{00} + f_1(0)r_{01} \\ d_2 = r_{00}r_{11} - r_{01}r_{10} & n_{01} = -f_0(N)d_2 & n_{11} = -f_0(0)d_2 \\ & n_{02} = f_0(N)r_{10} + f_1(N)r_{11} & n_{12} = f_0(0)r_{10} + f_1(0)r_{11} \\ & n_{03} = -f_1(N)d_2 & n_{13} = -f_1(0)d_2 \end{array} . \tag{2.3.7}$$

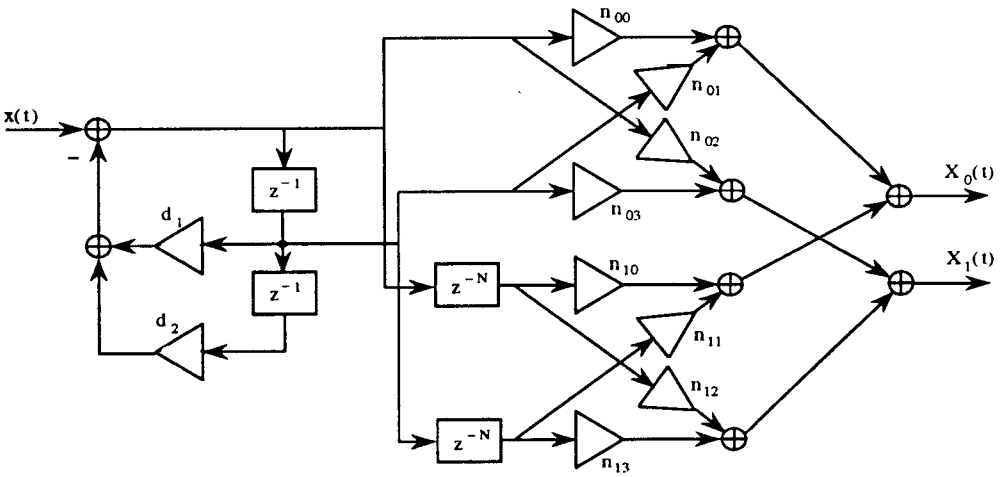


Figure 2.5: IIR architecture for $M = 2$.

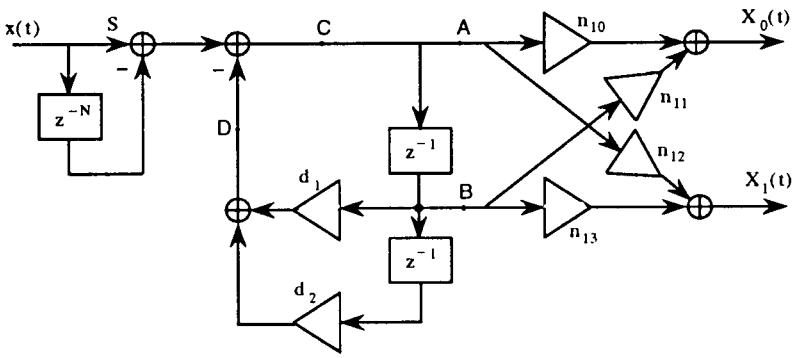


Figure 2.6: IIR architecture for $M = 2$ if the periodicity property is satisfied.

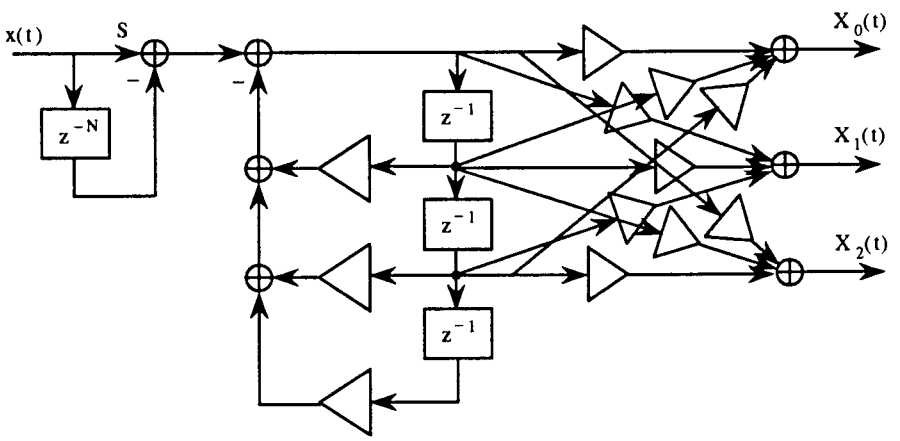


Figure 2.7: IIR architecture for $M = 3$ if the periodicity property is satisfied.

2.3.4 IIR Architecture Based on the Difference Equation Property

An alternative approach to the problem of designing the IIR architecture is based on the defining equation of $X_p(t)$ (2.2.2) and the difference equation property of the kernel group specified in Lemma 2.5. In more concrete terms, we can compute the $\mathcal{Z}$ transform of a kernel function $f_p(n)$ based on the difference equation (2.2.12) and then calculate the transfer function of the system specified by (2.2.2). The following lemmas describe how we can obtain the desired transfer function if we are specified the difference equation parameters. The special case of a difference equation of order $M = 2$ is first considered, the reason being both its importance for a number of practical applications [34] and its simplicity.

Lemma 2.9 Let the kernel function $f_p(\cdot)$ satisfy the second order difference equation

$$f_p(n) = \gamma_1 f_p(n-1) + \gamma_2 f_p(n-2), \qquad n = 1, 2, \cdots, N. \tag{2.3.8}$$

The transfer function $H_p(z)$ of the system specified in (2.2.2) is

$$H_p(z) = \frac{f_p(N-1) + \frac{1}{\gamma_2} f_p(N) z^{-1}}{1 - \frac{\gamma_1}{\gamma_2} z^{-1} - \frac{1}{\gamma_2} z^{-2}} - z^{-N} \frac{f_p(-1) + \frac{1}{\gamma_2} f_p(0) z^{-1}}{1 - \frac{\gamma_1}{\gamma_2} z^{-1} - \frac{1}{\gamma_2} z^{-2}}. \quad \square \tag{2.3.9}$$

The proof of this Lemma was originally given in [34]. In Appendix, we present a proof with more elaborate formalization that suggests the generalization considered in Lemma 2.10.

The parameter values of the associated IIR architecture in Fig. 2.5 is a direct outcome of Lemma 2.9:

$$\begin{array}{llll} d_1 = -\gamma_1/\gamma_2 & n_{00} = f_0(N-1) & n_{10} = f_0(-1) \\ d_2 = -1/\gamma_2 & n_{01} = f_0(N)/\gamma_2 & n_{11} = f_0(0)/\gamma_2 \\ & n_{02} = f_1(N-1) & n_{12} = f_1(-1) \\ & n_{03} = f_1(N)/\gamma_2 & n_{13} = -f_1(0)/\gamma_2 \end{array} \tag{2.3.10}$$

The generalization of Lemma 2.9 for arbitrary values of the order M of the difference equation follows:

Lemma 2.10 Let the kernel function $f_p(\cdot)$ satisfy the M^{th} order difference equation (2.2.12). Then, the transfer function $H_p(z)$ of the system specified in (2.2.2) is given by the expression in (2.3.4), where

$$\begin{aligned} a(z) &= 1 + \sum_{n=0}^{M-1} \frac{\gamma_{M-n}}{\gamma_M} z^{-n} - \frac{1}{\gamma_M} z^{-M} \\ b_p^0(z) &= \sum_{n=0}^{M-1} \left[\frac{1}{\gamma_n} \sum_{q=M-n}^{M} \gamma_q f_p(N+M-n-q-1) \right] z^{-n} \\ b_p^1(z) &= \sum_{n=0}^{M-1} \left[\frac{1}{\gamma_n} \sum_{q=M-n}^{M} \gamma_q f_p(M-n-q-1) \right] z^{-n}. \ \square \end{aligned} \tag{2.3.11}$$

Lemma 2.10 gives a means for computing the IIR parameter values that is considerably easier from the alternative way of carrying out the algebraic computations involved in (2.3.4). Finally, as a direct consequence of Lemma 2.10 we have:

Corollary 2.3 Let the kernel function $f_p(\cdot)$ satisfy:

1. The M^{th} order difference equation (2.2.12).

2. The condition

$$\frac{f_p(N)}{f_p(0)} = \frac{f_p(N-1)}{f_p(-1)} = \cdots = \frac{f_p(N-M+1)}{f_p(-M+1)} = S, \tag{2.3.12}$$

   for some constant S.

Then, the transfer function $H_p(z)$ of the system specified in (2.2.2) is given by (2.3.6), where $a(z)$ and $b_p^1(z)$ are specified in (2.3.11) and S in (2.3.12). $\square$

We may observe that (2.3.12) has the same effect on the IIR architecture with (2.3.2), the defining equation of the periodicity property for a kernel group. This fact suggests the following extension of the definition of the periodicity property:

Periodicity Property (extension): We shall say that a kernel function $\phi(\cdot)$ satisfies the periodicity property (PP) if there is a positive integer M and a non-zero constant S such that

$$\frac{\phi(N)}{\phi(0)} = \frac{\phi(N-1)}{\phi(-1)} = \cdots = \frac{\phi(N-M+1)}{\phi(-M+1)} = S$$

is satisfied. □

A direct consequence of (2.3.2) and (2.3.12) is the following corollary:

Corollary 2.4 If a kernel group satisfies the periodicity property, then the ratio value S in (2.3.2) will be either $S = 1$ or $S = -1$. □

2.3.5 IIR Architecture Design for Mapping Operators

So far, we have discussed the procedure for computing the transfer function that is associated to a given kernel group. We have shown how this transfer function is determined from two different starting points: the matrix difference equation (2.2.1) and the scalar difference equation (2.2.12). In the sequel, we will consider the implementation of the associated mapping operator, which is the goal of our construction. As a direct consequence of (2.2.6), the desired transfer function $H(z)$ is

$$H(z) = \sum_{p=0}^{M-1} c_p H_p(z),$$

where $H_p(z), p = 0, 1, \cdots, M-1$ are the transfer functions of the members of the associated kernel group and $c_p, p = 0, 1, \cdots, M-1$ are specified by the algorithm design procedure. Based on Lemmas 2.8 and 2.10 one can show that

$$H(z) = \frac{1}{a(z)} \sum_{p=0}^{M-1} c_p b_p^0(z) - z^{-N} \frac{1}{a(z)} \sum_{p=0}^{M-1} c_p b_p^1(z), \tag{2.3.13}$$

where the expressions of $a(z)$,$b_p^0(z)$ and $b_p^1(z)$ are described by Lemma 2.8 or by Lemma 2.10, depending on the specifications we are given. In a similar way, based on Corollaries 2.2 and 2.3, one can show that for the case where the associated kernel group satisfies the periodicity property the transfer function

we were after is:

$$H(z) = \left(S - z^{-N}\right) \frac{1}{a(z)} \sum_{p=0}^{M-1} c_p b_p^1(z), \tag{2.3.14}$$

where the expressions of $a(z)$ and $b_p^1(z)$ are specified as above.

We conclude our discussion on IIR architectural implementations with some comments on the implementation cost [1]. For the denominator $a(z)$ in (2.3.13) we need M multipliers and M adders. For the two numerators of this expression we need $2M$ multipliers and $2(M-1)$ adders. An additional adder is needed for the addition in (2.3.13). If the periodicity property is satisfied, the implementation of the numerator in (2.3.14) requires M multipliers and $M-1$ adders. Note that no multiplier is needed for the factor S, since the constant S takes values in $\{1, -1\}$. The overall cost is shown on Table 2.1. A comparison of the lattice and the IIR architectures on the basis of the costs on Table 2.1 will yield the following conclusion: The IIR architecture is better if the periodicity property is satisfied by the underlying kernel group, while the lattice architecture is appropriate for the cases where the above property is not satisfied.

Note that the implicit assumption we have made is that only one kernel function from the associated kernel group participated in the linear expression that specifies the mapping operator in consideration (cf. Eqn. (2.2.5)). A decision rule that encounters all the different factors affecting the proper choice of the architecture is provided in Chapter 4.

2.4 Implementing Sliding and Block Transforms

An $N \times N$ data transform can be viewed as a bank of N mapping operators of length N. A time-recursive implementation of these operators yields a locally interconnected, modular, regular, scalable with N design, with linear cost $O(N)$ (in terms of operator counts). In particular, the constant term underlying the asymptotic cost expression can be made linear in terms of the associated kernel

[1] The IIR structure we consider throughout this paper is the well known type-1 realization and the cost analysis that follows is based on this fact. Nevertheless, any one of the known filter realizations can be used for implementing the transfer functions we specify in this Subsection.

group size M, as manifested by the figures in Table 2.1. In the introductory Chapter, we have distinguished between the sliding and the block transforms. We observe on Table 2.1 that such classification reflects different implementation costs. This is justified as follows.

The output of the operators that implement a block transform are sampled at the time instances $t = 0, N, 2N, \cdots$. Consequently, between two adjacent sampling instances we compute $N - 1$ pieces of data that are neglected. The only purpose of this computation is to have a transition phase to computing the data output at the next time instance that is a multiple of N. Consider now the computation of the first valid output that is at time instant $t = N$. The scenario for producing this output amounts to initializing the memory elements of the time-recursive structure at $t = 0$ and feeding the N first input samples. If we set to 0 the memory elements periodically, with period N, we can periodically imitate the computation of the initialization phase, while being able to produce all the useful output data. The consequence of this observation is a simplification of the time-recursive design for the operators in block transforms: the delay element z^{-N} will never deliver a non-zero quantity and therefore it should be replaced by 0 in (2.3.13) and (2.3.14) (as well as in (2.3.4), (2.3.6), (2.3.9) and (2.3.11)). The architecture designs need to be changed accordingly. For example, both IIR structures in Fig. 2.5 and 2.6 reduce to the one in Fig. 2.8. Similarly, the lattice structure in Fig. 2.1 reduces to the one in Fig. 2.9.

A specific instance of this class of circuits, namely the DFT IIR structure, is the well known Goertzel filter [19, 3, 4].

Observe that the periodicity property has an interesting interpretation in this context: If the mapping operators that implement a data transform satisfy PP, the implementation cost of the block transform is almost identical (it differs by one adder) to the one of the sliding transform.

Note finally that the decimation in Fig. 2.8 lets a substantial part of the circuitry operate at minimum rate (that is N times lower than the input data rate).

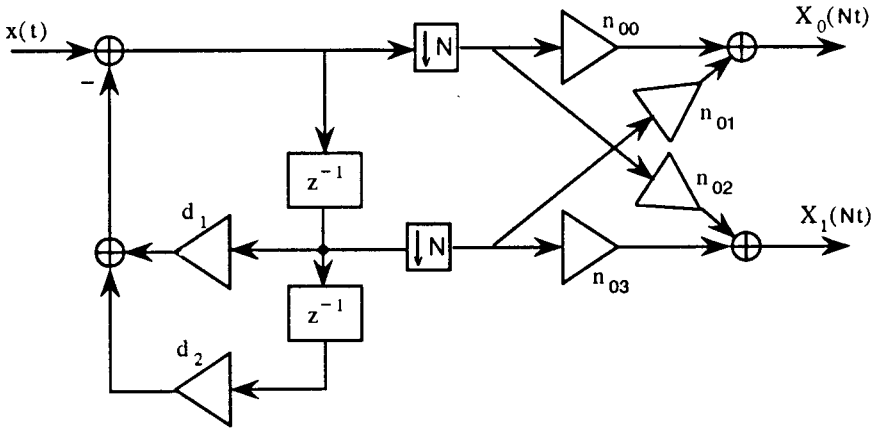


Figure 2.8: IIR architecture for $M = 2$ for an operator used in block transform.

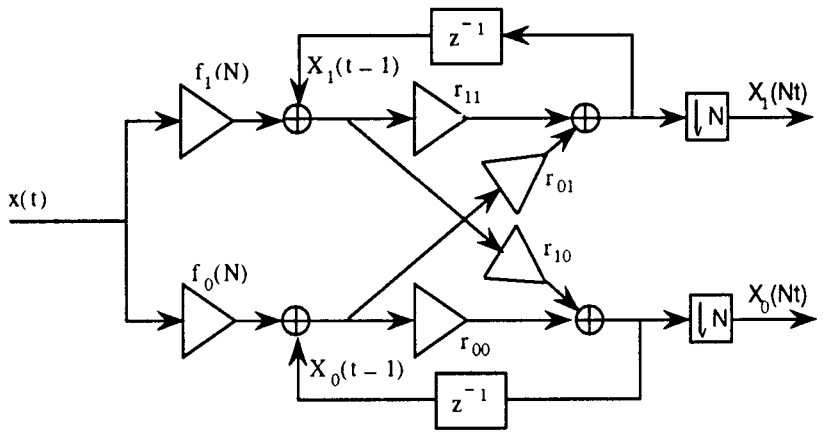


Figure 2.9: Lattice architecture for $M = 2$ for an operator used in block transform.

2.5 Conclusion

In this Chapter, we studied the architectural implementation of the class of discrete time, time invariant, compactly supported, linear (mapping) operators. We showed that the implementation of a given mapping operator is based on the implementation of an associated kernel group and we introduced three properties of kernel groups that are instructive for our design: The shift property (SP), the difference equation property (DEP), and the periodicity property (PP).

We demonstrated the design of a lattice architecture based on SP and the design of an IIR architecture based on either SP or DEP. We realized that PP yields certain cost reduction and it can be used to affect the choice between the two candidate architectural options. We considered the architectures associated to both sliding and block data transforms.

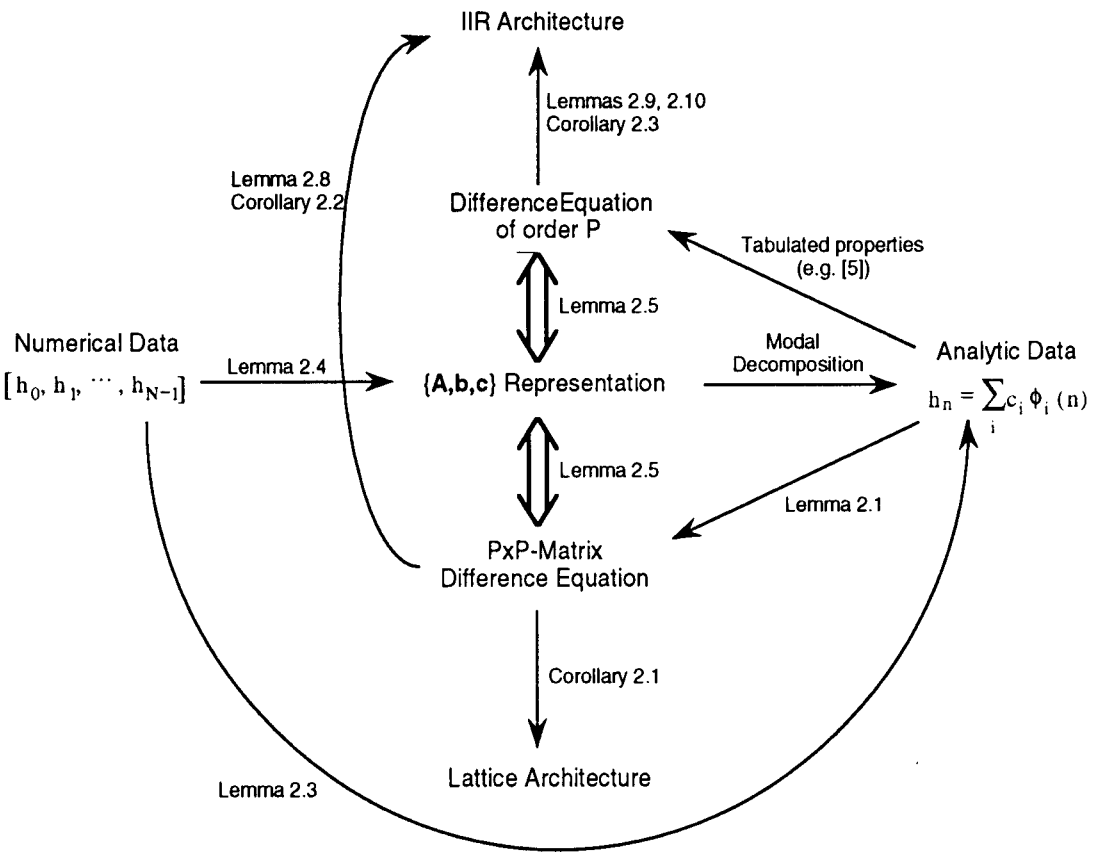


Figure 2.10: Overview of the architecture design procedure.

A comprehensive overview of the design procedure that summarizes the above results is given on Fig. 2.10. The algorithm design procedure suggested in Subsections 2.2.3 and 2.2.5, along with the cost figures on Table 2.1 can be used as design guides. The latter provide an estimate of the constant factor underlying the asymptotic expressions in Tables 1.1 and 1.2 in Chapter 1. Based on this background, an architecture design procedure is developed in Chapter 3 that can be used for routinely obtaining the time-recursive architecture of a given mapping operator.

CHAPTER
3

Design Procedure

The purpose of Chapter 3 is two-fold. First, to integrate the background provided in Chapter 2 into a **Generic Architecture Design Procedure** and second, to demonstrate its usage with applications in real-time SIMO (single-input multiple output) and MISO (multiple-input single-output) architectures for data transforms.

In Section 3.1, we draw a Generic Design Procedure for time-recursive architecture design. In Section 3.2, we discuss an example of the IIR architecture: the Discrete Cosine Transform (DCT) and the inverse DCT. In Section 3.3, we draw the architecture for the Discrete Fourier Transform (DFT), as an example of lattice architecture. Based on the DFT architecture a design for the Cepstral Transform is also derived. For both cases of DCT and DFT the transform kernels can easily be expressed as linear combinations of the kernel functions specified in Lemma 2.2, thus simplifying considerably the design procedure. In Section 3.4, we present an example for which such simple manipulation is not possible and therefore, the systematic approach presented in Subsection 2.2.5 is employed: the Discrete Wavelet Transform (DWT). We conclude with Section 3.5.

3.1 Generic Design Procedure

In this Section, we introduce a Generic Design Procedure for the time-recursive architectures. This is a routine that integrates the distinct parts of the design vehicle we have studied in Chapter 2. The input specification can be either the coefficient vector $[h_0\ h_1\ \cdots\ h_{N-1}]$ or a kernel function of the form $\phi(\cdot) =$

$\sum_i c_i \phi_i(\cdot)$, where the functions $\{\phi_i(\cdot)\}$ belong in the class of functions specified in Lemma 2.2. In Fig. 3.1, we present the flow diagram of the Generic Design Procedure that addresses the architecture design problem for both of these cases. For the majority of interesting applications the specified input expression can be manipulated so that kernel groups of size $M = 2$ suffice for the architecture design. This is desirable, since it implies locality and low complexity in the resulted design. Therefore, the design procedure will be focused on kernel groups of size $M = 2$, that also serves the purpose of simplicity and clarity in this presentation. Nevertheless, we believe it deserves the name "generic" since it considers all different factors that affect the architectural structure. Also, it conveys basic notions, so that the design rules and procedures in Fig. 3.1 can be easily modified to accommodate arbitrary values of M.

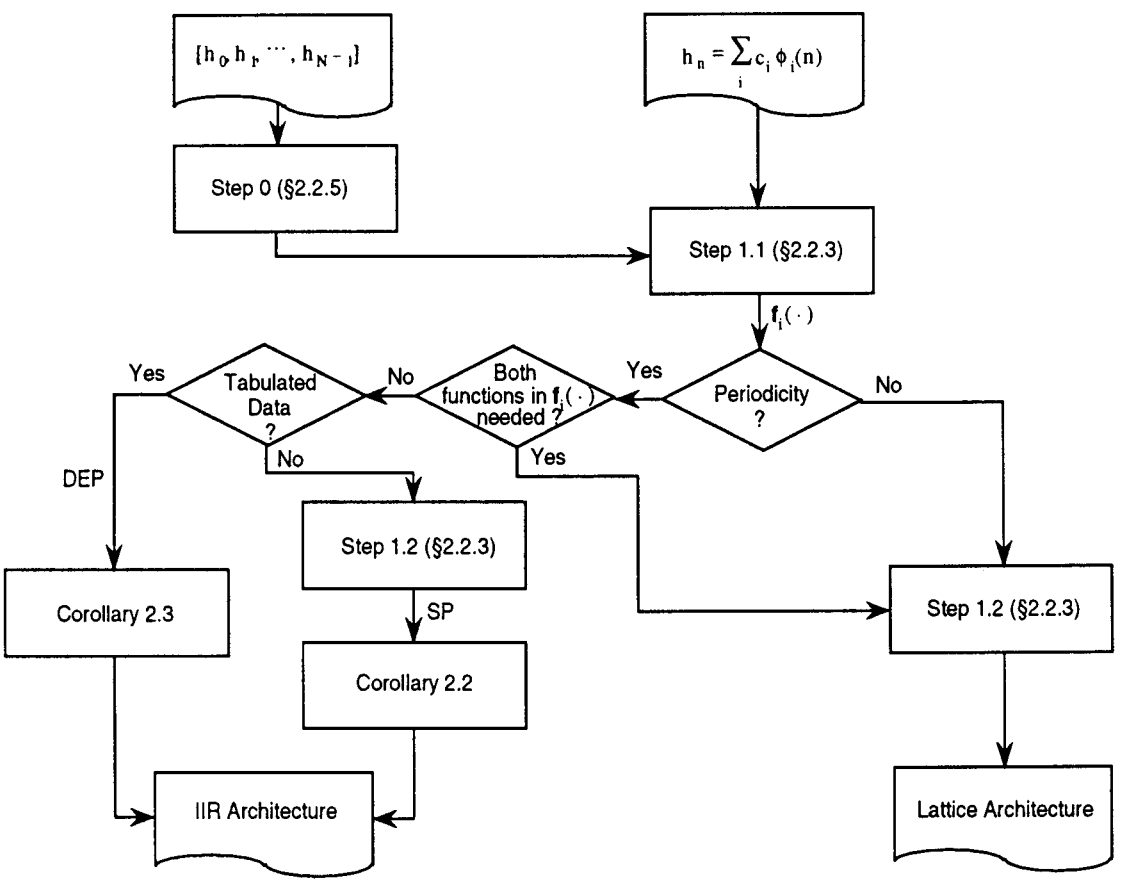


Figure 3.1: Architecture design procedure.

In the sequel, we follow the flow diagram in Fig. 3.1. If the input is specified in the coefficient vector format the preprocessing described in Subsection 2.2.5 has to be employed, resulting an expression of the mapping operator coefficients in terms of sine, cosine and exponential functions. This is labeled with Step 0 in the flow graph in Fig. 3.1. The mapping operator specified as a linear expression of kernel functions (regardless whether this is the output of Step 0 or the provided input) is fed to Step 1.1 described in Subsection 2.2.3. In this step of the design procedure we determine the kernel groups $\{\mathbf{f}_i(\cdot)\}$ that are associated to the functions $\{\phi_i(\cdot)\}$. For each kernel group $\mathbf{f}_i$ we question the periodicity property (PP). If PP is not satisfied the lattice architecture is decided and it is determined by Step 1.2. Otherwise, we question whether both members of $\mathbf{f}_i$ participate in the expression $\phi(\cdot) = \sum_i c_i \phi_i(\cdot)$. If this is the case, then the lattice architecture is preferable, since it performs the computation pertinent to the second kernel function at no additional cost. Furthermore, the resulted lattice structure often comprises of a rotation circuit that can be implemented very efficiently by using a CORDIC processor [23] or distributed arithmetic [52]. On the other hand, the IIR architecture is recommended if one of the kernel functions in $\mathbf{f}_i$ is to be implemented. If this kernel function ϕ_i satisfies the difference equation property (see Subsection 2.2.6) the IIR architecture parameters can easily be determined by using Corollary 2.3 (see also Fig. 2.5 and Eqn. (2.3.10)). Although given a kernel function one can construct the associated difference equation, we suggest this path only if this equation can be found in tabulated form (see for example [5]). Otherwise, a less painful way to determine the IIR architecture parameters is to determine first the corresponding lattice parameters by following Step 1.2 and then using Corollary 2.2 (see also Fig. 2.5 and Eqn. (2.3.7)).

In the following Sections we give a number of architecture design examples of the above design procedure.

3.2 Example A: Discrete Cosine Transform

In this Section, we use the DCT as the first demonstration example of the design procedure we have developed in the previous Section. The latter dictates an IIR architecture. Also, with a novel derivation for the Inverse DCT (IDCT) we show that the basic building module used in the forward DCT can also be used in the implementation of the IDCT.

Among the transform coding schemes the Discrete Cosine Transform is widely adopted in a variety of real-time applications, including data compression [26, 58, 31] and transform domain adaptive filtering [41, 43]. Due to the advances in ISDN network and high definition TV (HDTV) technology, the need of an efficient implementation of the DCT has become a very important question [24]. Many different approaches have been proposed for the implementation of DCT [22, 30, 33, 34, 42, 57, 59]. The time-recursive approach has been proposed for the sliding version of the DCT by [43, 61, 42]. For the sliding DCT the transform coefficients are computed for all displacements of a sliding window over the data stream. This is the situation for the transform domain adaptive filtering [12, 41, 43, 51]. Similar computational structures have recently been proposed for the architectural implementation of the block DCT with application on real-time data compression [33, 9, 34].

3.2.1 SIMO Architecture for the Forward DCT

The N-point Discrete Cosine Transform of a semi-infinite sequence of (real, scalar) data $x(\cdot)$ consists of N semi-infinite sequences $X_{DCT}(k,\cdot), k = 0, 1, \cdots, N-1$ defined as follows:

$$X_{DCT}(k,t) = c_k \sum_{n=0}^{N-1} \cos \frac{k\pi}{N} \left(n + \frac{1}{2}\right) x(t+n-N+1), t = 0, 1, \cdots \qquad (3.2.1)$$

where $c_0 = \sqrt{\frac{1}{N}}$ and $c_k = \sqrt{\frac{2}{N}}, k = 1, 2, \cdots, N-1$. Consequently, the k^{th} frequency component of the DCT is specified by the mapping operator

$$\left[h_n = c_k \cos \frac{k\pi}{N}(n + \frac{1}{2}), n = 0, 1, \cdots, N-1 \right].$$

The magnitude responses of the DCT filter bank for $N = 16$ is given in Fig. 3.2.

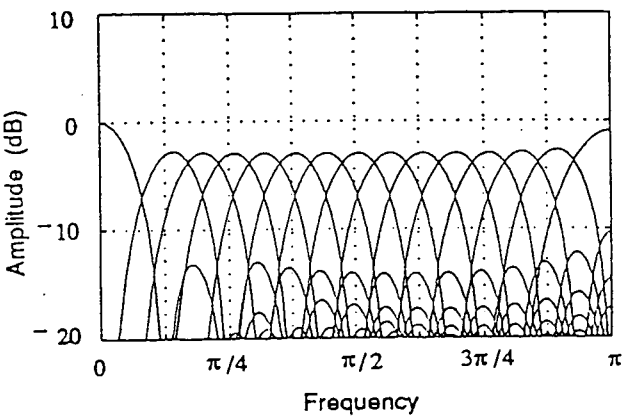


Figure 3.2: The magnitude responses of the DCT filter bank for $N = 16$.

Since we have

$$\begin{bmatrix} \cos \frac{k\pi}{N}(n-1+\frac{1}{2}) \\ \sin \frac{k\pi}{N}(n-1+\frac{1}{2}) \end{bmatrix} = \begin{bmatrix} \cos \frac{k\pi}{N} & \sin \frac{k\pi}{N} \\ -\sin \frac{k\pi}{N} & \cos \frac{k\pi}{N} \end{bmatrix} \begin{bmatrix} \cos \frac{k\pi}{N}(n+\frac{1}{2}) \\ \sin \frac{k\pi}{N}(n+\frac{1}{2}) \end{bmatrix} \tag{3.2.2}$$

the kernel group

$$\begin{bmatrix} f_{k,0}(n) \\ f_{k,1}(n) \end{bmatrix} = \begin{bmatrix} \cos \frac{k\pi}{N}(n+\frac{1}{2}) \\ \sin \frac{k\pi}{N}(n+\frac{1}{2}) \end{bmatrix} \tag{3.2.3}$$

satisfies the shift property. Thus, the mapping operator is specified by the expression $h_n = c_k \phi(n) = c_k f_{k,0}(n)$. This completes the first Step 1.1 of our design procedure. From (3.2.3) we obtain

$$\begin{bmatrix} f_{k,0}(0) \\ f_{k,1}(0) \end{bmatrix} = \begin{bmatrix} \cos \frac{k\pi}{2N} \\ \sin \frac{k\pi}{2N} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} f_{k,0}(N) \\ f_{k,1}(N) \end{bmatrix} = (-1)^k \begin{bmatrix} f_{k,0}(0) \\ f_{k,1}(0) \end{bmatrix}. \tag{3.2.4}$$

Consequently, the periodicity property is satisfied and S in (2.3.2) is $S = (-1)^k$. Note that the same result can be obtained in an easier way by using Lemma 2.7.

Thus, the IIR architecture is recommended (cf. Fig. 3.1).

We have determined two different ways to compute the parameters of this architecture:

1. (3.2.2) yields:

$$\mathbf{R} = \begin{bmatrix} \cos\frac{k\pi}{N} & \sin\frac{k\pi}{N} \\ -\sin\frac{k\pi}{N} & \cos\frac{k\pi}{N} \end{bmatrix}. \tag{3.2.5}$$

   Note that (3.2.4) and (3.2.5) provide the lattice architecture parameters. Based on Corollary 2.2 and more specifically on (2.3.7) we can compute the IIR architecture parameters.

2. The kernel functions in (3.2.3) satisfy the difference equation (2.2.12) with $M = 2$, $\gamma_1 = -1$ and $\gamma_2 = 2\cos\frac{k\pi}{N}$ [5]. The initial conditions for $f_{k,0}(\cdot)$ and $f_{k,1}(\cdot)$ are $f_{k,0}(-1) = f_{k,0}(0) = \cos\frac{k\pi}{2N}$ and $f_{k,1}(-1) = -f_{k,1}(0) = -\sin\frac{k\pi}{2N}$ respectively. The architecture parameters can now be computed based on Corollary 2.3 and consequently on (2.3.10).

Both approaches yield the following parameter values:

$$d_1 = -2\cos\frac{k\pi}{N}, \quad d_2 = 1, \quad n_{10} = -n_{11} = \cos\frac{k\pi}{2N}.$$

The resulted architecture is shown on Fig. 3.3. The circuit that computes the $N = 8$−point sliding DCT is depicted on Fig. 3.4. The block version of the DCT is obtained by introducing downsampling modules at the points A and B in Fig. 3.3, as dictated by the structure in Fig. 2.8 and by omitting the delay element z^{-N} at the input.

Note that the DCT circuit can be viewed as a normalized paraunitary system [55, 54]. Consequently, the associated realizations have their poles on the unite circle [55, 54]. Such phenomenon may cause instability on the resulted IIR implementation. This subtle point can be handled by quantizing the coefficients involved in the design, so that the poles locate inside the unit circle [32].

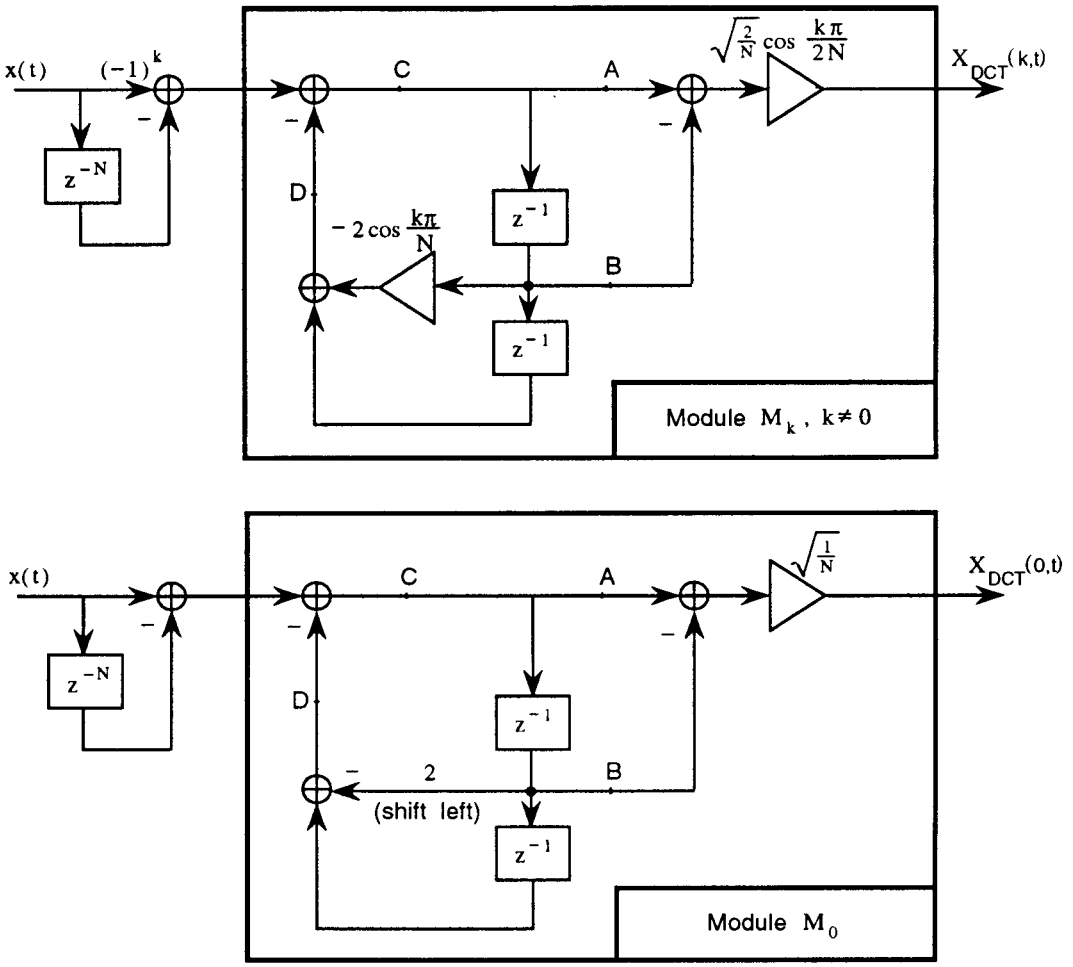


Figure 3.3: IIR architecture for the DCT kernel function.

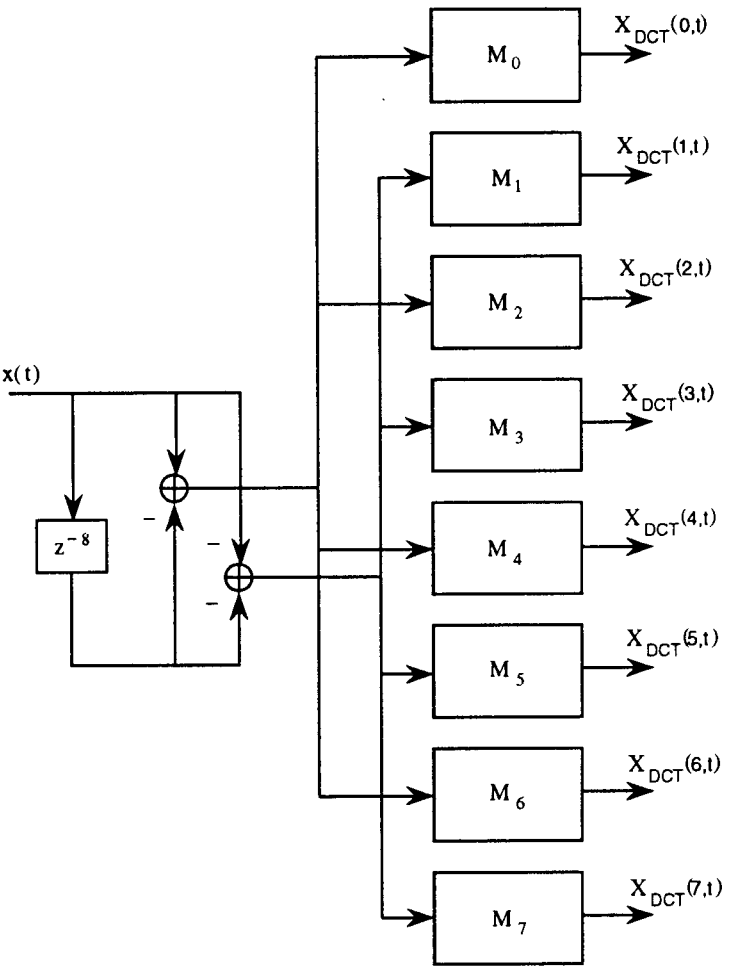


Figure 3.4: Recursive architecture for the DCT.

3.2.2 MISO Architecture for the Inverse DCT

The N-point inverse DCT (IDCT) reconstructs the input sequence points $x(N(t-1)+1), x(N(t-1)+2), \cdots, x(Nt)$ at the time instants $Nt, Nt+1, \cdots, Nt+N-1$ as follows:

$$x(N(t-1)+n+1) = \sum_{k=0}^{N-1} c_k \cos\left[\frac{k\pi}{N}\left(n+\frac{1}{2}\right)\right] X_{DCT}(k, Nt), \quad t = 1, 2, \cdots \tag{3.2.6}$$

This formulation reflects the data model of the block DCT (that is, the DCT coefficients are decimated by a factor equal to N). Before we proceed with the treatment of this case, we would like to consider the simpler but rather

unrealistic case where all the coefficients $\{X_{DCT}(k,t)\}_{k=0}^{N-1}$, $t = 1, 2, \cdots$ produced by a sliding DCT are available. In such case, the reconstruction of the original data sequence $x(t), t = 1, 2, \cdots$ is trivially obtained as follows

$$x(t) = \sum_{k=0}^{N-1} c_k \cos \frac{k\pi}{2N} X_{DCT}(k,t), \quad t = 1, 2, \cdots$$

Nevertheless, the information carried by the coefficients $\{X_{DCT}(k,t)\}_{k=0}^{N-1}$, $t = 1, 2, \cdots$ is highly redundant, thus usually not all the coefficients are available.

In the sequel, we consider the more realistic case of the decimated set of DCT coefficients $\{X_{DCT}(k,Nt)\}_{k=0}^{N-1}$, $t = 1, 2, \cdots$. (3.2.6) is equivalent to

$$x(N(t-1)+n+1) = \sum_{k=0}^{N-1} c_k y_{k,0}(Nt+n+1), \quad t = 0, 1, \cdots, \quad n = 0, 1, \cdots, N-1,$$

where

$$y_{k,0}(Nt+n+1) = f_{k,0}(n) X_{DCT}(k,Nt), \quad k = 0, 1, \cdots, N-1 \tag{3.2.7}$$

and $f_{k,0}(\cdot)$ is defined in (3.2.3).

In this Subsection, we show how we can evaluate the expression in (3.2.7) in a time-recursive way. We consider first the general case of a kernel group of size $M = 2$: $\mathbf{f}(\cdot) = [f_0(\cdot)\ f_1(\cdot)]^T$. Both the lattice and the IIR architectures are discussed.

Suppose that the kernel group $\mathbf{f}(\cdot)$ satisfies SP. Then, (2.2.1) yields

$$\mathbf{f}(n+1) = \mathbf{R}^{-1}\mathbf{f}(n), \quad n = 0, 1, \cdots, N-1. \tag{3.2.8}$$

Let $\mathbf{y}(\cdot)$ be

$$\mathbf{y}(Nt+n+1) = \mathbf{f}(n) X_{DCT}(0,Nt), \quad t = 1, 2, \cdots, \quad n = 0, 1, \cdots, N-1,$$

or

$$\mathbf{y}(Nt+n) = \mathbf{f}(n-1)X_{DCT}(0,Nt), \quad t = 1,2,\cdots, \quad n = 1,2,\cdots,N, \quad (3.2.9)$$

where $X_{DCT}(0,t), t = 0,1,\cdots$ is the transform sequence that corresponds to the 0^{th} kernel function of $\mathbf{f}(\cdot)$. (3.2.8) implies:

$$\mathbf{y}(Nt+n+1) = \mathbf{R}^{-1}\mathbf{f}(n-1)X_{DCT}(0,Nt)$$

and therefore, the quantities $\mathbf{y}(Nt+n+1), n = 0,1,\cdots,N-1$ can be evaluated by the following recursive algorithm:

$$\mathbf{y}(Nt+1) = \mathbf{f}(0)X_{DCT}(0,Nt) \quad (3.2.10)$$

$$\mathbf{y}(Nt+n+1) = \mathbf{R}^{-1}\mathbf{y}(Nt+n), \quad n = 1,2,\cdots,N-1. \quad (3.2.11)$$

This algorithm is implemented by the lattice architecture in Fig. 3.5.a. Note that the parameters $\tilde{r}_{ij}$, $i = 0,1$, $j = 0,1$ are the elements of the matrix $\mathbf{R}^{-1}$.

An IIR implementation of the same algorithm is obtained if we consider the transfer function of the system specified by (3.2.10) and (3.2.11):

Lemma 3.1 Let $\mathbf{f}(\cdot)$ be a kernel group of size 2 that satisfies SP. Then, there is an IIR structure with two delay elements that can implement the algorithm in (3.2.10)-(3.2.11). Furthermore, the coefficients of the transfer function

$$H(z) = \frac{n_0 + n_1 z^{-1}}{1 + d_1 + d_2 z^{-2}},$$

of the system with input $X_{DCT}(0,Nt), t = 1,2,\cdots$ and output $y_0(Nt+n+1)$, $t = 1,2,\cdots$, $n = 0,1,\cdots,N-1$ (that is the sequence of the 0^{th} elements of vector $\mathbf{y}$) are specified by the expressions

$$\begin{aligned} d_1 &= -[\tilde{r}_{00} + \tilde{r}_{11}] & d_2 &= \tilde{r}_{00}\tilde{r}_{11} - \tilde{r}_{10}\tilde{r}_{01} \\ n_0 &= f_0(N) & n_1 &= f_1(N)\tilde{r}_{01} - f_0(N)\tilde{r}_{11} \end{aligned} \quad (3.2.12)$$

where $\tilde{r}_{ij}, i = 0, 1, j = 0, 1$ are the elements of the matrix $\mathbf{R}^{-1}$. □

The IIR architecture is depicted in Fig. 3.5.b.

The circuit that computes $y_{k,0}(\cdot)$ for the IDCT can be designed by a simple parameter substitution in the structure of Fig. 3.5.b. Surprisingly, we realize that the IIR module for the IDCT is identical to the one for the direct DCT. N such modules are needed for the computation of the $N-$point IDCT. The resulted design for the 8−point block IDCT is shown on Fig. 3.6.

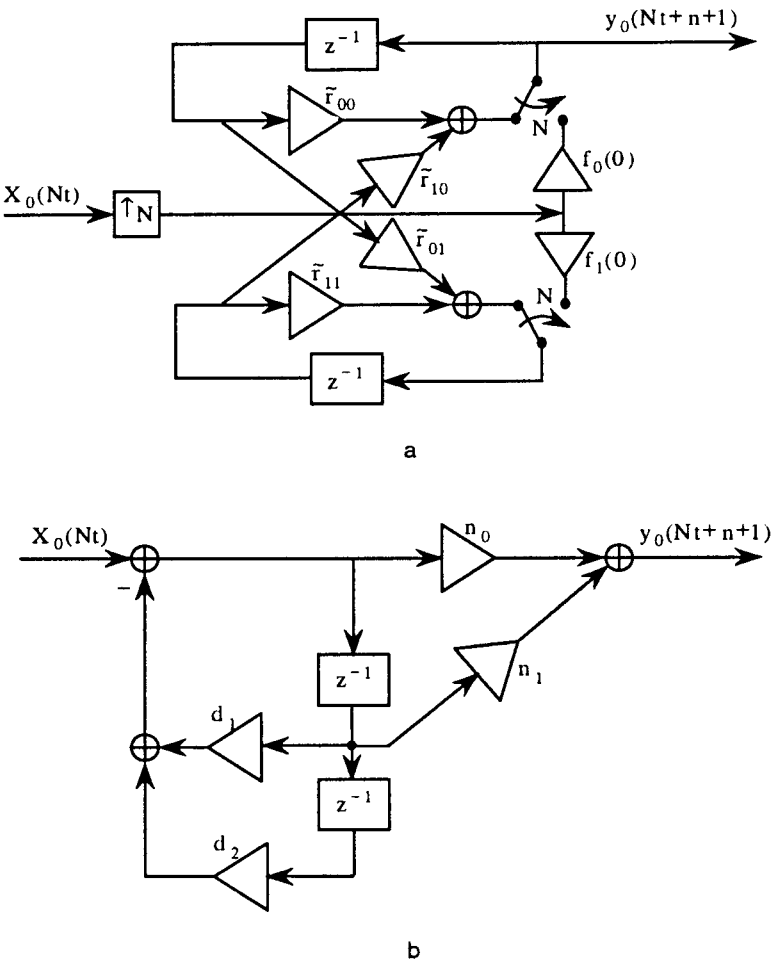


Figure 3.5: Inverse transform module.

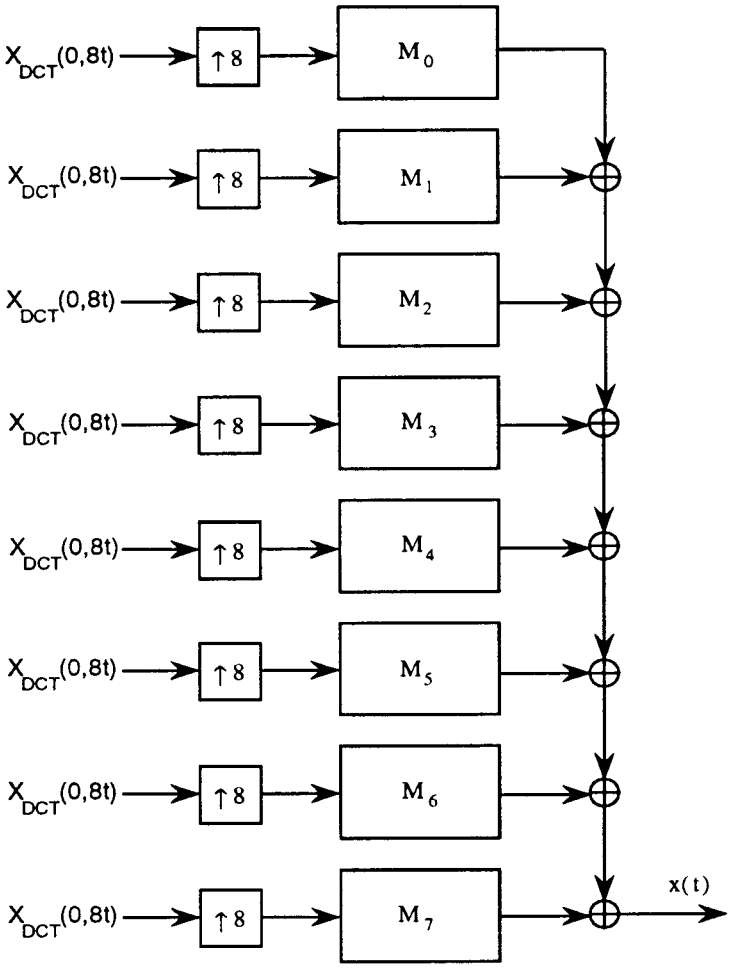


Figure 3.6: Recursive architecture for the IDCT.

3.2.3 Cost Issues

Only two multipliers are needed for the computation of the k^{th} frequency component ($k \neq 0$) of the DCT (cf. Fig. 3.3). The implementation cost for the N-point DCT is $2N - 1$ multipliers and $3N + 2$ adders. These expressions are shown also on Table 3.1. The implementation cost and the throughput expression for a fully pipelined implementation of a fast algorithm proposed in [59] are also given on Table 3.1 [1].

[1] We deliberately have chosen this algorithm as an example of implementation based on a fast algorithm. For a more detailed comparison of DCT algorithms the interested reader may refer to [33].

		rate constraint	implementation cost (mult, add, rotation)
DCT	time-rec.,sliding	$M_P + A_P = u$	$2N-1, 3N+2, 0$
	time-rec.,block	$M_P + A_P = u$	$2N-1, 3N, 0$
	fast algo.,sliding	$M_S + A_S = u$	$\frac{3N}{4}(\log_2 N - 2) + 4$,
	fast algo.,block	$M_S + A_S = Nu$	$\frac{7N}{4}(\log_2 N - 1) + 4, 0$
DFT	time-rec.,sliding	$A_P + R_P = u$	$0, N, N-1$
	time-rec.,block	$A_P + R_P = u$	$0, N-1, N-1$
	fast algo.,sliding	$M_S + A_S = u$	$\frac{N}{2}(\log_2 N - 3) + 2$,
	fast algo.,block	$M_S + A_S = Nu$	$\frac{N}{2}(3\log_2 N - 5) + 4, 0$
DWT	time-rec.,sliding	$2M_P + 7A_P + R_P = u$	$15\log_2 N, 20\log_2 N, 4\log_2 N$

Table 3.1: Cost metrics for the architectural implementation of block transforms. M_P, A_P and R_P denote the time delays associated with a bit-parallel implementation of the multiplier, the adder and the rotation circuit respectively. M_S, A_S and R_S denote the corresponding time delays for a bit-serial implementation.

	time-recursive (mult,add)	FFT-like,sliding (mult,add)
DCT	$2N-1, 3N+2$	$\frac{3N}{4}(\log_2 N - 2) + 4$, $\frac{7N}{4}(\log_2 N - 1) + 4$
DFT	$3N-3, 3N-2$	$\frac{N}{2}(\log_2 N - 3) + 2$, $\frac{N}{2}(3\log_2 N - 5) + 4$
MLT	$5N-3, 5N+3$	$\frac{N}{2}(\log_2 N + 5)$, $\frac{3N}{2}(\log_2 N + 1)$
DWT	$31\log_2 N, 28\log_2 N$	

Table 3.2: Cost metrics (multiplication and addition counts) for the uniprocessor implementation of sliding transforms.

For the derivation and proper interpretation of the throughput rate expressions on the same Table, we need to consider the data model from a closer perspective. In the applications of interest, such as communication of audio, video, sonar and radar data, the input is provided in a serial way. Let us denote with u the time-unit, that is the time lapsing between two adjacent input data. Let us recall also, that the locality property of the time-recursive design makes the bit-parallel implementation of the arithmetic operators feasible. We will denote with M_P and A_P the time required for a multiplication and an addition respectively, as opposed to M_S and A_S that will denote the time needed for the

same operations when implemented in a bit-serial way. The latter must be employed for the architectures based on fast algorithms, since they require global communication. For real-time applications the throughput rate th, that is the number of data samples to be processed per time-unit, needs to be equal to $th = 1$. Otherwise, if $th < 1$, the circuit will not be able to handle all the arriving data, so storage of data and off-line processing will be needed. On the other hand, if $th > 1$, the circuit will be periodically idle since no input data will be available. Under this light of timing information and throughput requirements we will next consider the DCT implementation.

First, for the time-recursive architecture of the sliding DCT circuit the data flow of the structures in Fig. 3.3 and Fig. 3.4 dictate that between two adjacent output data samples two multiplications and four additions have to be performed. By introducing buffers at the points A, B, C and D in Fig. 3.3 pipeline processing is possible, so that the computation time is specified by one multiplication and one addition. Consequently, the real-time processing requirement is equivalent to

$$th = \frac{u}{M_P + A_P} = 1.$$

In other words, the implementation should meet the constraint

$$M_P + A_P = u. \tag{3.2.13}$$

Furthermore, if this is true the latency of the circuit will be equal to $3Nu$. Consider now the fast algorithm design for the sliding DCT that has an inherently parallel-input parallel-output nature. At each time instant an input data arrives, and a sliding window of length N moves to the next position. The real-time processing requirement dictates that before the next input data arrives an $N \times N$ DCT must be performed. An architecture based fast transform consists of a sequence of $\log_2 N$ alternating stages of multiply-add pairs and butterfly interconnection networks. The latency time from input to output for a fully parallel structure will be equal to $\log_2 Nu$. For a fully parallel and pipelined structure each stage can operate on a different set of data, so at a time interval

equal to the latency time above $\log_2 N$ output data sets will be available. In conclusion, the real-time processing requirement will be equivalent to

$$M_S + A_S = u. \tag{3.2.14}$$

A comparison of (3.2.13) with (3.2.14), as well as the latencies of the time-recursive and the architecture based on a fast algorithm yields a strong superiority to the former. In other words, (3.2.13) is substantially easier to meet than (3.2.14). There are two reasons for this:

1. The locality property of the time-recursive architecture allows a shorter internal clock cycle than the one employed by the architecture based on a fast algorithm since the latter makes extensive use of global interconnections.

2. Even if the same clock cycle intervals are used for the two circuits, the ratio $\frac{X_S}{X_P}$ will be of the order of the wordlength used in the finite word length implementation [21]. Here, X_S and X_P denote the time necessary to carry out an operation (either addition or multiplication) with a bit-serial and a bit-parallel implementation respectively. It worths mentioning that the use of distributed arithmetic [60, 52] in implementing bit-parallel operations has been proved to be very effective in the applications of interest [9]. In particular, it is possible to execute one multiplication per clock cycle with this approach.

For the time-recursive implementation of the block DCT the architecture depicted in Fig. 3.3 and Fig. 3.4 needs some modifications as we have mentioned already: the output of the circuit is decimated by a factor equal to N, while the delay element z^{-N} at the input of the structure is not needed, as explained in Subsection 2.4. If the decimation is performed at the points A and B in Fig. 3.3 only one multiplier and two adders need to operate at input data rate, while one multiplier and one adder operate at rate N times lower. By employing the pipeline processing described above we obtain the time lapsing between

two adjacent output samples: $N(M_P + A_P)$. For real-time data processing the time available for this computation is equal to Nu. Consequently, the real-time processing requirement is again specified by (3.2.13). Also, the latency time $3Nu$ as in the case of the sliding DCT. On the other hand, an architecture based on a fast transform will operate as follows. The data will arrive serially, and stored in a buffer of length N. If we assume the buffer feeds the fully parallel and pipelined structure described above. This structure will produce a new set of output data in a time interval equal to $M_S + A_S$. The available time for this computation is equal to Nu. Consequently, the real-time processing requirement dictates

$$th = \frac{Nu}{M_S + A_S} = 1,$$

or

$$M_S + A_S = Nu. \tag{3.2.15}$$

The latency time in the pipelined structure will be $\log_2 N(M_S + A_S)$. We need to add on this the waiting time of the data in the input buffer. This ranges from $1u$ to Nu with an average (approximately) $\frac{N}{2}$. In total, the (average) latency time is $N(\log_2 N + \frac{1}{2})u$. Apparently, the constraint in (3.2.15) is substantially easier to meet from the corresponding one of the sliding DCT (3.2.14). On the other hand, a comparison with (3.2.13) under the light of the comments on the implementation of arithmetic operators made above, yields the conclusion that the two constraints have comparable difficulty. Nevertheless, this comparison should not be viewed in isolation from the companion problem of area minimization, as well as the properties of modularity, regularity and scalability in N. In this perspective, the time-recursive approach suggests very competitive designs for block transform circuits [33, 9], that have been proved to be asymptotically optimal [34].

3.3 Example B: Discrete Fourier Transform

In this Section we present the design of a lattice architecture for the DFT. The time-recursive implementation of the DFT has been considered both in the context of adaptive filtering [7, 10, 43] and spectrum analysis [32, 47]. Here we also consider the architecture of the Inverse DFT (IDFT). Finally, we demonstrate how these architectures can be incorporated by an architectural implementation of the Cepstral Transform [46].

3.3.1 SIMO Architecture for the Forward DFT

The N-point DFT of a semi-infinite sequence of (real, scalar) data $x(\cdot)$ consists of N complex semi-infinite sequences $X_{DFT}(k,\cdot), k = 0, 1, \cdots, N-1$ defined as follows

$$X_{DFT}(k,t) = \sqrt{\frac{1}{N}} \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}kn} x(t+n-N+1), \quad t = 0, 1, \cdots.$$

So

$$\begin{aligned} \mathcal{R}\{X_{DFT}(k,t)\} &= \sqrt{\frac{1}{N}} \sum_{n=0}^{N-1} \cos\frac{2\pi}{N}kn \; x(t+n-N+1) \\ \mathcal{I}\{X_{DFT}(k,t)\} &= -\sqrt{\frac{1}{N}} \sum_{n=0}^{N-1} \sin\frac{2\pi}{N}kn \; x(t+n-N+1), \end{aligned}$$

where $\mathcal{R}\{\cdot\}$ and $\mathcal{I}\{\cdot\}$ denote the real and the imaginary part of the bracketed quantity. Consequently, we seek the implementation of the following two mapping operators

$$\begin{aligned} h_{k,n} &= \sqrt{\tfrac{1}{N}} \cos\tfrac{2\pi}{N}kn \\ g_{k,n} &= -\sqrt{\tfrac{1}{N}} \sin\tfrac{2\pi}{N}kn, \end{aligned} \tag{3.3.1}$$

$k = 0, 1, \cdots, N-1$. Let

$$\begin{bmatrix} f_{k,0}(n) \\ f_{k,1}(n) \end{bmatrix} = \begin{bmatrix} \cos\frac{2\pi}{N}kn \\ \sin\frac{2\pi}{N}kn \end{bmatrix}$$

specify the associated kernel groups $\mathbf{f}_k(n), k = 0, 1, \cdots, N-1$. For the kernel group $\mathbf{f}_k(\cdot)$ we have $\mathbf{f}_k(n-1) = \mathbf{R}\mathbf{f}_k(n)$, where

$$\mathbf{R}_k = \begin{bmatrix} \cos\frac{2k\pi}{N} & \sin\frac{2k\pi}{N} \\ -\sin\frac{2k\pi}{N} & \cos\frac{2k\pi}{N} \end{bmatrix}. \tag{3.3.2}$$

We also have

$$\begin{bmatrix} f_{k,0}(0) \\ f_{k,1}(0) \end{bmatrix} = S \begin{bmatrix} f_{k,0}(N) \\ f_{k,1}(N) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{3.3.3}$$

where $S = 1$. Therefore, the periodicity property is satisfied. Since both member functions of the kernel group appear in the mapping operator decomposition (3.3.1), the lattice architecture is recommended (cf. Fig. 3.1). In Fig. 3.7, we provide the lattice architecture module that is used as the building block of the DFT architectural implementation. The latter is depicted on Fig. 3.8 for the case of $N = 8$. Observe that the lattice structures specify rotation operations.

3.3.2 MISO Architecture for the Inverse DFT

For the inverse discrete Fourier transform of a sequence of complex N-vectors $X_{DFT}(k, Nt), k = 0, 1, \cdots N-1$ and $t = 0, 1, \cdots$ we have

$$x(N(t-1)+n+1) = \sqrt{\frac{1}{N}} \sum_{k=0}^{N-1} X_{DFT}(k, Nt) e^{j\frac{2\pi}{N}kn}. \quad n = 0, 1, \cdots, N-1, \tag{3.3.4}$$

The time-recursive realization of (3.3.4) can be obtained in a way similar to the one followed for the IDCT (see Subsection 3.2.2). From (3.3.4) we get

$$\begin{aligned} &x(N(t-1)+n+1) = \\ &\sqrt{\frac{1}{N}} \sum_{k=0}^{N-1} \left(x_0(k, N(t-1)+n+1) + j x_1(k, N(t-1)+n+1) \right), \end{aligned} \tag{3.3.5}$$

where

$$\begin{bmatrix} x_0(k, N(t-1)+n+1) \\ Y_1(k, N(t-1)+n+1) \end{bmatrix} = X_0(k, Nt) \begin{bmatrix} f_{k,0}(n) \\ f_{k,1}(n) \end{bmatrix} + X_1(k, Nt) \begin{bmatrix} -f_{k,1}(n) \\ f_{k,0}(n) \end{bmatrix} \tag{3.3.6}$$

and $X_0(k,t)$ and $X_1(k,t)$ are the real and imaginary parts of $X_{DFT}(k, Nt)$.

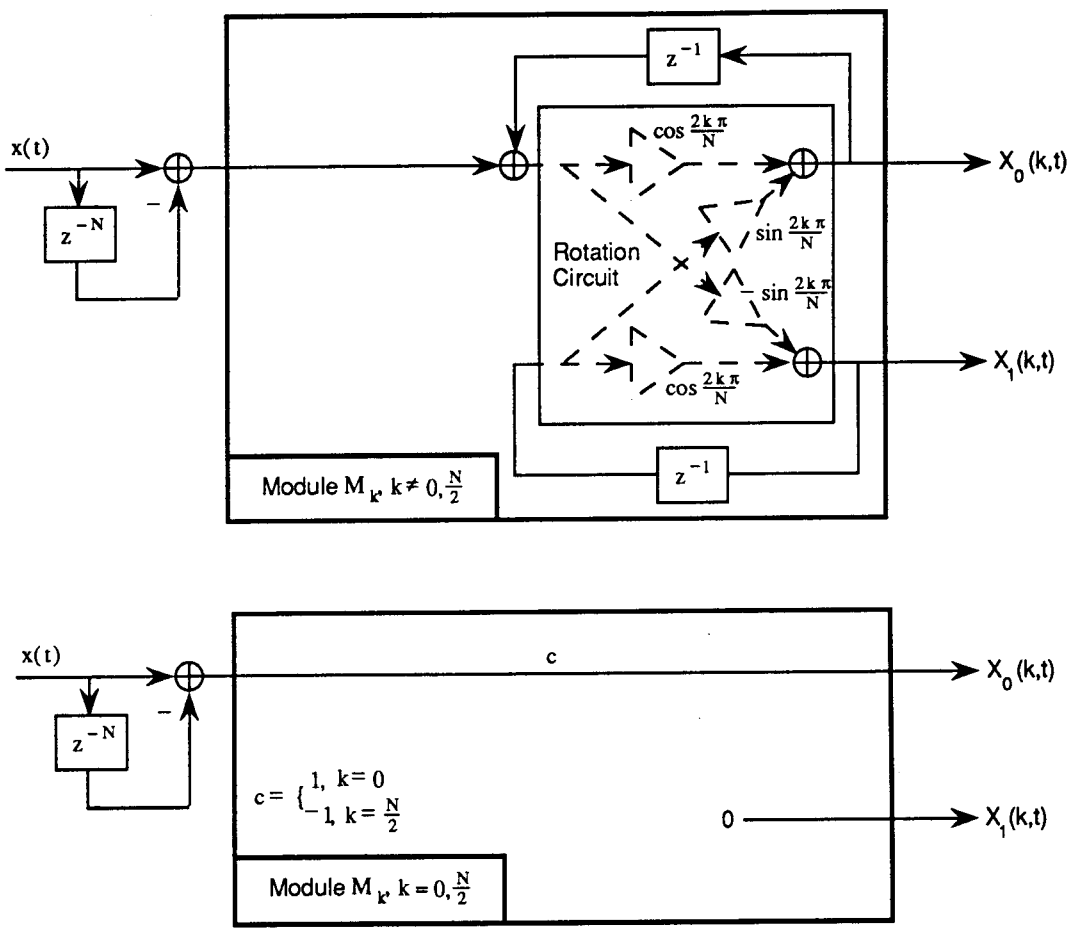


Figure 3.7: Lattice architecture for the DFT module.

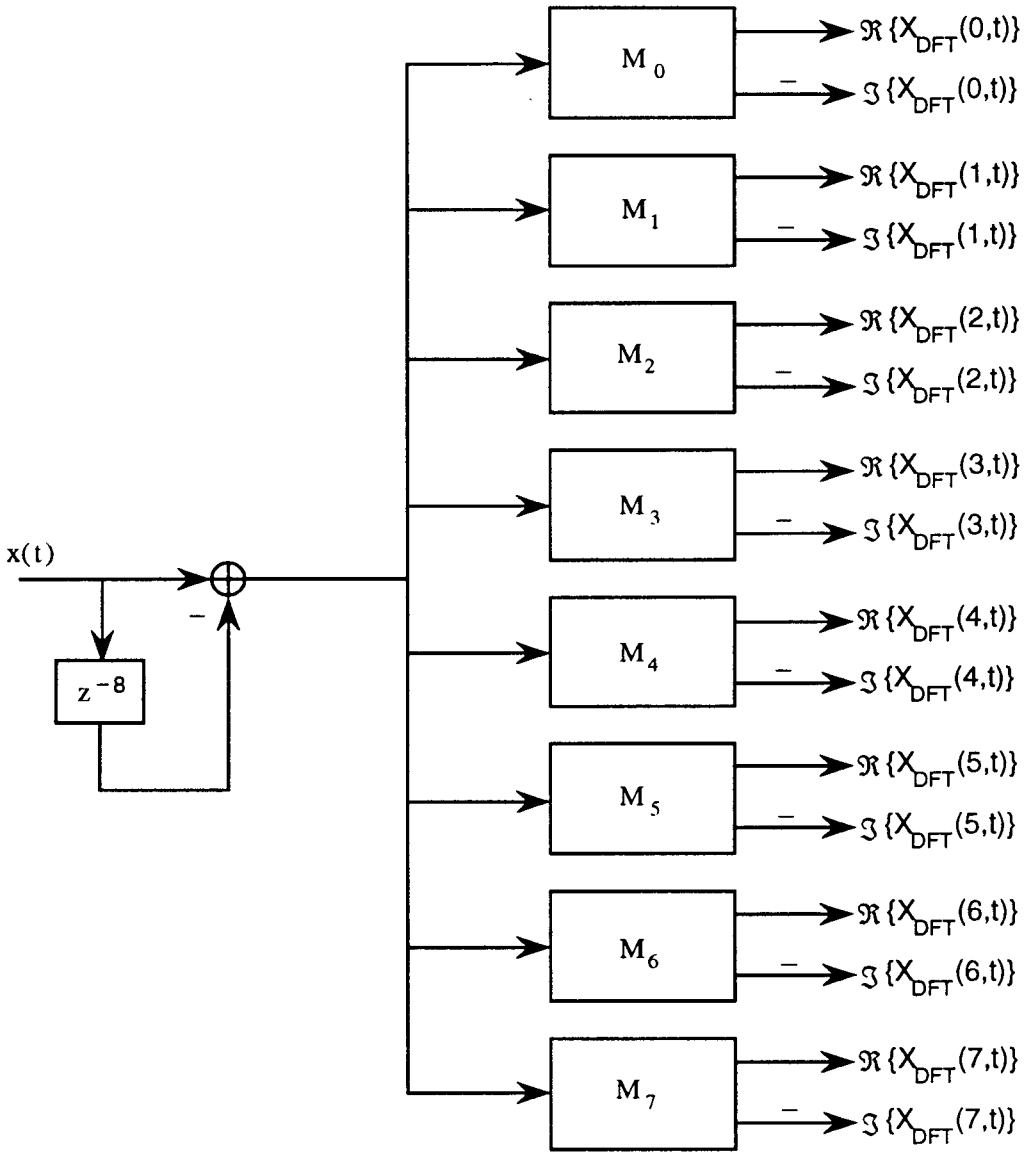


Figure 3.8: Recursive architecture for the DFT.

The computation in (3.3.6) can be implemented by the scheme in Fig. 3.9. The details for the modules $\widetilde{M}_k, k = 0, 1, \cdots, N-1$ are given in Fig. 3.10. The latter are capable of evaluating the summands in (3.3.4) for the real input

$X_i(k,t), i \in \{0,1\}$. This is justified by the fact that (for real input $X_i(k,t)$):

$$\begin{bmatrix} x_0(k, N(t-1)+n+1) \\ x_1(k, N(t-1)+n+1) \end{bmatrix} = X_i(k,t)\mathbf{f}_k(n)$$

$$= \begin{cases} X_i(k,Nt)\mathbf{R}_k^{-1}\mathbf{f}_k(n-1) = \mathbf{R}_k^{-1} \begin{bmatrix} x_0(k, N(t-1)+n+1) \\ x_1(k, N(t-1)+n+1) \end{bmatrix}, & 1 \le n < N \\ X_i(k,Nt)\mathbf{f}_k(0), & n = 0. \end{cases}$$

The resulting time-recursive architecture for the size $N = 4$ IDFT is shown in Fig. 3.11.

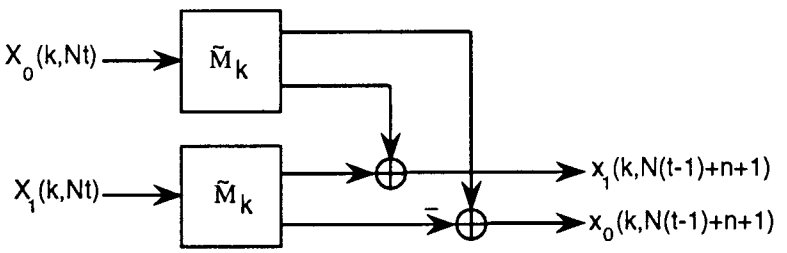


Figure 3.9: IDFT module for complex input.

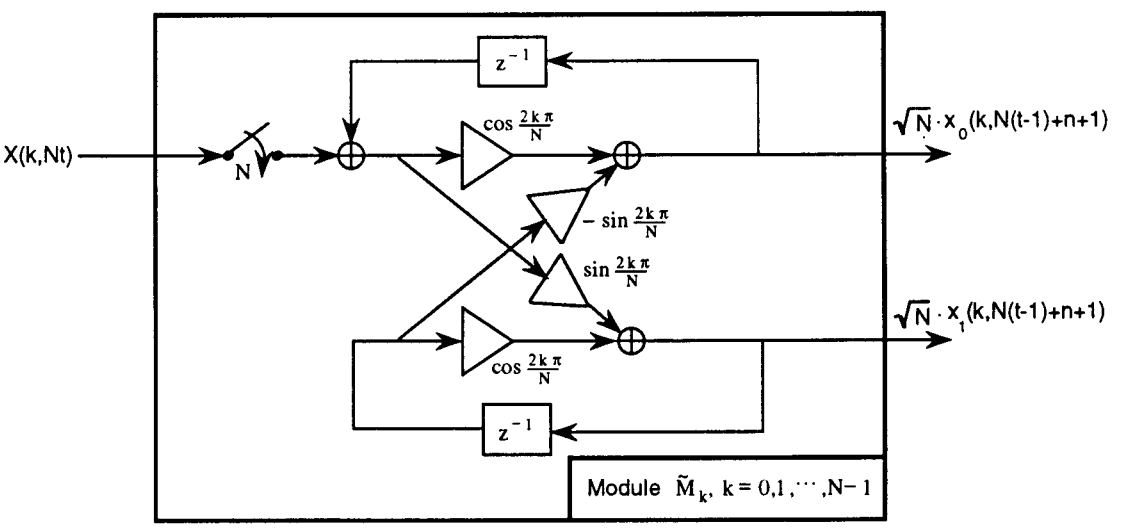


Figure 3.10: Details for real input IDFT module.

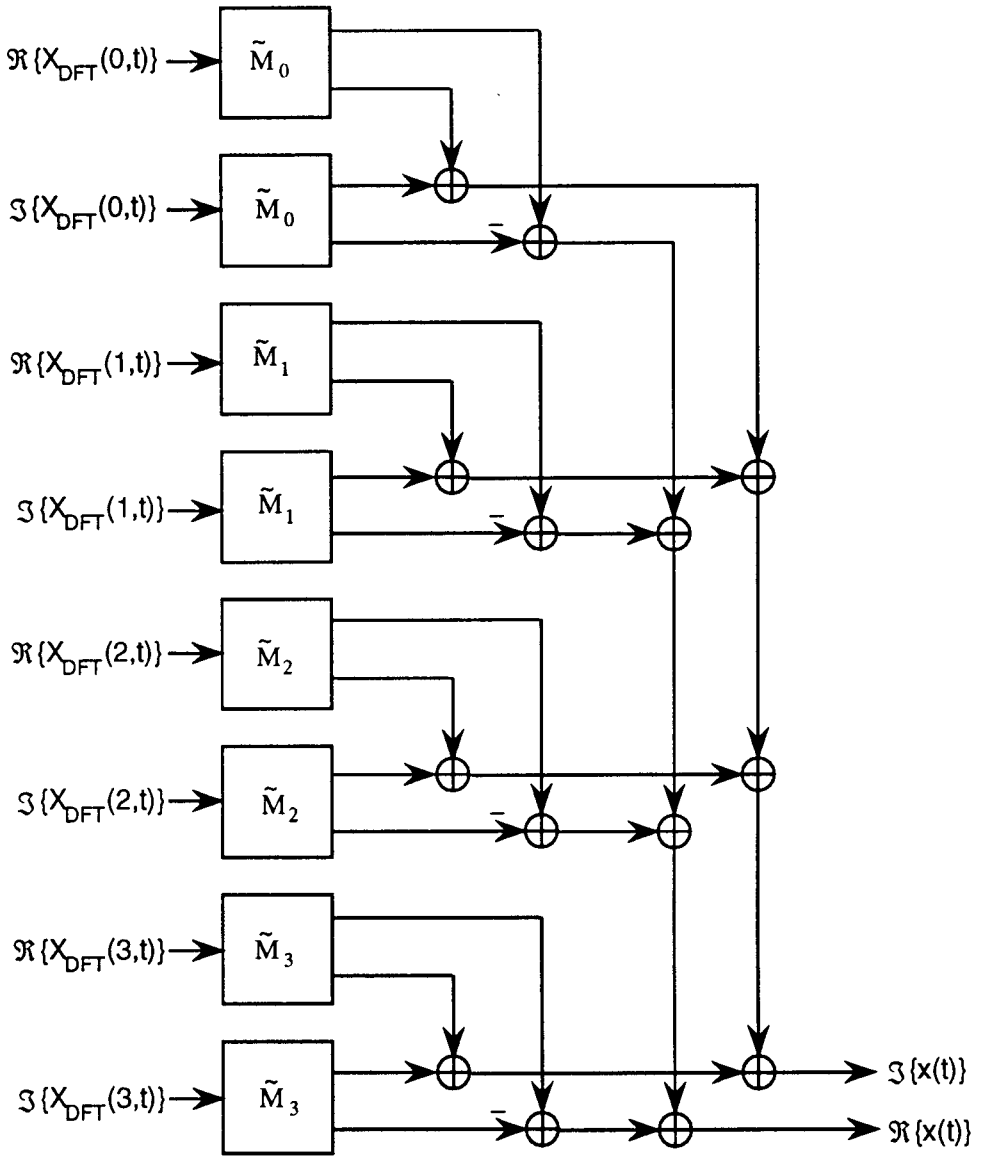


Figure 3.11: Recursive architecture for the IDFT.

3.3.3 SIMO Architecture for the Inverse DFT

A different input data model is considered in this Subsection. Suppose that the sequence of N-vectors $X_{DFT}(k, Nt), k = 0, 1, \cdots N-1$ and $t = 0, 1, \cdots$ are supplied in a serial manner, so that the input consists of a semi-infinite sequence of data $y(t), t = 0, 1, \cdots$. For the sake of simplicity assume real valued data. The

function of the k^{th} IDFT operator is specified as follows:

$$Y_{IDFT}(k,t) = \sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}kn} y(t+n-N+1), \quad t = 0, 1, \cdots.$$

So, the real and the imaginary parts of the output data respectively are:

$$\mathcal{R}\{Y_{IDFT}(k,t)\} = \textstyle\sum_{n=0}^{N-1} f_{k,0}(n) y(t+n-N+1), \quad t = 0, 1, \cdots$$
$$\mathcal{I}\{Y_{IDFT}(k,t)\} = \textstyle\sum_{n=0}^{N-1} f_{k,1}(n) y(t+n-N+1), \quad t = 0, 1, \cdots,$$

where

$$\begin{bmatrix} f_{k,0}(n) \\ f_{k,1}(n) \end{bmatrix} = \begin{bmatrix} \cos\frac{2\pi}{N}kn \\ \sin\frac{2\pi}{N}kn \end{bmatrix} \triangleq \mathbf{f}_k(n).$$

The time-recursive implementation of the kernel group $\mathbf{f}_k(n)$ is already studied in Subsection 3.3.1. More specifically, Fig. 3.7 depicts the lattice architecture for the building module. The architecture for the IDFT differs from the DFT architecture (see Fig. 3.8) only by the sign of the lower (imaginary) output of the module. This is minus $(-)$ for the DFT and plus $(+)$ for the IDFT case. Furthermore, observe that the input are transform coefficients associated with non overlapping blocks of the original data. Consequently, this SIMO (single-input multiple-output) architecture can operate only in the context of a block transform (as opposed to the sliding transform).

3.3.4 Cost Issues

For the computation of the k^{th} frequency component ($k \neq 0$) of the DFT one rotation circuit and one real adder are needed (cf. Fig. 3.7). The implementation cost of the N-point DFT is $N-1$ rotation circuits and N adders. By letting u be the time unit, the real-time processing requirement is expressed by the throughput rate constraint depicted in Table 3.1. The implementation cost and the throughput rate constraint for a fully pipelined implementation of the split radix FFT algorithm in [13] are also given on Table 3.1. All the comments we have made in Subsection 3.2.3 for the DCT apply for the implementation of the

DFT as well.

3.3.5 Cepstral Transform Architecture

The size N Cepstral Transform $y(t), t = 0, 1, \cdots$ of a real valued sequence $x(t), t = 0, 1, \cdots$ is defined as follows [46]

$$y(t) = IDFT\{\log(DFT\{x(t)\})\}, \quad t = 0, 1, \cdots. \tag{3.3.7}$$

The discrete Fourier transform operates on N subsequent samples of the input sequence $x(t+n), n = 0, 1, \cdots, N-1$.

The log operator in (3.3.7) denotes the complex logarithm of a complex argument. We do not address the question of implementing this *log* operator here. We assume that we are provided a logarithm circuit that accommodates properly the phase wrapping requirements of the application that uses our Cepstral transform design [54]. We believe that such a circuit can be built with one of the following approaches:

1. Use a CORDIC processor.

2. Use a distributed arithmetic approach and ROM tables.

3. Use an analog nonlinearity.

The choice of the logarithm circuit will depend on the throughput requirements, as we will see later.

With the use of the DFT and IDFT modules we presented in the previous Sections, we can have two variations in the architecture for the Cepstral transform depending on the speed the logarithm module can operate. First, suppose that the latter can operate at a throughput rate equal to the data input rate, that is it can perform one operation per time unit. This can be true for an implementation of the logarithm based on distributed arithmetic, or an analog nonlinearity. The resulted architecture of the Cepstral transform for $N = 4$ is shown in Fig. 3.12. The DFT modules feed the logarithm elements with one

datum per time unit. This implies that the time variable n in (3.3.4) can always be $n = 0$, since the t variable increments by 1 at every time instant. In this case, the modules $\widetilde{M}_k, k = 0, 1, \cdots, N-1$ vanish. The implementation cost is $N-2$ rotation circuits, $1 + N + 2(N-1) = 3N - 1$ adders and N "fast" logarithm circuits [2].

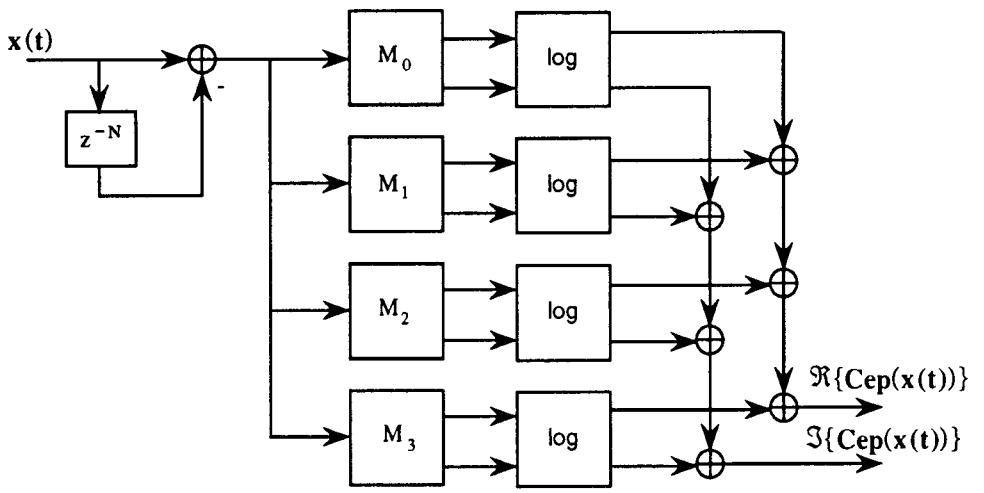


Figure 3.12: Cepstral transform architecture based on fast logarithm circuit.

Second, suppose that the logarithm element can operate at a throughput rate N times lower than the data input rate, that is, it can perform one operation every N time units. This can be true for a CORDIC implementation of the logarithm circuit. The resulted architecture for $N = 4$ is shown in Fig. 3.13. The output of the DFT modules is fully decimated and fed to the logarithm elements. The latter produce an output every N time instants which is fed to the IDFT modules. In this case, the time variable n takes values in the range $n = 0, 1, \cdots, N-1$ and t is incremented by N every N time instants The implementation cost of this architecture is $3N - 6$ rotation circuits, $6N - 1$ adders and N "slow" logarithm circuits.

Both architectures are very suitable for VLSI implementation that can achieve high operation speeds and they can be used for real-time computation of the

[2]Note that the logarithm circuit needs to incorporate the $\sqrt{N}$ factor associated with the DFT and the IDFT. In other words, the logarithm circuit needs to implement the expression $\frac{1}{\sqrt{N}} \log(\frac{1}{\sqrt{N}} x)$, where x denotes a complex number.

Cepstral transform.

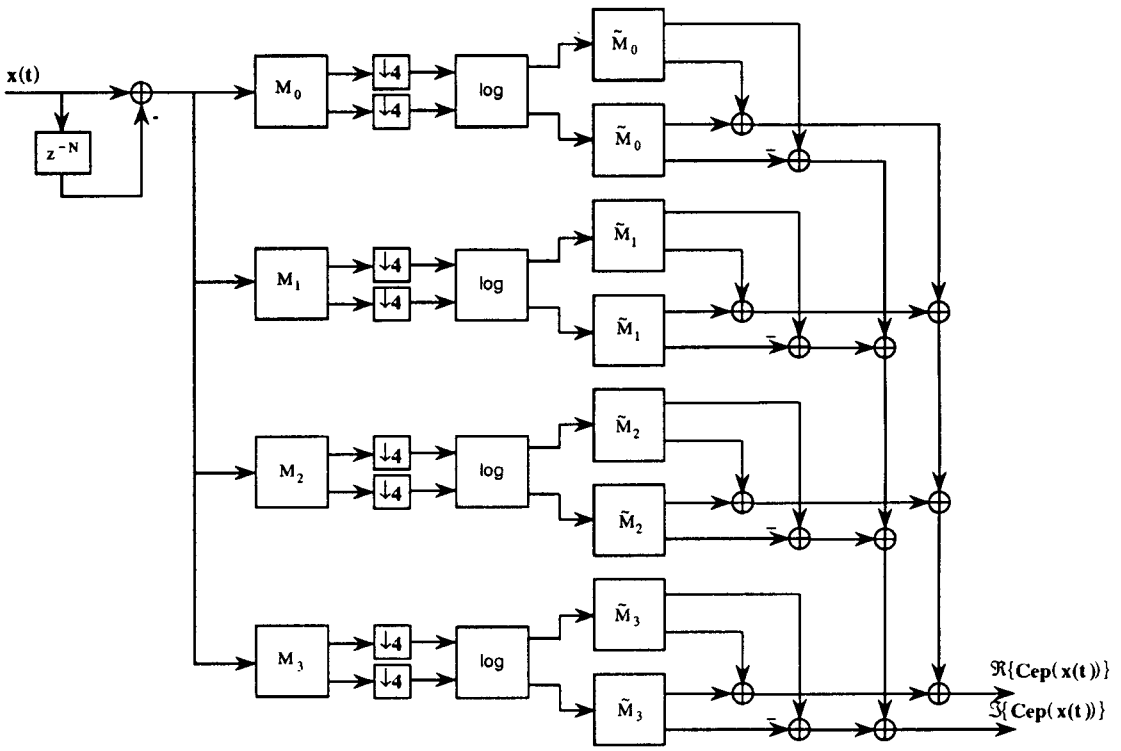


Figure 3.13: Cepstral transform architecture based on slow logarithm circuit.

3.4 Example C: Discrete Wavelet Transform

The mapping operators considered in the first two examples, the DCT and the DFT, were sinusoidal functions. Since these functions belong in the class specified by Lemma 2.2, the time-recursive architecture was derived by following the procedure described in Subsection 2.2.3. In this Section, we consider the implementation of a pair of finite impulse response filters (FIR),

$$\mathbf{H} = [h_{N-1}\ h_{N-2}\ \cdots\ h_0] \quad \text{and} \quad \mathbf{G} = [g_{N-1}\ g_{N-2}\ \cdots\ g_0],$$

that are used in the implementation of the Discrete Wavelet Transform (DWT) [35, 48, 50, 28]. We use the procedure described in Subsection 2.2.5 in order to express the above coefficient vectors in terms of the kernel functions specified in

Lemma 2.2. The resulted linear expression is used as the basis for the design of a time-recursive lattice architecture.

3.4.1 Architecture for the DWT/IDWT

Implementing the filters **H** and **G** is equivalent to implementing the mapping operators

$$[h_0\ h_1\ \cdots\ h_{N-1}],\ \text{ and }\ [g_0\ g_1\ \cdots\ g_{N-1}].$$

Since the input specification to our Generic Design Procedure is in coefficient vector form, we follow the left branch of the flow diagram in Fig. 3.1.

Step 0.1: Consider the linear system with the N first Markov coefficients being equal to the N columns of the matrix

$$\begin{bmatrix} h_0 & h_1 & \cdots & h_{N-1} \\ g_0 & g_1 & \cdots & g_{N-1} \end{bmatrix}.$$

We specify the partial realization $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ of minimal order M for this linear system [27, 29], so that

$$\begin{bmatrix} h_n \\ g_n \end{bmatrix} = \mathbf{c}\mathbf{A}^n\mathbf{b}, \quad n = 0, 1, \cdots, N-1. \tag{3.4.1}$$

Step 0.2: Bring the triplet $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ in the modal canonical form [27]. If our system is a normalized paraunitary system [54] the magnitude of all the eigenvalues of the system matrix **A** will be equal to 1. For the sake of concreteness, suppose that the order of the system is $M = 3$. The format of the matrix **A** will be as follows:

$$\mathbf{A} = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & \beta \end{bmatrix},$$

where α takes values in the interval $[0, 2\pi)$ and β equals either 1 or -1. The $M \times 1$ vector **b** and the $2 \times M$ matrix **c** do not have any particular structure.

Step 0.3: By substituting the above expression of $\mathbf{A}$ in (3.4.1) and expanding the matrix notation we obtain

$$\begin{bmatrix} h_n \\ g_n \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \end{bmatrix} \begin{bmatrix} \cos\alpha n & \sin\alpha n & 0 \\ -\sin\alpha n & \cos\alpha n & 0 \\ 0 & 0 & \beta^n \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

and consequently

$$\begin{bmatrix} h_n \\ g_n \end{bmatrix} = \begin{bmatrix} c_{00}b_0 + c_{01}b_1 \\ c_{10}b_0 + c_{11}b_1 \end{bmatrix} \cos\alpha n + \begin{bmatrix} -c_{01}b_0 + c_{00}b_1 \\ c_{11}b_0 + c_{10}b_1 \end{bmatrix} \sin\alpha n + \begin{bmatrix} c_{02}b_2 \\ c_{12}b_2 \end{bmatrix} \beta^n. \tag{3.4.2}$$

Step 1.1: The kernel groups we need to implement are

$$\mathbf{f}_0(n) = \begin{bmatrix} f_{00}(n) \\ f_{01}(n) \end{bmatrix} = \begin{bmatrix} \cos\alpha n \\ \sin\alpha n \end{bmatrix} \quad \text{and} \quad \mathbf{f}_1(n) = f_{10}(n) = \beta^n.$$

Step 1.2: For the kernel group $\mathbf{f}_0(\cdot)$ we have $\mathbf{f}_0(n-1) = \mathbf{R}\mathbf{f}_0(n)$ with

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}.$$

We also have

$$\begin{bmatrix} f_{00}(0) \\ f_{01}(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} f_{00}(N) \\ f_{01}(N) \end{bmatrix} = \begin{bmatrix} \cos\alpha N \\ \sin\alpha N \end{bmatrix}.$$

The resulted architecture implies module M_0 in Fig. 3.14. As we have seen in Chapter 2 (Lemma 2.7), the periodicity property is satisfied if $\alpha = \frac{k\pi}{N}$ with k being an integer, which is not true in general. On the other hand, for the singleton kernel group $\mathbf{f}_1(\cdot)$ we have $\mathbf{f}_1(n-1) = \mathbf{R}\mathbf{f}_1(n)$ with $\mathbf{R} = \frac{1}{\beta}$ and also $f_{10}(0) = 1$, $f_{10}(N) = \beta^N$. The associated architecture is demonstrated by module M_1 in Fig. 3.14. No multipliers are needed for the implementation of $\mathbf{f}_1(\cdot)$ if the eigenvalues have unit magnitude. The architectural implementation

of the given pair of mapping operators for the case where $M = 3$ is shown in Fig. 3.14.

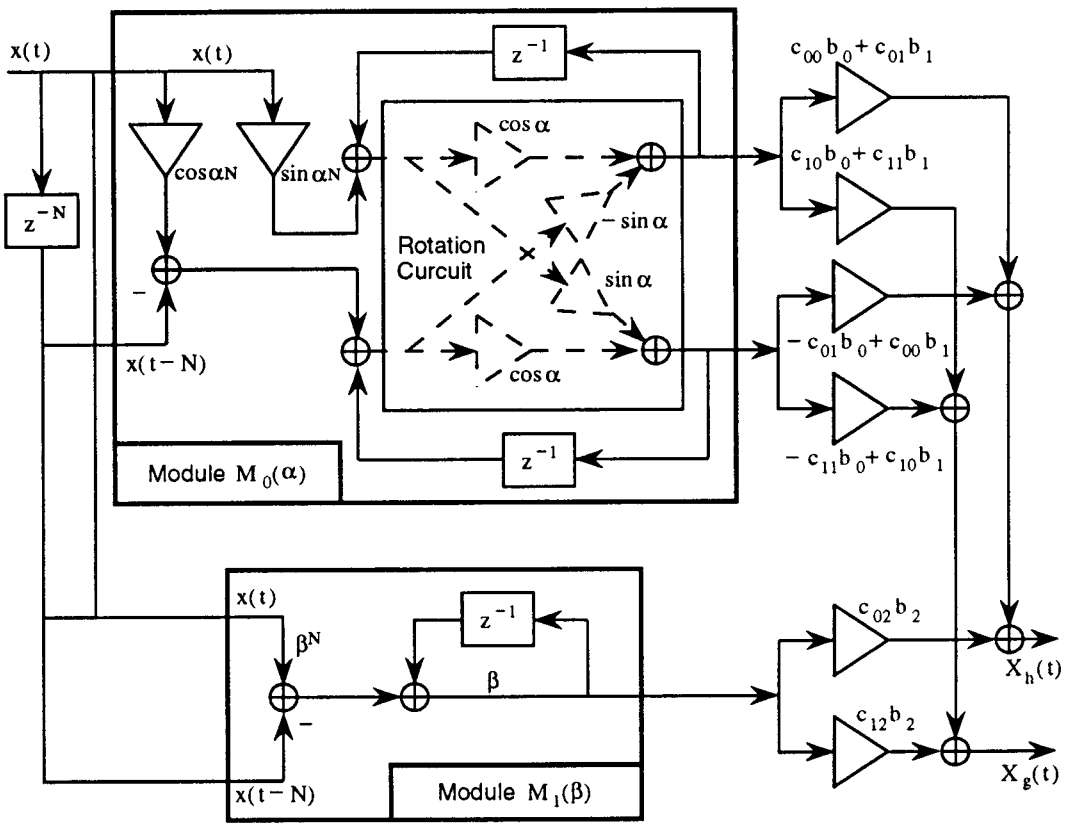


Figure 3.14: The architectural modules used for DWT.

For the general case of a system of an arbitrary order M with distinct, unit magnitude eigenvalues we need to implement M kernel functions. Among these functions no more than two are in the form of $\mathbf{f}_1(\cdot)$ seen above (since only two distinct such functions exist with $\beta = 1$ and $\beta = -1$) and they are implemented by module M_1. The rest of the kernel functions will group into pairs of cosine-sine functions specified by the parameter α, as dictated by $\mathbf{f}_0(\cdot)$ in the above example, and they can be implemented by module M_0.

In the sequel, we consider the implementation of the pair of wavelet filters **H** and **G**, whose coefficients, obtained in [2], are given on Table 3.3. The lengths of the filters **H** and **G** are 9 and 7 respectively. The size of the kernel group we have to implement (that is the order of the associated linear system) is $M = 6$.

The architecture involves two copies of module M_0 and two copies of module M_1. The values of the parameters α and β, as well as the output weights are given on Table 3.3. The resulted architecture is shown on Fig. 3.15.

h_n	g_n	α or β	$weight_h$	$weight_g$
0.0267	0.0000	1.0	0.1781	-0.0032
-0.0168	0.0456	-1.0	-0.0056	0.1778
-0.0782	-0.0287	1.1085	-0.0881	-0.0235
0.2668	-0.2956		-0.3089	-0.0781
0.6029	0.5575	2.0929	-0.0575	-0.1511
0.2668	-0.2936		0.0998	0.2673
-0.0782	-0.0287			
-0.0168	0.0456			
0.0267	0.0000			

Table 3.3: Example of wavelet filter coefficients and the associated architecture parameters.

For the inverse Discrete Wavelet Transform (IDWT) we have to implement the mirror filters $\widetilde{\mathbf{H}}$ and $\widetilde{\mathbf{G}}$ of $\mathbf{G}$ and $\mathbf{H}$ respectively [2]. The architectural implementation of the IDWT is obtained from the corresponding DWT by replacing the parameters α and β by $\pi - \alpha$ and $-\beta$ respectively.

3.4.2 Cost Issues

The implementation of module M_0 in Fig. 3.14 requires 2 multipliers, 3 adders and one rotation circuit. For the implementation of module M_1 we need 2 adders. We implement the desired pair of mapping operators as two weighted sums of the outputs of the above described parts. If the size of the associated kernel group is M the cost of this interconnection is $2M$ multipliers and $2(M-1)$ adders. The overall cost of the design is not higher from $3M$ multipliers, $\lfloor 7M/2 \rfloor$ adders and $M/2$ rotation circuits. For the example we consider in Fig. 3.15 the size of the kernel group is $M = 6$ and the implementation cost is 15 multipliers, 20 adders and 2 rotation circuits. Note that a rotation circuit can be implemented very efficiently with a CORDIC processor [23] or a distributed arithmetic design [52].

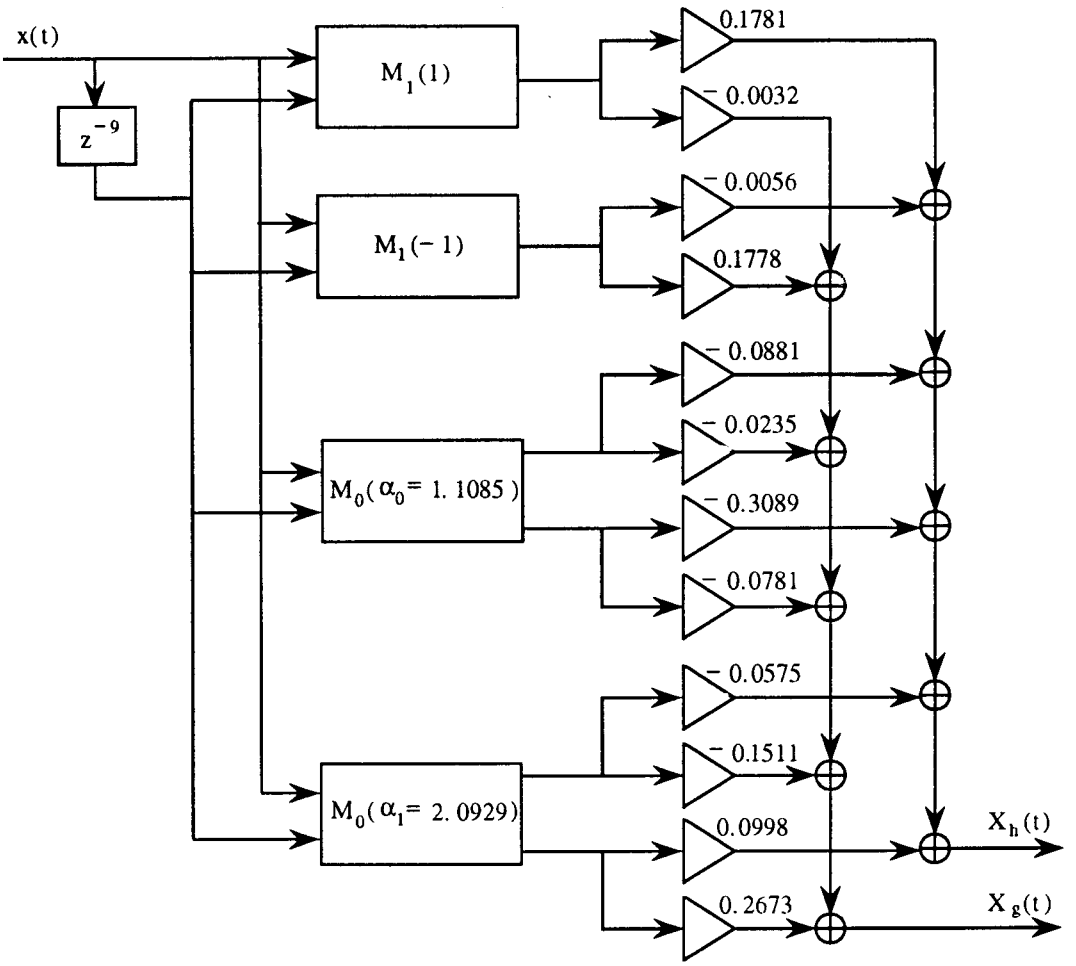


Figure 3.15: Architecture for the DWT filters specified in table 3.

Assume that N uniform subband divisions of a signal spectrum yield the same resolution quality with $\log_2 N$ logarithmicly spaced subband divisions. For example, suppose that the 16-point DFT yields the same resolution quality with the DWT of with 4 $\mathbf{H} - \mathbf{G}$ pairs [3][35]. The cost implied by using the architecture in Fig. 3.15 as the building block for the DWT tree structure is given on Table 3.1. Efficient FFT-based algorithms have been developed for the implementation of the DWT on a uniprocessor [48]. Nevertheless, these algorithms map on very expensive architectures. For concreteness, let us focus on the example we considered in the previous subsection. The DWT filters have

[3]Knowles has implemented this scheme with only one filter pair but he uses feedback very extensively [28].

lengths 9 and 7. For these lengths a 16-point FFT should be used, that yields $5.76 \log_2 N$ multiplications and $14.72 \log_2 N$ additions per data sample [48]. This algorithm maps on an architecture with $144 \log_2 N$ multipliers and $368 \log_2 N$ adders. The same numbers in multiplications and additions are needed for the sliding version of the DWT per data sample. Smaller but still prohibitive is the cost if the short length FIR filter technique is used instead [48]. On the other hand, both the architectural implementation and the sliding version of the DWT can be obtained by directly implementing the FIR filters $\mathbf{H}$ and $\mathbf{G}$. This straightforward implementation is the most efficient among the ones we have discussed in this Subsection for both of the above cases.

3.5 Conclusion

The Generic Design Procedure we propose is an efficient tool for exploiting the various design alternatives and obtaining time-recursive architectures. Research areas that can be benefited by this tool include real-time data compression, adaptive filtering and spectrum analysis.

The examples of time-recursive computation we have considered exhibit both the strengths and the weaknesses of the time-recursive computation. More precisely, for the DCT and the DFT the proposed architectures compete very well with different implementation schemes that appear in the literature. We conjecture that this is a consequence of the fact that our designs are composed by linear, time invariant (LTI) components. It is well known that the eigenfunctions (modes) of such LTI systems are exponential functions. As a result, the mapping operators that can be expanded in a sum of only a few exponentials (for example, 2 for the case of DCT and DFT and 4 for the case of MLT that is studied in Chapter 4) can be implemented efficiently with a time-recursive computation. Furthermore, the implementation cost is independent of the length of the mapping operator. On the other hand, if the expansion involves a "fair" number of exponential functions the resulted architecture will not be cost efficient. This is the very case of the DWT, where the time-frequency locality of

wavelets guarantees that the expansion of the DWT filters involves a fair amount of exponentials.

In conclusion, the time-recursive computation is appropriate for implementing narrowband FIR filters as well as a wide variety of data transforms, including the discrete sinusoidal transforms like the Discrete Fourier Transform, the Discrete Cosine Transform, the Discrete Sine Transform, the Discrete Hartley Transform and some Lapped Orthogonal Transforms.

CHAPTER
4

Application on QMF Banks and Data Transforms

The Quadrature Mirror Filter (QMF) banks play an important role in multirate system theory and design with an impact to numerous applications (see [54] and the references therein). The connection of the QMF banks with a class of data transforms, like the Short Time Fourier Transform (STFT), the Discrete Wavelet Transform (DWT) and the Lapped Orthogonal Transforms (LOT) has attracted a lot of interest lately, that gave rise to a number of novel transform basis. The latter yield performance characteristics similar to the ones of the QMF banks in addition to an easier design and cost efficient implementations [35, 39, 54].

In this Chapter, we propose the use of time-recursive computation as the substitute of fast algorithms used in the implementation of two kinds of filter banks: the **uniform-DFT filter bank** and the **cosine modulated filter bank**. In this way, the overall design involves local interconnections and therefore it becomes more appropriate for VLSI implementation, thus facilitating the single-chip implementation of the QMF analysis and the QMF synthesis system. We also consider the architectural implementation of some data transforms, namely a Short Time Fourier Transform, the Modulated Lapped Transform (MLT) [36] and an Extended Lapped Transforms (ELT) [38] that are special cases of the above filter banks. The designs we propose for all three transforms are modular, regular and they require local communication, thus they are very appropriate for VLSI implementation. In addition, they are very efficient in terms of area utilization because: first, they have linear requirements in operation counts,

second, a substantial part of the computation is carried out by rotation circuits that can be implemented very efficiently [52] and third, the designs are based on locally interconnected building modules that are almost the same in number and complexity with the corresponding ones of the DFT.

In Section 4.1, we provide some background information about QMF banks, we describe the uniform-DFT QMF bank and we introduce some modification to the existing scheme. In Section 4.2, after a short discussion on the cosine modulated QMF bank, we introduce a time-recursive structure for computing the modulation matrix. In Section 4.3, we introduce the time-recursive implementation of the STFT with Hanning windowing of the data. This design was first outlined in [32]. In Section 4.4, we derive the time-recursive architectures for MLT and inverse MLT. Finally, in Section 4.5, we derive the time-recursive architecture for an ELT. It worths noting that although the basis functions of this transform form an N-band QMF bank with filter length equal to $4N$, the implementation cost is only $3N+4$ multipliers, $4N+4$ adders and $N+2$ rotation circuits.

We focus on the design of the analysis systems (with the exception of the case of MLT). The synthesis counterparts can be derived in similar ways.

4.1 Uniform-DFT Filter Banks

4.1.1 Background

Consider the structure in Fig. 4.1.a, composed by a filter bank $H_0(z), H_1(z), \cdots, H_{N-1}(z)$ followed by decimators. Suppose that the filters have perfect bandpass responses with equal bandwidths and they collectively cover all the frequency range $[0, 2\pi]$. Then, the original signal $x(t)$ can be recovered from the subband signals $x_0(Nt), x_1(Nt), \cdots, x_{N-1}(Nt)$ by proper choice of the (perfect passband) filters $F_0(z), F_1(z), \cdots, F_{N-1}(z)$ in the synthesis system in Fig. 4.1.b. Nevertheless, if realizable filters are used the reconstructed signal $\hat{x}(t)$ may suffer from alias error, as well as magnitude and/or phase distortion. These analysis/synthesis structures are well studied in the literature under the name Quadra-

ture Mirror Filters (QMF) (see for example [54] and the references therein).

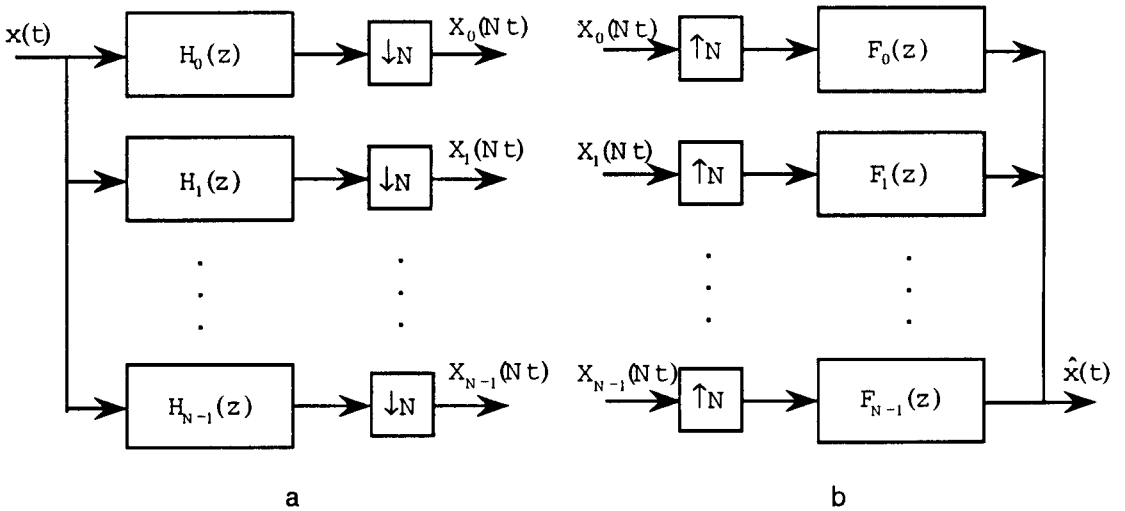


Figure 4.1: N-channel QMF bank: a. analysis system, b. synthesis system.

A set of N filters $H_k(z), k = 0, 1, \cdots, L-1$ forms a uniform-DFT filter bank if the filters are related as follows:

$$H_k(z) = H_0(e^{-j\frac{2\pi}{N}k}z). \tag{4.1.1}$$

In other words, the frequency responses of the filters are modulated versions of the response of a prototype filter $H_0(z)$, as shown in Fig. 4.2.

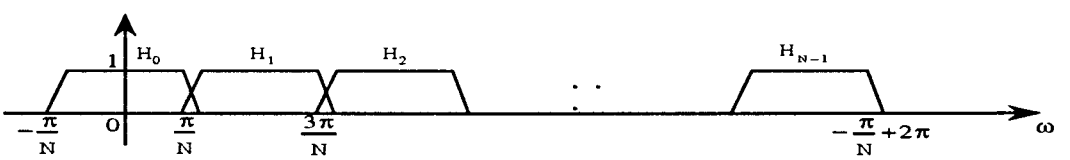


Figure 4.2: Typical frequency responses of uniform-DFT filters.

Consider the polyphase decomposition of the prototype filter [54]:

$$H_0(z)) = \sum_{l=0}^{+\infty} h(l)z^{-l} = \sum_{n=0}^{N-1} z^{-n}E_n(z^N), \tag{4.1.2}$$

where

$$E_n(z)) = \sum_{l=0}^{+\infty} e_n(l) z^{-l} \quad \text{and} \quad e_n(l) = h(Nl+n), \quad 0 \le n \le N-1.$$

For instance, if $N = 2$ we have the following polyphase representation:

$$H_0(z) = E_0(z^2) + z^{-1} E_1(z^2), \quad \text{where} \quad e_0(l) = h(2l), e_1(l) = h(2l+1), \quad l = 0, 1, \cdots.$$

Since $\left(e^{-j\frac{2\pi}{N}k}\right)^N = 1$, from (4.1.1) and (4.1.2) we obtain:

$$H_k(z) = \sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}kn} z^{-n} E_l(z^N), \quad k = 0, 1, \cdots, N-1. \tag{4.1.3}$$

Therefore, the N filters can be implemented by using the structure shown in Fig. 4.3, where we have used the noble identity specified in Fig. 4.4 [54]. The IDFT block denotes the $N \times N$ inverse discrete Fourier transform.

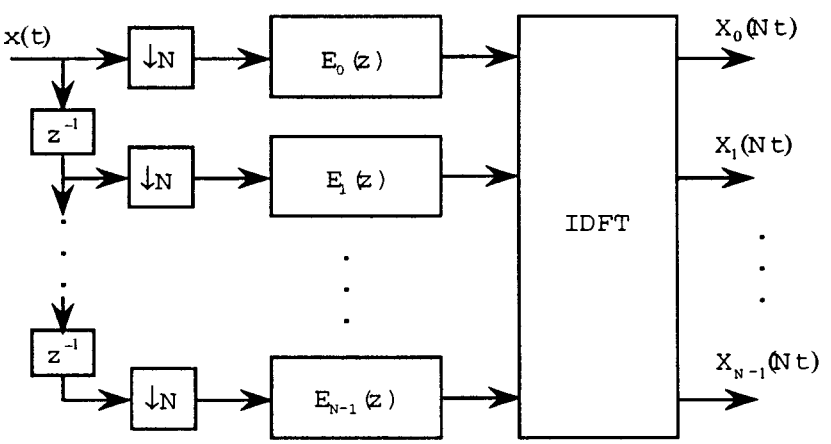


Figure 4.3: The uniform-DFT bank using polyphase decomposition.

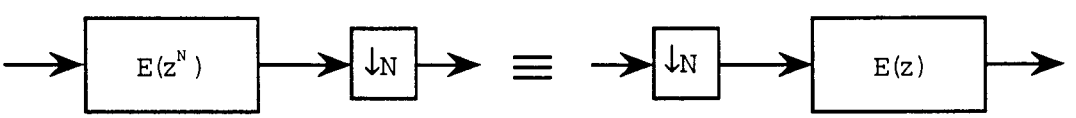


Figure 4.4: The noble identity.

There is a number of advantages in this implementation [54]:

1. For the filter design process, only the prototype filter $H_0(z)$ needs to be optimized. This filter will be lowpass with bandedge at $w = \frac{\pi}{L}$.

2. If $H_0(z)$ is FIR with length L the implementation cost will be L multipliers, $L - N$ adders plus the cost of the IDFT. The total cost is much lower than the implementation cost of NL multipliers and $N(L-1)$ adders that are required by the corresponding traditional designs. Furthermore, if N is a power of 2, then the FFT can be used to implement the IDFT in an efficient manner.

3. The polyphase components operate at minimum rate, that is the input data rate, as dictated by the transfer functions $E_0(z), E_1(z), \cdots, E_{N-1}(z)$.

4.1.2 Time-Recursive IDFT in the Uniform-DFT Filter Bank

The IDFT module in the decimated uniform-DFT filter bank can be implemented in a time-recursive way, either by using a MISO computation (see Subsection 3.3.2) or a SIMO computation (see Subsection 3.3.3). The overall uniform-DFT filter bank structure utilizing the SIMO approach is depicted in Fig. 4.5 and it operates as follows: the incoming signal and its delayed versions are decimated and filtered by the polyphase components $E_0(z), E_1(z), \cdots, E_{N-1}(z)$ at minimum rate that is N times lower than the input data rate. The output of these components is loaded by the Shift Array (SA). Between two adjacent loads SA shifts its data upwards and feeds the IDFT modules as shown in Fig. 4.5. In this way, SA ensures that the data vector of length N which is produced by the polyphase components will be fed to all of the IDFT modules. The output of the latter is decimated by a factor of N.

Observe that the time-recursive implementation of the IDFT involves unit delay elements. Consequently, we can not use the noble identity (see Fig. 4.4) in order to obtain minimum operation rate for the polyphase components and the IDFT modules. This problem is alleviated by the use of decimators and switches at the input and the output of the polyphase components (see Fig. 4.5). In this way, the polyphase components operate at minimum rate, that is N times lower

from the operation rate of SA and the IDFT modules (i.e. the input data rate). We have seen that the IDFT modules can be implemented by a lattice structure. This consists of a rotation circuit that can be implemented very efficiently by using distributed arithmetic [60, 52]. Consequently, the increased rate requirement for the IDFT modules can be efficiently accommodated.

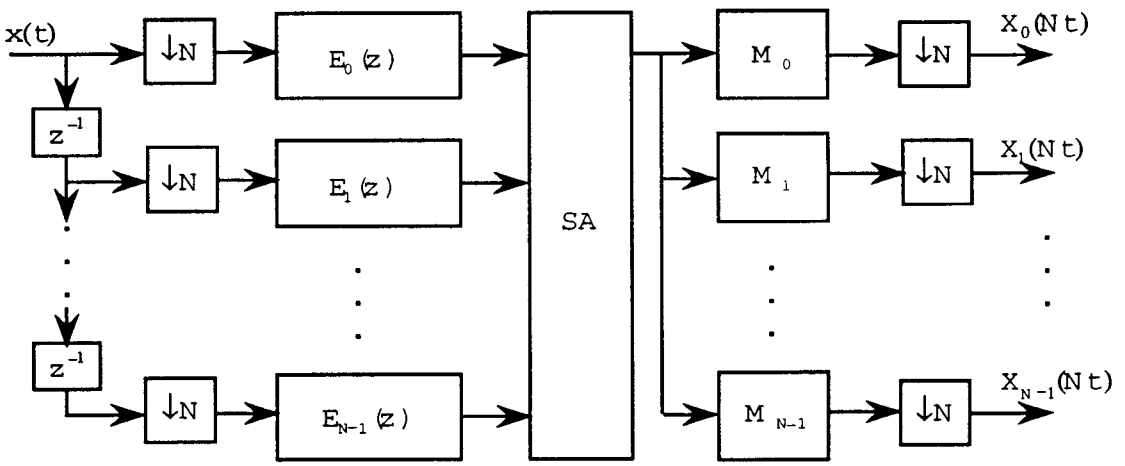


Figure 4.5: Decimated uniform-DFT bank using polyphase decomposition and time-recursive implementation for the IDFT.

In conclusion, the advantages in implementing the uniform-DFT filter bank with the above described structure are:

1. Locality is maintained in the filter bank, that is very important in VLSI implementations, since bit-parallel operations and a fast circuit clock can be used.

2. Filter banks with large number of filters N can be implemented in VLSI. This is a consequence of the fact that the design is scalable and it involves linear implementation cost.

3. The design is efficient for arbitrary values of the filter bank size N, unlike the FFT based designs that are more efficient for N being a power of 2.

4.2 Cosine Modulated Filter Banks

4.2.1 Background

The **pseudo QMF banks** are designed to achieve alias cancellation, approximate linear phase and minimum magnitude distortion of the reconstructed signal [45]. The filters $H_0(z), H_1(z), \cdots, H_{N-1}(z)$ are modulated versions of a prototype filter $P_0(z)$. A cosine matrix is used for performing the modulation operation. More specifically, the analysis filters are:

$$H_k(z) = \sum_{n=0}^{2N-1} c_{kn} z^{-n} E_n(-z^{2N}), \quad k = 0, 1, \cdots, N-1,$$

where $E_n(z), n = 0, 1, \cdots, 2N-1$ are the $2N$ polyphase components of a prototype filter $P_0(z)$ and the coefficients c_{kn} are

$$c_{kn} = 2\cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n-\frac{L-1}{2}\right)+\theta_k\right], \tag{4.2.1}$$

where $k = 0, 1, \cdots, N-1$, $n = 0, 1, \cdots, 2N-1$ and $\theta_k = (-1)^k \frac{\pi}{4}$. The resulting QMF analysis structure is depicted in Fig. 4.6. This enjoys all the advantages of the uniform-DFT filter bank (cf. Subsection 4.1.1); also, the modulation matrix has real coefficients. Furthermore, if the length L of the prototype filter $P_0(z)$ is an even multiple of the filter bank size N, then the modulation matrix can be implemented based on fast Discrete Cosine Transform algorithms [54].

Perfect Reconstruction (PR) can be achieved by a cosine modulated filter bank if on the already described cosine modulated filter bank we impose two additional constraints: first, the prototype filter $P_0(z)$ needs to be symmetric and second, the pairs of the polyphase components $E_k(z), E_{k+N}(z), k = 0, 1, \cdots, N-1$ need to be pairwise complimentary [1] [54]. The structure for the cosine modulated PR analysis filter bank is shown in Fig. 4.7.

[1] Two filters $H_0(z)$ and $H_1(z)$ are pairwise complementary if they satisfy the constraint

$$\left|H_0(e^{jw})\right|^2 + \left|H_1(e^{jw})\right|^2 = 1, \quad w \in [-\pi, \pi].$$

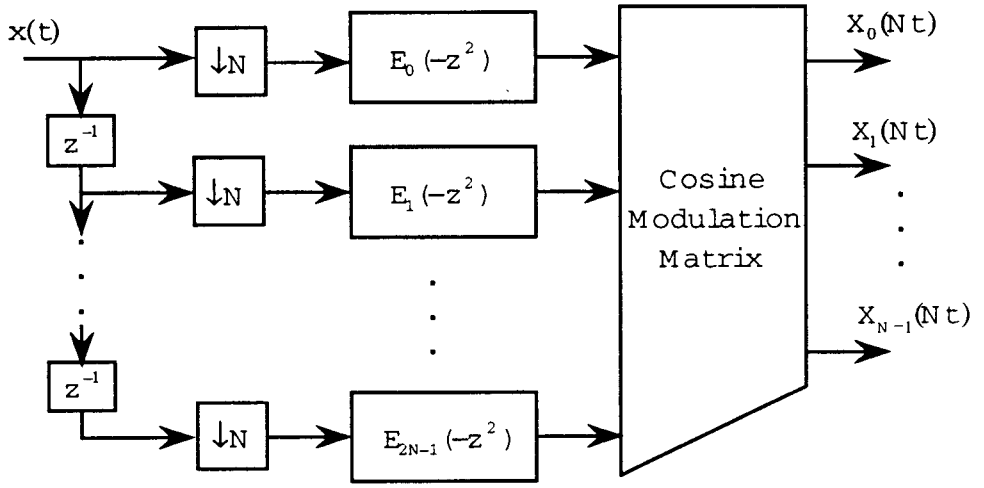


Figure 4.6: QMF analysis bank based on cosine modulation.

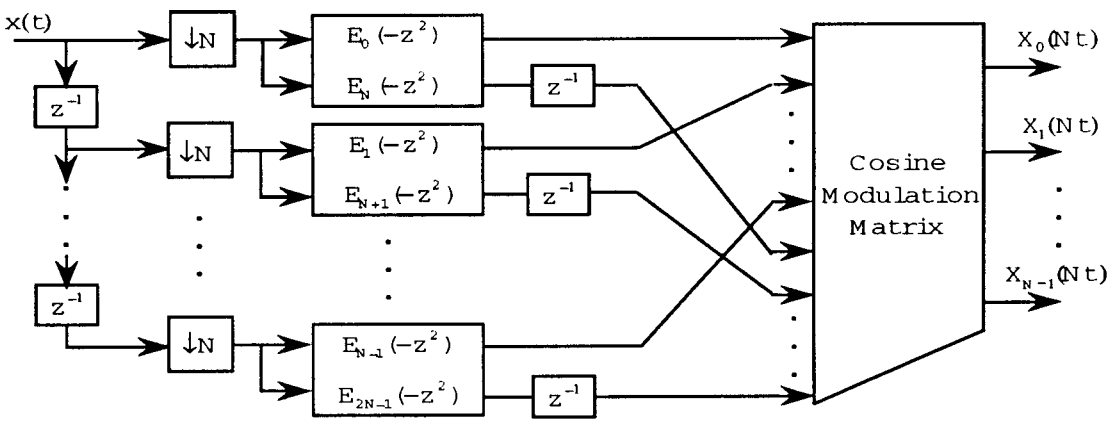


Figure 4.7: PR analysis bank based on cosine modulation.

4.2.2 Time-Recursive Computation of the Cosine Modulation Matrix

For the cosine modulation matrix specified by (4.2.1) we need to implement N mapping operators $\mathbf{c}_k$ of length $2N$

$$\mathbf{c}_k = [c_{k0}\ c_{k1}\ \cdots\ c_{k,2N-1}], \qquad k = 0, 1, \cdots, N-1.$$

The kernel group $\mathbf{f}_k(n)$ needed for the time-recursive evaluation of the k^{th} mapping operator is

$$\mathbf{f}_k(n) = \begin{bmatrix} f_{k,0}(n) \\ f_{k,1}(n) \end{bmatrix} = \begin{bmatrix} \cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n-\frac{L-1}{2}\right)+\theta_k\right] \\ \sin\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n-\frac{L-1}{2}\right)+\theta_k\right] \end{bmatrix}, \qquad \theta_k = (-1)^k\frac{\pi}{4}.$$

For this kernel group we have

$$\mathbf{f}_k(n-1) = \mathbf{R}\mathbf{f}_k(n), \quad \text{where} \quad \mathbf{R} = \begin{bmatrix} \cos\frac{\pi}{N}\left(k+\frac{1}{2}\right) & \sin\frac{\pi}{N}\left(k+\frac{1}{2}\right) \\ -\sin\frac{\pi}{N}\left(k+\frac{1}{2}\right) & \cos\frac{\pi}{N}\left(k+\frac{1}{2}\right) \end{bmatrix}$$

and

$$\mathbf{f}_k(0) = -\mathbf{f}_k(2N) = \begin{bmatrix} \cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\frac{L-1}{2}-\theta_k\right] \\ -\sin\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\frac{L-1}{2}-\theta_k\right] \end{bmatrix}.$$

Consequently, $\mathbf{f}_k(n)$ satisfies the periodicity property as this is defined in Chapter 2. Also, since only one of the two kernel functions in $\mathbf{f}_k(\cdot)$ is sufficient for our computation (that is $f_{k,0}(\cdot)$), the flow graph in Fig. 3.1 dictates that the IIR architecture should be adopted. This is shown in Fig. 2.8. Based on (2.3.7) we obtain the following expressions for the parameters of this design:

$$\begin{aligned} d_1 &= 2\cos\left[\tfrac{\pi}{N}\left(k+\tfrac{1}{2}\right)\right] & n_{00} &= -\cos\left[\tfrac{\pi}{N}\left(k+\tfrac{1}{2}\right)\tfrac{L-3}{2}-\theta_k\right] \\ d_2 &= 1 & n_{01} &= \cos\left[\tfrac{\pi}{N}\left(k+\tfrac{1}{2}\right)\tfrac{L-1}{2}-\theta_k\right] \\ & & n_{02} &= -\sin\left[\tfrac{\pi}{N}\left(k+\tfrac{1}{2}\right)\tfrac{L-3}{2}-\theta_k\right] \\ & & n_{03} &= \sin\left[\tfrac{\pi}{N}\left(k+\tfrac{1}{2}\right)\tfrac{L-1}{2}-\theta_k\right] \end{aligned}.$$

4.2.3 Modified Implementation of the Cosine Modulated Filter Bank

The cosine modulation matrix that appears in Fig. 4.6 and Fig. 4.7 can be implemented with a bank of isolated modules described in Subsection 4.2.2. The overall structure for the case of the PR cosine modulated filter bank is depicted in Fig. 4.8. This structure operates very similarly to the one in Fig. 4.5. The incoming signal and its delayed versions are decimated and filtered by the

$2N$ polyphase components $E_0(-z^2), E_1(-z^2), \cdots, E_{2N-1}(-z^2)$ that operate at minimum rate. The output of these components is loaded by the Shift Array (SA) that operates at the input data rate, that is N times faster than the polyphase components. At each time instant the 0^{th} element of SA is fed to the N modules $M_0, M_1, \cdots, M_{N-1}$ that implement the $2N \times N$ cosine modulation matrix. Simultaneously, at each time instant the contents of SA are shifted upwards one position.

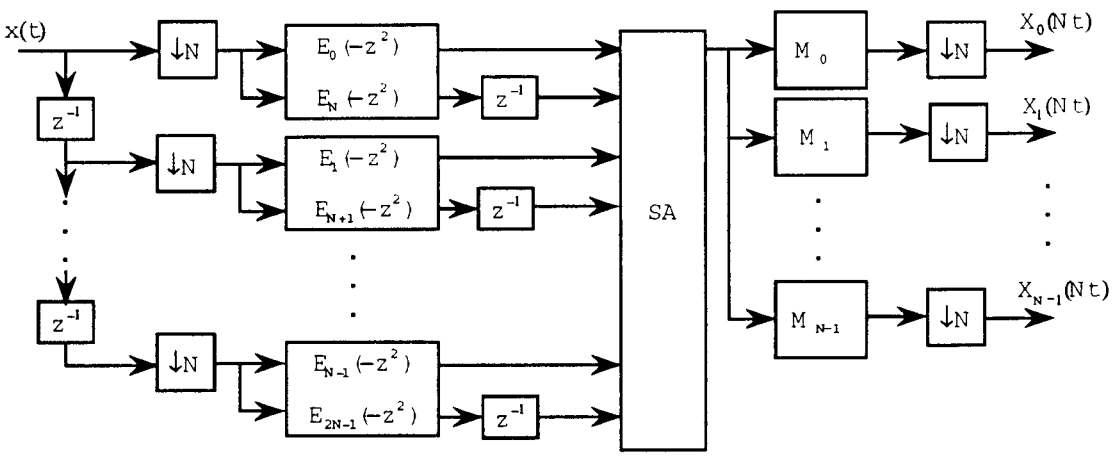


Figure 4.8: PR QMF analysis bank using time-recursive approach for implementing the cosine modulation matrix.

Note that for the pseudo QMF bank, the polyphase components are placed in an increasing order (see Fig. 4.6). Consequently, SA can be implemented with shift registers of length $2N$. On the other hand, for the PR QMF the polyphase components are placed according to the sequence $0, N, 1, N+1, \cdots, N-1, 2N-1$. In this case, SA consists of two shift arrays of length N interconnected as shown in Fig. 4.9.

The structure in Fig. 4.8 inherits all the advantages and the subtle points about the nonuniform operation rate of the structure in Fig. 4.5. In addition, the polyphase components in the PR QMF design in Fig. 4.8 can be realized as cascades of interchanging unit delay elements and rotation circuits [55, 54]. The latter can be implemented in an area efficient manner with CORDIC processors [23]. On the other hand, the IIR modules used for the realization of the

cosine modulation matrix, requires the implementation of five multipliers. Only one of them needs to operate at the increased rate and it can be implemented by fast distributed arithmetic as ROM lookup table [60]. In conclusion, we anticipate that an efficient single-chip implementation of the cosine modulated PR QMF is feasible based on the architecture described above.

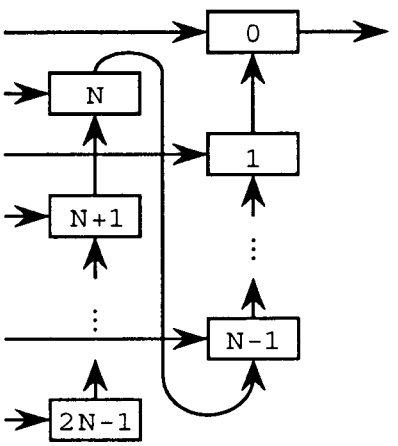


Figure 4.9: The Shift Array (SA) used in the PR QMF structure.

4.3 Short Time Fourier Transform with Hanning Window

In this Section we consider the implementation of the Short Time Fourier Transform with Hanning window. This is a special case of the uniform-DFT QMF bank [2], where the prototype filter $H_0(z)$ is the length-N Hanning window. In this case, the the polyphase components in Fig. 4.5 are scalar constants (length-1 FIR filters), equal to the window coefficients.

The use of the window originates from a weakness of the Discrete Fourier Transform (DFT). Although the DFT has been used extensively in spectrum analysis and adaptive filtering, its ability to resolve a weak signal in a presence of a stronger one is limited. This constraint stems from the fact that the rectangular window inherent in the DFT has the disadvantage that the peak sidelobe of the associated spectrum is down only 13dB from the mainlobe level, as shown

[2]More accurately, the IDFT matrix in Fig. 4.5 should be replaced by the DFT matrix, that amounts to a number of sign inversions as pointed out in Subsection 3.3.3.

in Fig. 4.10.a. The problem is alleviated by using non-rectangular windowing of the data, yielding the Short Time Fourier Transform (STFT) [32].

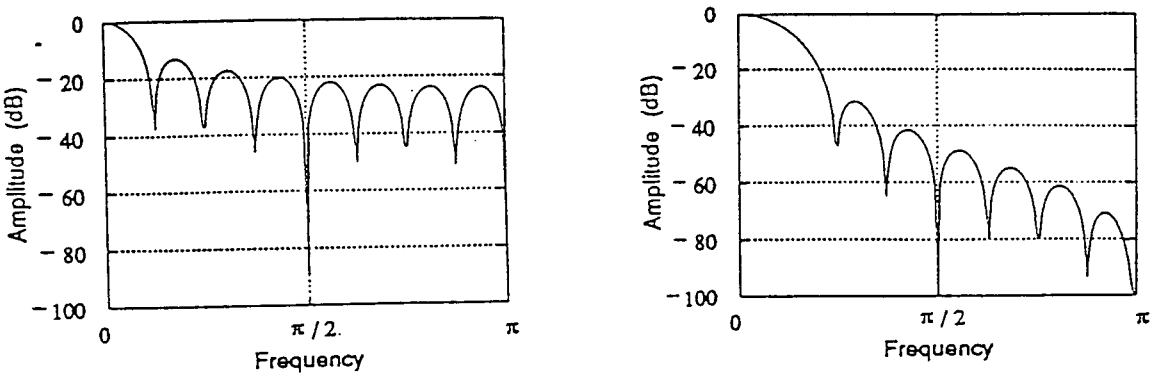


Figure 4.10: The spectrum of a. the rectangular window and b. the Hanning window of length $N = 16$.

A complete survey of the window functions and their properties can be found in [20]. The windowing is traditionally implemented as a modulation operation in the time domain. Nevertheless, it can also be implemented in the frequency domain by reversing the order of the polyphase bank with the DFT module, yielding a convolution operation. This approach gives efficient implementations for the class of sum-of-cosine windows [6, 40]. One of the most common window functions in spectrum analysis, that is also a sum-of-cosine type of window, is the raised cosine or Hanning window $w_N(\cdot)$

$$w_N(n) = \frac{1}{2}\left(1 - \cos\frac{2\pi}{N}n\right).$$

The latter has an 18dB per octave rolloff rate and a peak sidelobe level of -31dB (see Fig. 4.10.b). In this Section, we design the time-recursive architectural implementation of the windowed STFT with the Hanning window.

4.3.1 Architecture for the STFT with Hanning window

The N-point windowed STFT of a semi-infinite sequence of (real, scalar) data $x(\cdot)$ consists of N semi-infinite complex sequences $X_{STFT}(k,\cdot), k = 0, 1, \cdots, N-$

1 defined as follows

$$X_{STFT}(k,t) = \sqrt{\frac{1}{N}} \sum_{n=0}^{N-1} w_N(n) e^{-j\frac{2\pi}{N}kn} x(t+n-N+1), \quad t = 0, 1, \cdots$$

where $w_N(\cdot)$ denotes the window function. For the choice of the Hanning window we have

$$\begin{aligned}
\mathcal{R}\{X_{STFT}(k,t)\} &= \sqrt{\tfrac{1}{N}} \textstyle\sum_{n=0}^{N-1} \tfrac{1}{2}\left(1 - \cos\tfrac{2\pi}{N}n\right) \cos\tfrac{2\pi}{N}kn \; x(t+n-N+1) \\
\mathcal{I}\{X_{STFT}(k,t)\} &= -\sqrt{\tfrac{1}{N}} \textstyle\sum_{n=0}^{N-1} \tfrac{1}{2}\left(1 - \cos\tfrac{2\pi}{N}n\right) \sin\tfrac{2\pi}{N}kn \; x(t+n-N+1),
\end{aligned}$$

where $\mathcal{R}\{\cdot\}$ and $\mathcal{I}\{\cdot\}$ denote the real and the imaginary part of the bracketed quantity. Consequently, we seek the implementation of the following two mapping operators

$$\begin{aligned}
h_{k,n} &= \tfrac{1}{2}\left(1 - \cos\tfrac{2\pi}{N}n\right) \cos\tfrac{2\pi}{N}kn \\
g_{k,n} &= -\tfrac{1}{2}\left(1 - \cos\tfrac{2\pi}{N}n\right) \sin\tfrac{2\pi}{N}kn,
\end{aligned}$$

$k = 0, 1, \cdots, N-1$. After some elaborations we obtain the equivalent expressions that follow:

$$\begin{aligned}
h_{k,n} &= -\tfrac{1}{4} f_{k-1,0}(n) + \tfrac{1}{2} f_{k,0}(n) - \tfrac{1}{4} f_{k+1,0}(n) \\
g_{k,n} &= -\tfrac{1}{4} f_{k-1,1}(n) + \tfrac{1}{2} f_{k,1}(n) - \tfrac{1}{4} f_{k+1,1}(n),
\end{aligned} \tag{4.3.1}$$

where

$$\begin{bmatrix} f_{k,0}(n) \\ f_{k,1}(n) \end{bmatrix} = \begin{bmatrix} \cos\frac{2\pi}{N}kn \\ \sin\frac{2\pi}{N}kn \end{bmatrix}$$

specifies the associated kernel group $\mathbf{f}_k(\cdot)$. The architectural implementation of this kernel group was discussed in Chapter 3 (see Fig. 3.7). The resulting architecture for the STFT is depicted in Fig. 4.11).

The IIR implementation for the building modules is also considered for the sake of completeness. By substituting the expressions (3.3.2) and (3.3.3) in (2.3.7) we obtain

$$\begin{aligned}
d_1 &= -2\cos\tfrac{2k\pi}{N} & n_{00} &= \cos\tfrac{2k\pi}{N} \\
d_2 &= 1 & n_{01} &= -1 \\
& & n_{10} &= \sin\tfrac{2k\pi}{N} \\
& & n_{11} &= 0.
\end{aligned}$$

The resulted IIR module is obtained by substituting these expressions in Fig. 2.6.

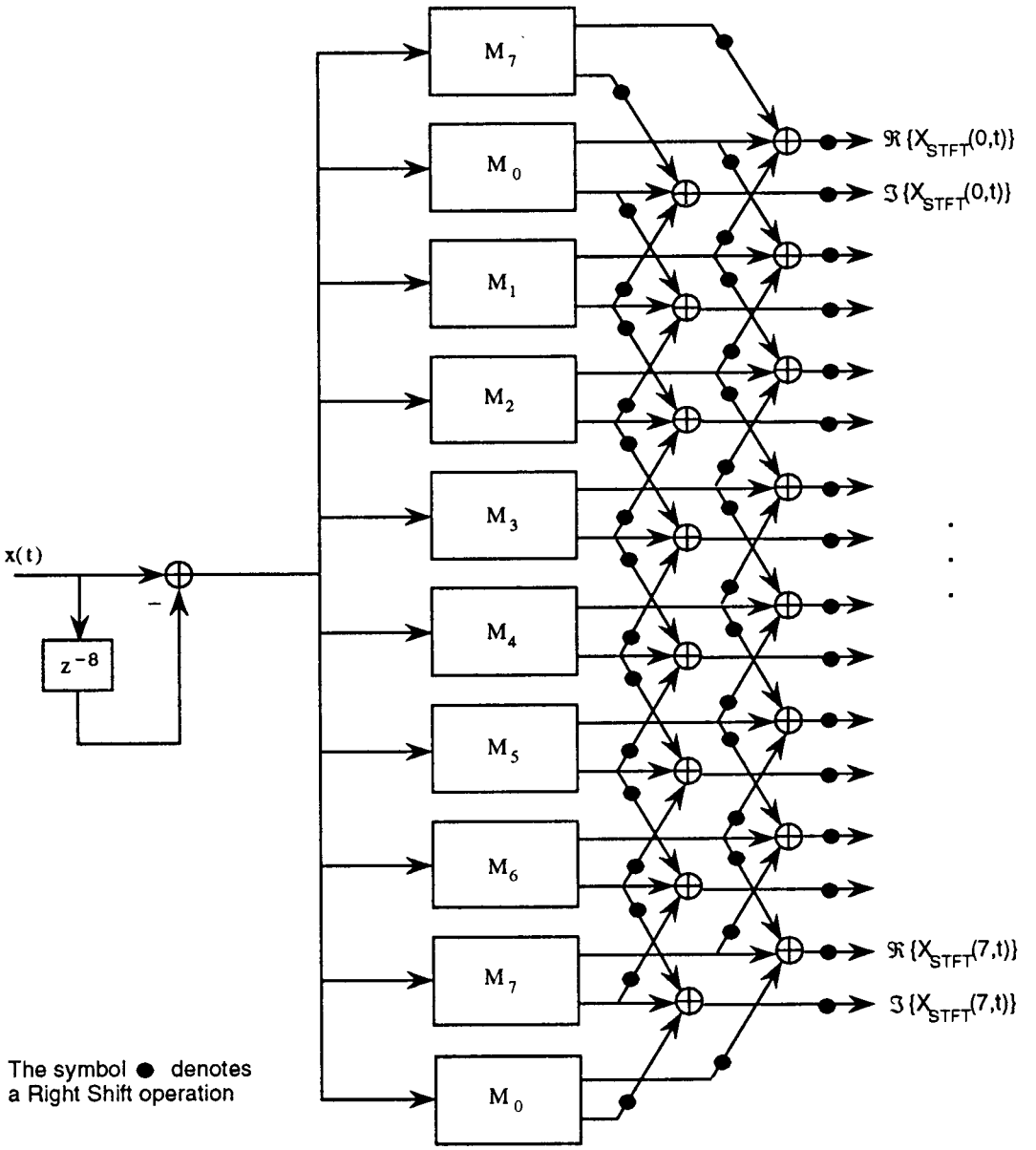


Figure 4.11: Time-recursive architecture of the STFT for $N = 8$.

4.3.2 Cost Issues

The overhead cost of the time-recursive implementation of the N-point STFT with respect to the cost of the DFT (see Table 3.1) is equal to $2N$ adders (and an equal number of shift registers). On the other hand, the corresponding overhead

for the FFT approach is equal to N multipliers. All the comments we have made in Chapter 3 regarding the throughput rate of the DCT and the DFT apply for the implementation of the STFT as well. The resulting cost metrics for the multiprocessor implementation of the STFT are depicted on Table 4.1, while the corresponding metrics for the uniprocessor implementation of the sliding version of the transform are given on Table 4.2.

		rate constraint	implementation cost (mult, add, rotation)
STFT	time-rec.,sliding	$A_P + R_P = u$	$0, 3N, N-1$
	time-rec.,block	$A_P + R_P = u$	$0, 3N-1, N-1$
	fast algo.,sliding	$M_S + A_S = u$	$\frac{N}{2}(\log_2 N - 1) + 2$,
	fast algo.,block	$M_S + A_S = Nu$	$\frac{N}{2}(3\log_2 N - 5) + 4, 0$
MLT	time-rec.,sliding	$M_P + A_P + R_P = u$	$2N+3, 3N+3, N-1$
	time-rec.,block	$M_P + A_P + R_P = u$	$2N+3, 3N+2, N-1$
	fast algo.,sliding	$M_S + A_S + R_S = u$	$\frac{N}{2}(\log_2 N + 1)$,
	fast algo.,block	$M_S + A_S + R_S = Nu$	$\frac{3N}{2}(\log_2 N + 1), \frac{N}{2}$
ELT	time-rec.,sliding	$M_P + A_P + R_P = u$	$3N+4, 4N+4, N+2$
	time-rec.,block	$M_P + A_P + R_P = u$	$3N+4, 4N+3, N+2$
	fast algo.,sliding	$M_S + A_S + 2R_S = u$	$\frac{N}{2}(\log_2 N + 1)$,
	fast algo.,block	$M_S + A_S + 2R_S = Nu$	$\frac{3N}{2}(\log_2 N + 2), N$

Table 4.1: Cost metrics for the architectural implementation of block transforms. M_P, A_P and R_P denote the time delays associated with a bit-parallel implementation of the multiplier, the adder and the rotation circuit respectively. M_S, A_S and R_S denote the corresponding time delays for a bit-serial implementation.

	time-recursive (mult,add)	FFT-like,sliding (mult,add)
STFT	$3N-3, 5N-2$	$\frac{N}{2}(\log_2 N - 1) + 2$, $\frac{N}{2}(3\log_2 N - 5) + 4$
MLT	$5N-3, 5N+3$	$\frac{N}{2}(\log_2 N + 5)$, $\frac{3N}{2}(\log_2 N + 1)$
ELT	$6N+10, 5N+7$	$\frac{N}{2}(\log_2 N + 8)$, $\frac{3N}{2}(\log_2 N + 2)$

Table 4.2: Cost metrics (multiplication and addition counts) for the uniprocessor implementation of sliding transforms.

Observe that the choice of the window has a direct impact on the communi-

cation requirements of the resulted architecture. The Hanning window implies communication links only among neighboring modules, while this is not true for an arbitrary window. There is a trade-off between the number of terms in the sum-of-cosine window and the locality of communication. On the other hand, for the uniprocessor time-recursive implementation of the sliding DFT the choice of the window affects the computational complexity. The discussion on windows in [6, 40] can be helpful for the choice of window in a particular application. In [40], some simple but efficient windows are presented, that yield no multipliers if implemented in frequency domain. In [6], the design of optimal sum-of-cosine windows in the least squares sense is reported.

4.4 Modulated Lapped Transform

In this Section, we consider the implementation of the Modulated Lapped Transform (MLT) [36, 39]. The basis functions of the N-point MLT constitutes an N-band PR cosine modulated QMF bank, for which the prototype filter is a sinusoidal function of length equal to $2N$ [39]. This implies that the polyphase components in Fig. 4.8 are length-2 FIR filters. In the following discussion, we present an alternative design that can be viewed as the result of changing the sequence of the polyphase component bank with the cosine modulation matrix, in a way similar to the one we have seen in Section 4.3.1 for implementing STFT.

The magnitude responses of the MLT basis functions are given in Fig. 4.12. The stopband attenuation is approximately $24dB$, that is considerably better from the $10dB$ worst case stopband attenuation of the DCT filter bank (cf. Fig. 3.2). As long as the transform domain adaptive filtering is concerned, the use of the MLT implies an increase of the rate of convergence in comparison with the rate associated to the use of DCT [39], while the same frequency domain characteristics corroborate for the use of MLT in subband coding for image and audio data [24, 25, 39, 49, 56]. Furthermore, being a Lapped Orthogonal Transform (LOT), MLT diminishes the blocking effect that appears at low bit rate data coding with transform techniques. The blocking effect is a natural

consequence of the independent processing of each block. Variations of the LOT have successfully been used in speech coding [36, 38, 39], image coding [37, 39] and motion estimation [62].

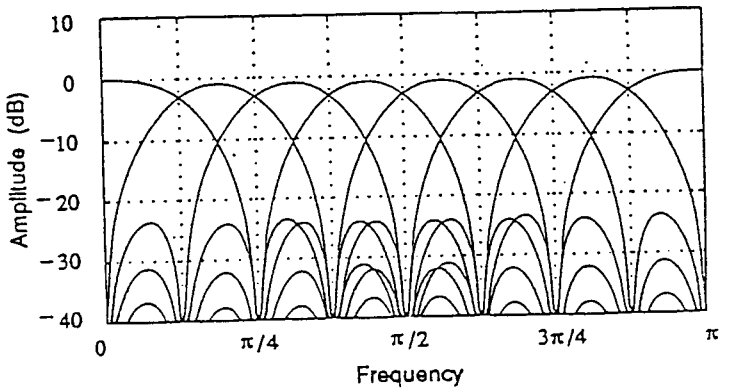


Figure 4.12: The magnitude responses of the MLT filter bank for $N = 8$.

The above facts highlight the importance of the proposed time-recursive architecture for both adaptive filtering purposes and data coding.

4.4.1 Architecture for the Forward MLT

The MLT operates on segments of data of length $2N$, $x(t+n-2N+1), n = 0, 1, \cdots, 2N-1$ and it produces N output coefficients $X_{MLT}(k,t), k = 0, 1, \cdots, N-1$ as follows [36]:

$$X_{MLT}(k,t) = c_k\sqrt{\tfrac{2}{N}}\sum_{n=0}^{2N-1}\sin\tfrac{\pi}{2N}\left(n+\tfrac{1}{2}\right)\cos\left[\tfrac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{N}{2}\right)\right] x(t+n-2N+1), \tag{4.4.1}$$

where $t = 0, 1, \cdots$ and $c_k = (-1)^{(k+2)/2}$ if k is even and $c_k = (-1)^{(k-1)/2}$ if k is odd. The sequence of the k^{th} output coefficients $X_{MLT}(k,t), t = 0, 1, \cdots$ can be thought of as the output of the mapping operator

$$h_{k,n} = \left(c_k\sqrt{\frac{2}{N}}\right)\sin\frac{\pi}{2N}\left(n+\frac{1}{2}\right)\cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right)\right]. \tag{4.4.2}$$

After a few algebraic manipulations, we derive the following decomposition of the mapping operator:

$$h_{k,n} = -\left(c_k\sqrt{\frac{1}{2N}}\right) f_{k+1,0}(n) - \left(c_k\sqrt{\frac{1}{2N}}\right) f_{k,1}(n), \quad k = 0, 1, \cdots, N-1, \quad (4.4.3)$$

where

$$\begin{bmatrix} f_{k,0}(n) \\ f_{k,1}(n) \end{bmatrix} = \begin{bmatrix} \cos\left[\frac{\pi}{N}k\left(n+\frac{1}{2}\right)+\left(k+\frac{1}{2}\right)\frac{\pi}{2}\right] \\ \sin\left[\frac{\pi}{N}k\left(n+\frac{1}{2}\right)+\left(k+\frac{1}{2}\right)\frac{\pi}{2}\right] \end{bmatrix} \triangleq \mathbf{f}_k(n)$$

is the associated kernel group. For this kernel group we have $\mathbf{f}_k(n-1) = \mathbf{R}_k\mathbf{f}_k(n)$, where

$$\mathbf{R}_k = \begin{bmatrix} \cos\frac{k\pi}{N} & \sin\frac{k\pi}{N} \\ -\sin\frac{k\pi}{N} & \cos\frac{k\pi}{N} \end{bmatrix}. \quad (4.4.4)$$

We also have

$$\begin{bmatrix} f_{k,0}(0) \\ f_{k,1}(0) \end{bmatrix} = S\begin{bmatrix} f_{k,0}(2N) \\ f_{k,1}(2N) \end{bmatrix} = \begin{bmatrix} \cos\left[\frac{k\pi}{2N}+\left(k+\frac{1}{2}\right)\frac{\pi}{2}\right] \\ \sin\left[\frac{k\pi}{2N}+\left(k+\frac{1}{2}\right)\frac{\pi}{2}\right] \end{bmatrix}, \quad (4.4.5)$$

where $S = 1$. Therefore, the periodicity property is satisfied. Since both member functions of the kernel group appear in the decomposition (4.4.3), the lattice architecture is recommended (cf. Fig. 3.1). In Fig. 4.13, we provide the lattice architecture module that is used as the building block in the MLT architectural implementation. The latter is depicted on Fig. 4.14 for the case of $N = 8$. Rotation circuits are used for the implementation of the lattice structure. The IIR implementation for building the MLT modules is also considered for the sake of completeness. By substituting the expressions (4.4.4) and (4.4.5) in (2.3.7) we obtain

$$\begin{aligned} d_1 &= -2\cos\tfrac{k\pi}{N} & n_{00} &= \cos\left[\left(k+\tfrac{1}{2}\right)\tfrac{\pi}{2} - \tfrac{k\pi}{2N}\right] \\ d_2 &= 1 & n_{01} &= -\cos\left[\left(k+\tfrac{1}{2}\right)\tfrac{\pi}{2} + \tfrac{k\pi}{2N}\right] \\ & & n_{10} &= \sin\left[\left(k+\tfrac{1}{2}\right)\tfrac{\pi}{2} - \tfrac{k\pi}{2N}\right] \\ & & n_{11} &= -\sin\left[\left(k+\tfrac{1}{2}\right)\tfrac{\pi}{2} + \tfrac{k\pi}{2N}\right]. \end{aligned}$$

The resulted IIR module is obtained by substituting the above expressions in Fig. 2.6.

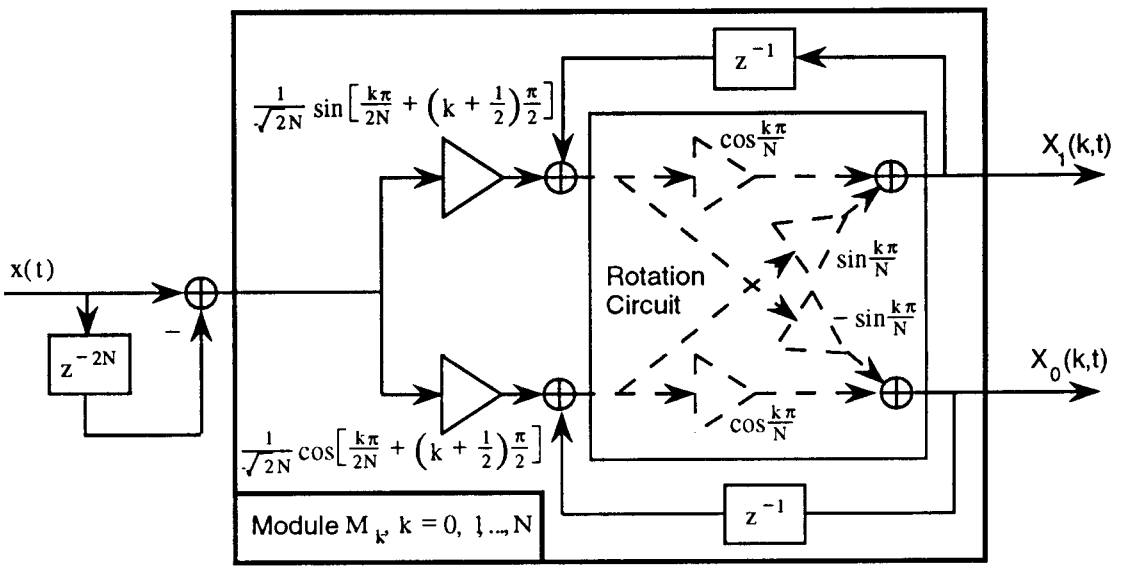


Figure 4.13: Lattice architecture for the MLT module.

4.4.2 Architecture for the Inverse MLT

The inverse MLT (IMLT) is specified by transposing the coefficient matrix of the direct transform. In closed form, the IMLT is

$$x(Nt+n+1) = \sqrt{\frac{2}{N}} \sin\frac{\pi}{2N}\left(n+\frac{1}{2}\right) \sum_{k=0}^{N-1} c_k \cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right)\right] X_{MLT}(k,Nt),$$

for $t = 1$ and

$$\begin{aligned} x(Nt+n+1) = {} & \sqrt{\tfrac{2}{N}} \sin\tfrac{\pi}{2N}\left(n+\tfrac{1}{2}\right) \textstyle\sum_{k=0}^{N-1} c_k \cos\left[\tfrac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{N}{2}\right)\right] X_{MLT}(k,Nt) + \\ & \sqrt{\tfrac{2}{N}} \sin\tfrac{\pi}{2N}\left(n+\tfrac{1}{2}+\tfrac{N}{2}\right) \\ & \textstyle\sum_{k=0}^{N-1} c_k \cos\left[\tfrac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{3N}{2}\right)\right] X_{MLT}(k,N(t-1)), \end{aligned} \tag{4.4.6}$$

for $t \leq 2$.

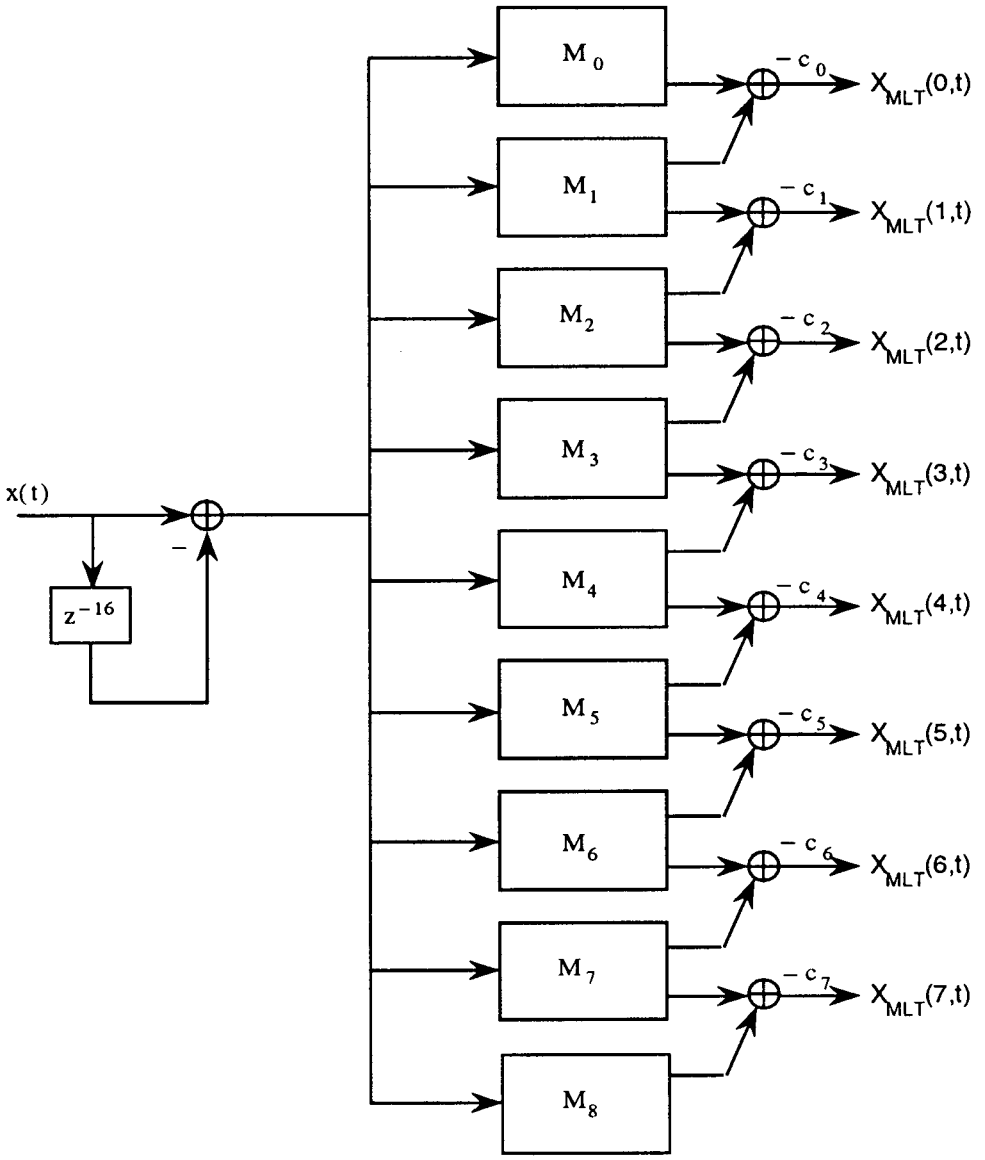


Figure 4.14: Time-recursive architecture for the MLT.

Since we have

$$\cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}+\frac{3N}{2}\right)\right] = (-1)^{k+1}\sin\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right)\right]$$

we obtain

$$x(Nt+n+1) =$$

$$\begin{cases} \sqrt{\frac{2}{N}} \sin \frac{\pi}{2N}\left(n+\frac{1}{2}\right) \sum_{k=0}^{N-1} c_k y_{k,0}(Nt+n+1), & t=1 \\ \sqrt{\frac{2}{N}} \sin \frac{\pi}{2N}\left(n+\frac{1}{2}\right) \sum_{k=0}^{N-1} c_k y_{k,0}(Nt+n+1)+ & \\ \sqrt{\frac{2}{N}} \sin \frac{\pi}{2N}\left(n+\frac{1}{2}+\frac{N}{2}\right) \sum_{k=0}^{N-1} (-1)^{k+1} c_k y_{k+1,1}(N(t-1)+n+1) & , \quad t>1, \end{cases}$$

where

$$y_{k,0}(Nt+n+1) = \cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right)\right] X_{MLT}(k,Nt)$$

and

$$y_{k,1}(Nt+n+1) = \sin\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right)\right] X_{MLT}(k,Nt).$$

Consider the kernel group

$$\mathbf{f}_k(n) = \begin{bmatrix} f_{k,0}(n) \\ f_{k,1}(n) \end{bmatrix} = \begin{bmatrix} \cos \frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right) \\ \sin \frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right) \end{bmatrix}. \tag{4.4.7}$$

For the kernel group $\mathbf{f}_k(\cdot)$ we have $\mathbf{f}_k(n-1) = \mathbf{R}_k \mathbf{f}_k(n)$, where

$$\mathbf{R} = \begin{bmatrix} \cos \frac{\pi}{N}\left(k+\frac{1}{2}\right) & \sin \frac{\pi}{N}\left(k+\frac{1}{2}\right) \\ -\sin \frac{\pi}{N}\left(k+\frac{1}{2}\right) & \cos \frac{\pi}{N}\left(k+\frac{1}{2}\right) \end{bmatrix} \quad \text{and} \tag{4.4.8}$$

$$\begin{bmatrix} f_{k,0}(0) \\ f_{k,1}(0) \end{bmatrix} = \begin{bmatrix} \cos \frac{(N+1)\pi}{2N}\left(k+\frac{1}{2}\right) \\ \sin \frac{(N+1)\pi}{2N}\left(k+\frac{1}{2}\right) \end{bmatrix}. \tag{4.4.9}$$

From (4.4.8) we get

$$\widetilde{\mathbf{R}} = \mathbf{R}^{-1} = \begin{bmatrix} \cos \frac{\pi}{N}\left(k+\frac{1}{2}\right) & -\sin \frac{\pi}{N}\left(k+\frac{1}{2}\right) \\ \sin \frac{\pi}{N}\left(k+\frac{1}{2}\right) & \cos \frac{\pi}{N}\left(k+\frac{1}{2}\right) \end{bmatrix}.$$

The periodicity property is not satisfied by the kernel group in (4.4.7). Consequently, the lattice architecture will be used as the building block for the architectural implementation of the IMLT (see Fig. 4.15). The architectural implementation of the IMLT is shown on Fig. 4.16.

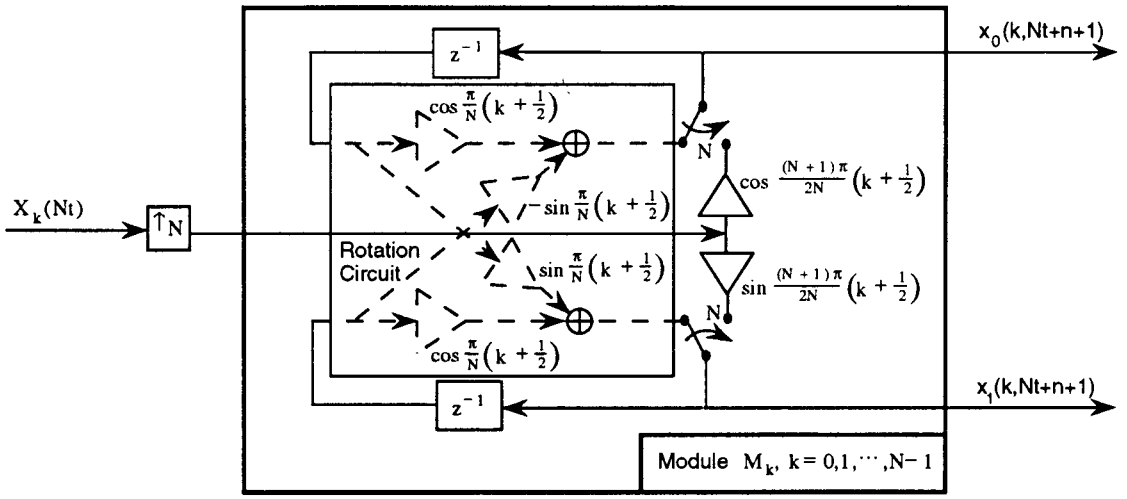


Figure 4.15: Lattice architecture for the IMLT module.

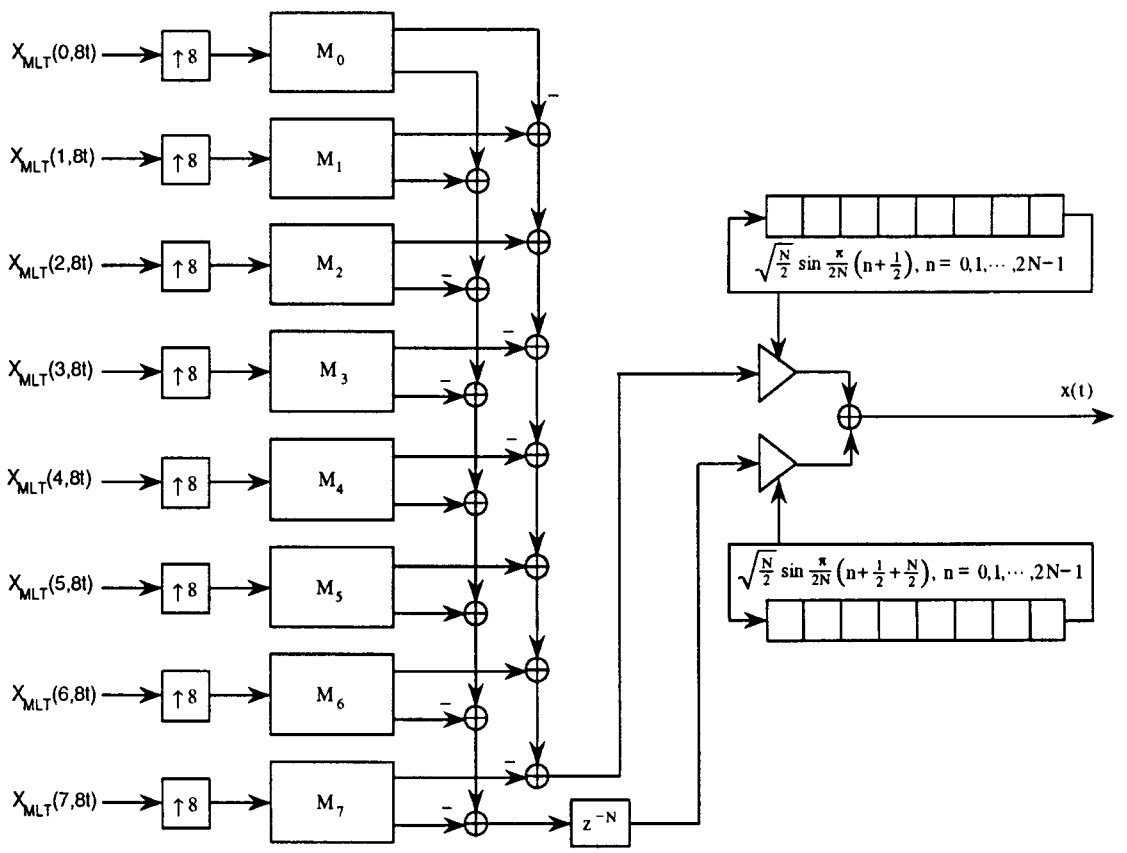


Figure 4.16: Time-recursive architecture for the IMLT.

4.4.3 Cost Issues

The cost of the time-recursive implementation of the MLT based on the lattice module (cf. Fig. 4.13)) is $N-1$ rotation circuits, $2N+3$ multipliers and $3N+3$ adders. The throughput of this architecture is given on Table 4.1. The implementation cost and the throughput expression for a fully pipelined implementation of the FFT-like algorithm in [36] are also given on Table 4.1. All remarks we have made in Chapter 3 for the throughput rate of the DCT apply for the implementation of the MLT as well. The cost expressions of the uniprocessor time-recursive implementation on Table 4.2 concern the IIR design.

It worths noting again the fact that the MLT constitutes a perfect reconstruction QMF bank that can be used in subband coding schemes. An $N-$band coding scheme may involve up to $6N-$tap FIR filters [26] and therefore it implies an implementation cost of the order of $6N^2$ (in multipliers and adders), while the use of the MLT in subband coding implies linear cost if the latter is implemented time-recursively (cf. Table 4.1).

4.5 Extended Lapped Transform with Basis Length = 4N

In Section 4.4, we discussed the importance of MLT in data coding and adaptive filtering that highlight the impact of the efficient VLSI architecture we propose. In fact, the implementation cost is comparable with the costs of the (already very efficient) time-recursive DCT (see Table 3.1 and 4.1). The performance characteristics of MLT can be further enhanced by using an Extended Lapped Transform (ELT) with longer kernel functions [38, 39], for example with length $4N$ or $6N$ instead of $2N$. Such basis functions can be viewed as PR cosine modulated QMF banks generated by a prototype filter of length $4N$ or $6N$ respectively, so that the corresponding FIR filters in the polyphase bank of Fig. 4.8 have lengths equal to 4 or 6. Apparently, a long prototype filter implies high implementation cost. However, prototype filters that generate the ELT transforms are sum-of-cosine type filters. For the filters in this class, the penalty for the length of the prototype filter translates mostly in communication cost

than in operator counts if a time-recursive implementation is considered. This is the exact analog of the situation with the sum-of-cosine windows in STFT (see Section 4.3).

In this Section, we consider the time-recursive architectural implementation for a specific ELT with basis length equal to $4N$. In other words, this ELT produces N transform coefficients based on $4N$ consequent input data samples. The time-recursive implementation of this transform retains the locality property and the implementation cost is slightly higher from the one of MLT. Consequently, we anticipate that based on the proposed implementation this ELT will become an important candidate in applications such as transform domain adaptive filtering and real-time data coding [24, 56].

4.5.1 Architecture for the Forward ELT

The ELT under consideration operates on segments of data of length $4N$, $x(t+n-4N+1), n = 0, 1, \cdots, 4N-1$ and it produces N output coefficients $X_{ELT}(k,t)$, $k = 0, 1, \cdots, N-1$ as follows [39]:

$$X_{ELT}(k,t) = \sqrt{\tfrac{2}{N}} \sum_{n=0}^{4N-1} \left[-\tfrac{1}{2\sqrt{2}} + \tfrac{1}{2} \cos \tfrac{\pi}{N} \left(n + \tfrac{1}{2}\right) \right] \cos \left[\tfrac{\pi}{N} \left(k + \tfrac{1}{2}\right) \left(n + \tfrac{1}{2} + \tfrac{N}{2}\right) \right] x(t + n - 4N + 1), \tag{4.5.1}$$

where $t = 0, 1, \cdots$. The sequence of the k^{th} output coefficients $X_{ELT}(k,t), t = 0, 1, \cdots$ can be thought of as the output of the mapping operator

$$h_{k,n} = \frac{1}{2\sqrt{2N}} \left[\alpha + 2 \cos \frac{\pi}{N} \left(n + \frac{1}{2}\right) \right] \cos \left[\frac{\pi}{N} \left(k + \frac{1}{2}\right) \left(n + \frac{1}{2}\right) + \left(k + \frac{1}{2}\right) \frac{\pi}{2} \right], \tag{4.5.2}$$

where $\alpha = -\sqrt{2}$. After a few algebraic manipulations, we derive the following decomposition of the mapping operator:

$$h_{k,n} = \frac{1}{2\sqrt{2N}} \left[-f_{k-1,1}(n) + \alpha f_{k,0}(n) + \mathbf{f}_{k+1,1}(n) \right], \quad k = 0, 1, \cdots, N-1, \tag{4.5.3}$$

where

$$\begin{bmatrix} f_{k,0}(n) \\ f_{k,1}(n) \end{bmatrix} = \begin{bmatrix} \cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}\right)+\left(k+\frac{1}{2}\right)\frac{\pi}{2}\right] \\ \sin\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}\right)+\left(k+\frac{1}{2}\right)\frac{\pi}{2}\right] \end{bmatrix}$$

is the associated kernel group $\mathbf{f}_k(n)$. For this kernel group we have $\mathbf{f}_k(n-1) = \mathbf{R}_k\mathbf{f}_k(n)$, where

$$\mathbf{R}_k = \begin{bmatrix} \cos\frac{\pi}{N}\left(k+\frac{1}{2}\right) & \sin\frac{\pi}{N}\left(k+\frac{1}{2}\right) \\ -\sin\frac{\pi}{N}\left(k+\frac{1}{2}\right) & \cos\frac{\pi}{N}\left(k+\frac{1}{2}\right) \end{bmatrix}. \tag{4.5.4}$$

We also have

$$\begin{bmatrix} f_{k,0}(0) \\ f_{k,1}(0) \end{bmatrix} = S\begin{bmatrix} f_{k,0}(4N) \\ f_{k,1}(4N) \end{bmatrix} = \begin{bmatrix} \cos\frac{\pi}{2}\left(1+\frac{1}{N}\right)\left(k+\frac{1}{2}\right) \\ \sin\frac{\pi}{2}\left(1+\frac{1}{N}\right)\left(k+\frac{1}{2}\right) \end{bmatrix}, \tag{4.5.5}$$

where $S = 1$. Therefore, the periodicity property is satisfied. Since both member functions of the kernel group appear in the decomposition (4.5.3), the lattice architecture is recommended (cf. Fig. 3.1). In Fig. 4.17, we provide the time-recursive architecture for the case of $N = 8$. The details of the lattice modules in Fig. 4.17 can be obtained by a simple parameter substitution.

4.5.2 Cost Issues

The cost of the time-recursive implementation of this ELT is $N + 2$ rotation circuits, $3N + 4$ multipliers and $4N + 4$ adders. All remarks we have made in Chapter 3 for the throughput rate of the DCT apply for the implementation of the MLT as well. The cost metrics for the multiprocessor implementation are summarized on Table 4.1. Observe that the implementation cost of the ELT is very close to the one of the DCT and the MLT, thus enabling the VLSI implementation of high fidelity audio coding schemes [39, 56, 25]. The cost metrics for the uniprocessor implementation of the sliding version of the ELT are given for completness on Table 4.2.

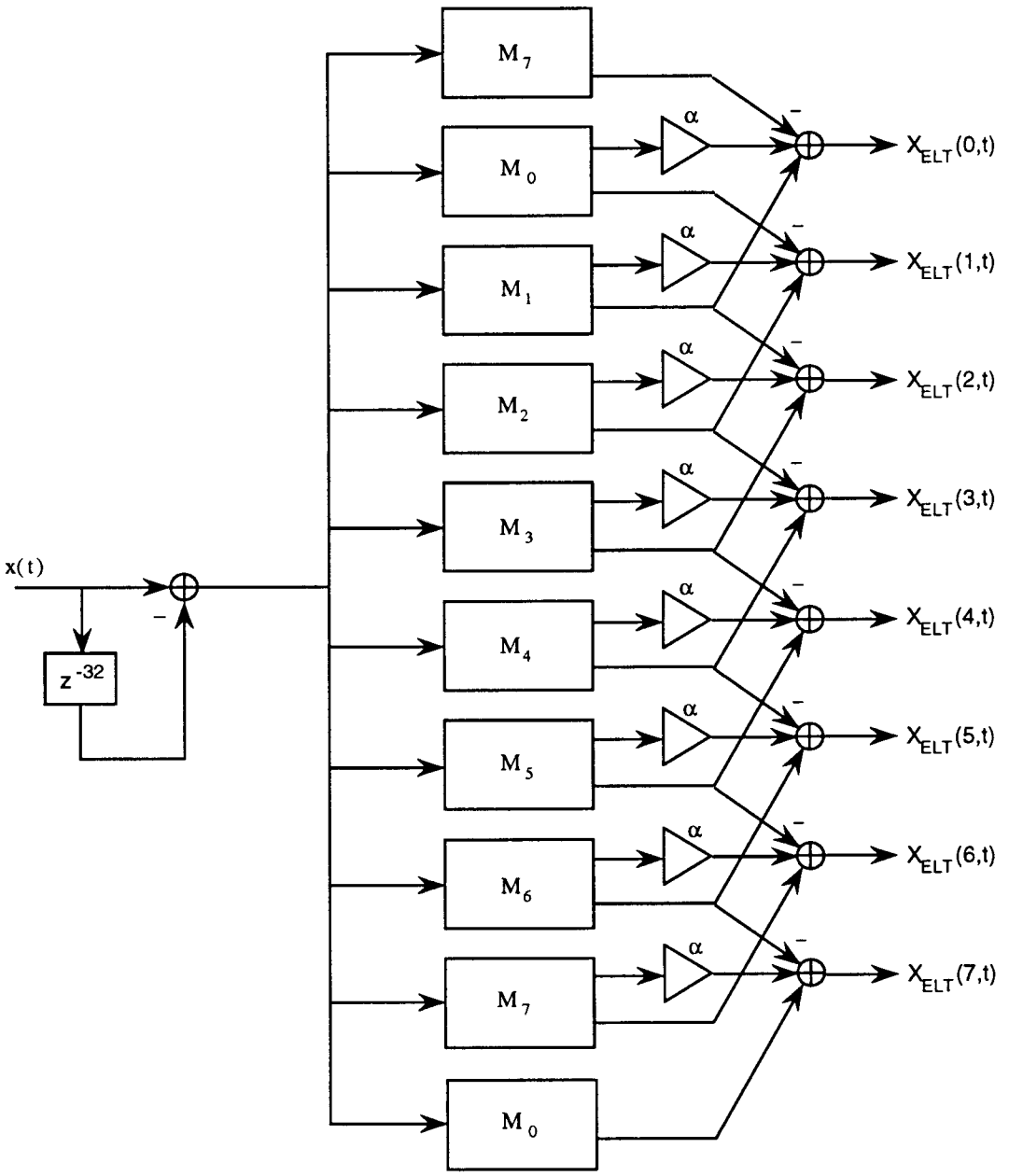


Figure 4.17: Time-recursive architecture for the ELT.

CHAPTER
5

An Example in Processing of 2D Data

Insofar, we have discussed the design of the building modules in time-recursive architectures (either lattice or IIR) and the implications to several applications in one-dimensional data processing.

The generalization of the time-recursive computation in separable, multi-dimensional data transforms has been introduced very recently [32, 9, 34], yielding very attractive architectures. These architectures exhibit all the advantages of the one-dimensional counterparts: they are modular and regular, they require local communication and they have linear cost (in terms of operation counts). In particular, they have been proved asymptotically optimal in area and speed [34]. Furthermore, they do not require any matrix transposition and they operate in a SIMO (single-input multiple-output) way, that is very appropriate in most applications with real-time computation requirements. In [34], the structure of the time-recursive architecture for a multi-dimensional separable transform is derived. Based on this, given the building module of the one-dimensional transform, one can design the architecture of the multi-dimensional transform in an inductive way. The combination of this result with the discussion we have made on the module design in Chapters 2 and 3 yield a powerful architecture design tool for multi-dimensional transforms with arbitrary dimensionality.

In this Chapter, we present the time-recursive computation of the 2-D Modulated Lapped Transform (2D MLT). Within this example, we present the basic concept of the induction process that develops the architecture of a higher-dimension data transform (in our case 2) based on the architecture of the lower-

dimension transform (in our case 1). Furthermore, we handle a number of subtleties mainly due to the communication of neighboring modules in the 1D MLT. Such communication is common in the time-recursive implementation of a number of interesting transforms as we have seen in Chapter 4. Finally, the architecture we propose for the 2D MLT has an interest in its own right, since the 2D MLT finds application in image coding [38, 49]. In Section 5.1, we derive the algorithm for computing the 2-D MLT. In Section 5.2, we present the architectural implementation. In Section 5.3, we discuss the implied cost.

5.1 Algorithm

In Chapter 4, we have seen that the one-dimensional MLT operates on segments of data of length $2N$, $x(t+m), m = 0, 1, \cdots, 2N-1$ and it produces N output coefficients $X_{MLT}(k,t), k = 0, 1, \cdots, N-1$ according to the formula:

$$X_{MLT}(k,t) = -c_k \sqrt{\frac{1}{2N}} \sum_{m=0}^{2N-1} \left[f_{k+1,0}(m) + f_{k,1}(m) \right] x(t+m), \qquad (5.1.1)$$

for $t = 0, 1, \cdots$, where

$$\begin{bmatrix} f_{k,0}(m) \\ f_{k,1}(m) \end{bmatrix} = \begin{bmatrix} \cos\left[\frac{\pi}{N} k \left(m + \frac{1}{2}\right) + \left(k + \frac{1}{2}\right) \frac{\pi}{2}\right] \\ \sin\left[\frac{\pi}{N} k \left(m + \frac{1}{2}\right) + \left(k + \frac{1}{2}\right) \frac{\pi}{2}\right] \end{bmatrix} \triangleq \mathbf{f}_k(m), \quad k = 0, 1, \cdots, N$$

and c_k are constants that take values in $\{-1, 1\}$.

The two-dimensional MLT operates on data blocks of size $2N \times 2N$, $x(t+m, s+n), m, n = 0, 1, \cdots, 2N-1$ and it produces N^2 output coefficients $X_{MLT}(k, l, t, s), k, l = 0, 1, \cdots, N-1$ as follows:

$$\begin{aligned} X_{MLT}(k,l,t,s) &= \frac{1}{2N} c_k c_l \sum_{m=0}^{2N-1} \sum_{n=0}^{2N-1} \left[f_{k+1,0}(m) f_{l+1,0}(n) + f_{k+1,0}(m) f_{l,1}(n) \right. \\ &+ \left. f_{k,1}(m) f_{l+1,0}(n) + f_{k,1}(m) f_{l,1}(n) \right] x(t+m, s+n). \quad (5.1.2) \end{aligned}$$

So, $X_{MLT}(k,l,t,s)$ can be evaluated as

$$\begin{aligned} X_{MLT}(k,l,t,s) &= c_k c_l \left[X_{0,0}(k+1,l+1,t,s) + X_{0,1}(k+1,l,t,s) \right. \\ &+ \left. X_{1,0}(k,l+1,t,s) + X_{1,1}(k,l,t,s) \right], \end{aligned} \tag{5.1.3}$$

where

$$X_{p,q}(k,l,t,s) = \frac{1}{2N} \sum_{m=0}^{2N-1} \sum_{n=0}^{2N-1} f_{k,p}(m) f_{l,q}(n) x(t+m, s+n), \quad p = 0,1, \quad q = 0,1$$

and $k, l = 0, 1, \cdots, N$. If we define

$$\mathbf{X}(k,l,t,s) \triangleq \begin{bmatrix} X_{0,0}(k,l,t,s) & X_{0,1}(k,l,t,s) \\ X_{1,0}(k,l,t,s) & X_{1,1}(k,l,t,s) \end{bmatrix} \tag{5.1.4}$$

we obtain the following expression for the 2D MLT

$$\mathbf{X}(k,l,t,s) = \frac{1}{2N} \sum_{m=0}^{2N-1} \sum_{n=0}^{2N-1} \mathbf{f}_k(m) \mathbf{f}_l^T(n) x(t+m, s+n), \quad p = 0,1, \quad q = 0,1. \tag{5.1.5}$$

The algorithm for computing the 2-D MLT consists of two steps: first, we compute the four two-dimensional transforms in (5.1.4) and then add the outcomes according to (5.1.3). Since

$$\mathbf{f}_k(m-1) = \mathbf{R}_k \mathbf{f}_k(m), \quad k = 0, 1, \cdots, N, \tag{5.1.6}$$

where

$$\mathbf{R}_k = \begin{bmatrix} \cos \frac{k\pi}{N} & \sin \frac{k\pi}{N} \\ -\sin \frac{k\pi}{N} & \cos \frac{k\pi}{N} \end{bmatrix}, \tag{5.1.7}$$

the $2N$ vectors

$$\mathbf{Y}(k,n,t,s) \triangleq \sum_{m=0}^{2N-1} \mathbf{f}_k(m) x(t+m, s+n), \quad n = 0, 1, \cdots, 2N-1$$

can be computed in a time-recursive way as follows:

$$\mathbf{Y}(k,n,t+1,s) = \mathbf{R}_k \left[\mathbf{Y}(k,n,t,s) - \mathbf{f}_k(0)x(t,s+n) + \mathbf{f}_k(2N)x(t+2N,s+n)\right], \tag{5.1.8}$$

where

$$\mathbf{f}_k(0) = \mathbf{f}_k(2N) = \begin{bmatrix} \cos\left[\frac{k\pi}{2N} + \left(k+\frac{1}{2}\right)\frac{\pi}{2}\right] \\ \sin\left[\frac{k\pi}{2N} + \left(k+\frac{1}{2}\right)\frac{\pi}{2}\right] \end{bmatrix}. \tag{5.1.9}$$

From (5.1.9) and the fact that

$$\mathbf{X}(k,l,t,s) = \sum_{n=0}^{2N-1} \mathbf{Y}(k,n,t,s)\mathbf{f}_l^T(n)$$

we can derive the following time-recursive algorithm for computing $\mathbf{X}(k,l,t+m,s)$, $m = 1,2,\cdots$:

$$\mathbf{X}(k,l,t+1,s) = \mathbf{R}_k \left[\mathbf{X}(k,l,t,s) + \mathbf{f}_k(0)\delta^T(l,t,s)\right], \tag{5.1.10}$$

$k,l = 0,1,\cdots,N$, where

$$\delta(l,t,s) \triangleq \sum_{n=0}^{2N-1} \tilde{x}(t,s+n)\mathbf{f}_l(n), \quad l = 0,1,\cdots,N$$

and

$$\tilde{x}(t,s) = x(t+2N,s) - x(t,s). \tag{5.1.11}$$

Based on (5.1.6) we derive a time-recursive algorithm for computing $\delta(l,t,s+n)$, $n = 0,1,\cdots,2N-1$:

$$\delta(l,t,s+1) = \mathbf{R}_l \left[\delta(l,t,s) + \mathbf{f}_l(0)\left(\tilde{x}(t,s+2N) - \tilde{x}(t,s)\right)\right], \tag{5.1.12}$$

for $l = 0,1,\cdots,N$.

In the perspective of the discussions on block transform implementations in Chapter 2, we can easily see that the term $\tilde{x}(t,s)$ in (5.1.11) reduces to $\tilde{x}(t,s) = x(t+2N,s)$ if the block MLT is considered (as opposed to the sliding

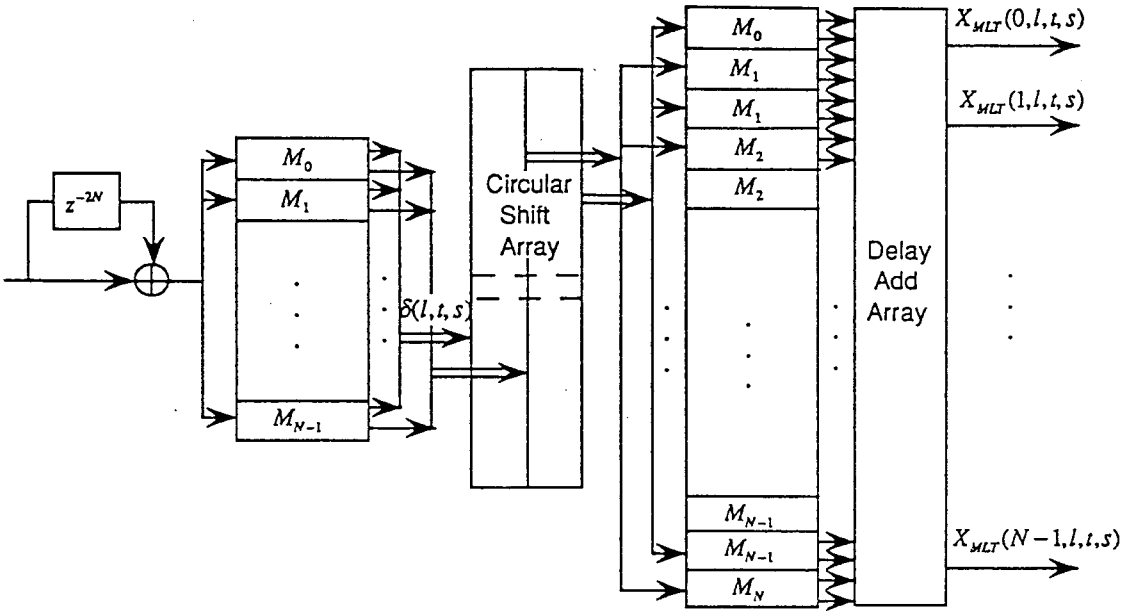


Figure 5.1: Recursive architecture for the 2-D MLT.

one). Similarly, the term $\tilde{x}(t,s)$ in (5.1.12) can be disregarded. Consequently, for the case of the block MLT the expression (5.1.12) becomes:

$$\delta(l,t,s+1) = \mathbf{R}_l\left[\delta(l,t,s) + \mathbf{f}_l(0)x(t+2N,s+2N)\right], \quad l = 0,1,\cdots,N. \quad (5.1.13)$$

In summary, the time-recursive algorithm for computing the 2-D MLT of the data strip $x(t+m,s+n), m,n = 0,1,\cdots,2N-1$ and $t = 0,1,\cdots,$ s =fixed, is given by (5.1.3), (5.1.10) and (5.1.12) for the case of the sliding 2-D MLT, while for the block 2-D MLT the latter can be replaced by (5.1.13).

5.2 Architecture

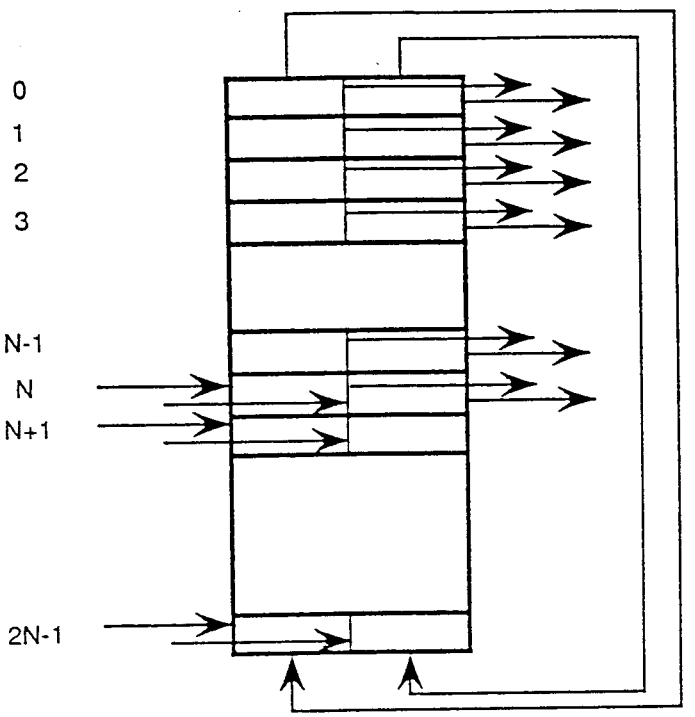
The architectural implementation of the block 2-D MLT is demonstrated in Fig. 5.1. The computational core consists of a cascade of two 1-D MLT stages. The first one is responsible for evaluating (5.1.13), that is transforming the input sequence with the proper cosine and sine kernels at the frequencies indexed by $l = 0,1,\cdots,N-1$. This stage is composed of N modules that need not exchange any information. The structure of these modules is depicted on Fig. 4.13.

A Circular Shift Array (CSA) (Fig. 5.2.a) of size $2N \times 2$ is used to feed each module of the second 1-D MLT stage with the output sequence of the first stage. The lower $N \times 2$ entries of the CSA are loaded every N time instants. At this point, the $(N+l)^{\text{th}}$ entry with $l = 0, 1, \cdots, N-1$ contains the 2×1 term $\delta(l, t, s)$. Then, the CSA rotates the data upwards and at each clock cycle it feeds the 1-D MLT modules of the second stage with the upper $N+1$ pairs of CSA entries. The contents (l,t) of the CSA (viz. frequency of the first stage,load time instant) are shown in Fig. 5.2.b for the case of $N = 4$.

The second stage of 1-D MLT modules evaluates (5.1.10). At this stage, we need $2N$ modules that differ from the one in Fig. 4.13 only at the delay elements. The latter will be z^{-N} instead of z^{-1}. This happens because for each vertical frequency component k the associated module needs to store and compute N transform coefficients that correspond to the frequencies $(k, l), l = 0, 1, \cdots, N-1$. The double copies of these modules are needed for transforming both terms of the 2×1 sequence of $\delta(l, t, s)$. Finally, the module with $k = N$ is used for computing the terms $X_{0,0}(k+1, l+1, t, s), X_{0,1}(k+1, l, t, s)$ in (5.1.3) with $k = N-1$. If we disregard the data skew introduced by the CSA, the output of the second 1-D MLT stage at a fixed time instant will consist of the terms $X_{p,q}(k, l, t, s), p = 0, 1, \quad q = 0, 1, \quad k = 0, 1, \cdots, N$ for some values of l, t and s in the ranges $\{0, 1, \cdots, N-1\}$, $\{0, N, 2N, \cdots\}$ and $\{0, N, 2N, \cdots\}$ respectively.

The Delay Add Array (DAA) (see Fig. 5.3) is responsible for synchronizing the skewed data and evaluating (5.1.3). Note that the elements denoted by D_N store the terms $X_{0,0}(k+1, 0, t, s), X_{1,0}(k, 0, t, s)$ at the beginning of each period of N cycles. These terms are equal to $X_{0,0}(k+1, N, t, s)$ and $X_{1,0}(k, N, t, s)$ respectively and they are utilized at the last cycle of each N cycle period.

The architecture in Fig. 5.1 can be modified to avoid the need of the delay elements that handle the data skew. This can be achieved by feeding all the modules in the second 1-D MLT stage with the same pair of entries of the CSA, for instance the first one. In this case, the segments of delay elements $\left[z^{-k}\ z^{-(k+1)}\ z^{-k}\ z^{-(k+1)}\right], k = 0, 1, \cdots, N$ in the DAA will be replaced by $N+1$ copies of the segment $[1\ z^{-1}\ 1\ z^{-1}]$.



a.

Time :	3x4	11	10	9	2x4	7	6	5	1x4	3	2	1
CSA	0											
0	20	13	12	11	10	03	02	01	00			
1	21	20	13	12	11	10	03	02	01	00		
2	22	21	20	13	12	11	10	03	02	01	00	
3	23	22	21	20	13	12	11	10	03	02	01	00
4	30	23	22	21	20	13	12	11	10	03	02	01
5	31	10	23	22	21	00	13	12	11		03	02
6	32	11	10	23	22	01	00	13	12			03
7	33	12	11	10	23	02	01	00	13			

b.

Figure 5.2: The Circular Shift Array: a. the architecture, b. the contents for the special case $N = 4$.

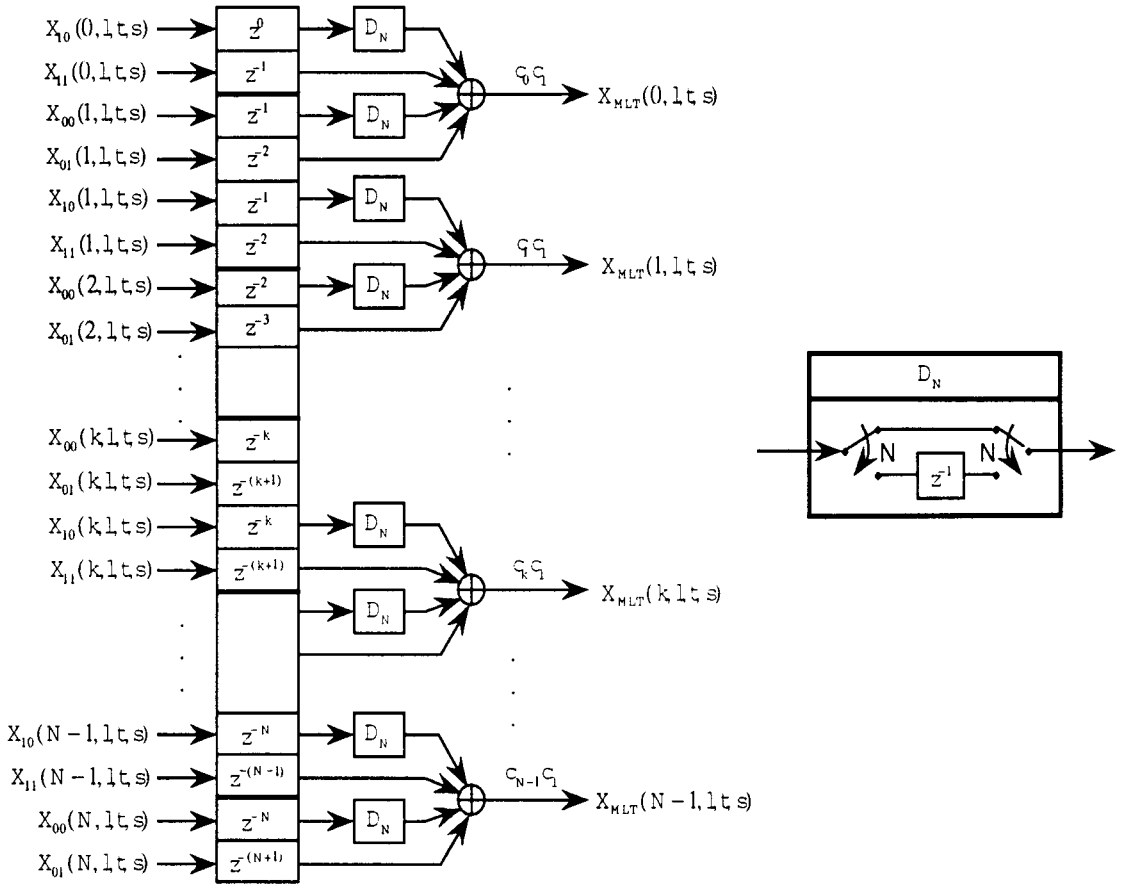


Figure 5.3: The Delay Add Array.

5.3 Implementation Issues

The proposed 2D MLT architecture is built using $3N$ basic processing elements, the 1D MLT modules, arranged in two linear stages, and avoiding the matrix transposition which is common in most of the separable 2D transform architectures. Overall, the 2D MLT architecture requires $O(N)$ processing elements, and $O(N^2)$ storage elements (one word each). Moreover, the architecture is regular and modular, using only local communications, and thus suitable for VLSI implementation.

CHAPTER
6

Conclusions and Further Research

In this dissertation, we have introduced an architectural framework for time-recursive computation that is particularly useful in developing efficient VLSI architectures for a variety of problems demanding real-time computation. Furthermore, based on this framework, we have obtained novel architectures for a number of data transforms and QMF banks. The resulted architectures are modular, regular, scalable, they require local communication and linear cost (in terms of operator counts). Real-time processing of audio, video, sonar and radar data can be benefited by these results. In what follows, we summarize our contributions in the perspective of the general setup of the three mappings **computation specification-to-algorithm, algorithm-to-architecture** and **architecture-to-VLSI implementation** and we suggest directions for further research.

The most important contribution in our work is the architectural framework for time-recursive computation. We have embodied the traditional two step approach of the computation specification-to-algorithm and the algorithm-to-architecture into a single step **computation specification-to-architecture**, for certain class of computations (namely the class of linear, discrete-time, time-invariant and compactly supported operators) and a class of architectures (namely the time-recursive architectures). We have revealed the common infrastructure of the time-recursive computations that have appeared in diverse areas in the literature, such as adaptive filtering, real-time data compression and spectrum analysis. We have introduced the **shift property**, the **differ-**

ence equation property and the **periodicity property** and we have shown how these dictate first, whether a time-recursive architecture is appropriate for a given computation, second, which is the best time-recursive architecture for this computation and third, what are the values of the parameters of the architecture. We have incorporated the appropriate tests and procedures into a **Generic Design Procedure** that can address the above issues in a routine way. The use and effectiveness of this Generic Design Procedure is further demonstrated by means of specific examples, namely the Discrete Cosine Transform (DCT), the Discrete Fourier Transform (DFT) and the Discrete Wavelet Transform (DWT). We have highlighted the implementation differences between the **sliding data transforms** and the **block data transforms**. We have discussed in detail the interpretation of cost and time requirements in the design of time-recursive architectures for real-time data transformation, by taking into account the SIMO (single-input multiple-output) nature of the computation. This model of computation is very useful in communication applications, where the data arrive in a serial way.

By using this architectural framework we have obtained novel architectures for the uniform-DFT QMF bank, the cosine modulated QMF bank, the 1-D and 2-D Modulated Lapped Transform (MLT), as well as an Extended Lapped Transform (ELT). These results are particularly important since MLT and ELT have been incorporated by the ISO-MPEG and ASPEC standards for audio coding with the name Modified DCT (MDCT). Furthermore, the architectural implementation of the Cepstral Transform and a Short Time Fourier Transform are considered based on the time-recursive architecture of the DFT. All of the above designs are modular, regular, with local communication and linear cost in operator counts. In particular, the 1-D MLT requires $2N + 3$ multipliers, $3N + 3$ adders and $N - 1$ rotation circuits, where N is the data block size. The 2-D MLT requires 3 1-D MLT circuits and no transposition. The ELT requires $3N+4$ multipliers, $4N+4$ adders and $N+2$ rotation circuits. These results have an impact in real-time audio and video data compression, in frequency domain adaptive filtering and in spectrum analysis.

Finally, it note the trade-off of the two alternative architectures we have proposed for the Cepstral Transform: one requires $3N-1$ adders, $N-2$ rotation circuits and N "fast" logarithm circuits, while the other requires $6N-1$ adders, $3N-6$ rotation circuits and N "slow" logarithm circuits. We anticipate that the "fast" logarithm circuits can be implemented with an analog nonlinearity, while the "slow" logarithm circuits can be implemented digitally. Consequently, we conjecture that a mixed digital-analog design is substantially better from the purely digital counterpart.

The work in this dissertation stimulates some important questions in three main directions: first, implementing the Generic Design Procedure by a CAD (Computer Aided Design) tool. The input data of this tool can be in the form of an algebraic formula. The output can be either a description of the architecture (for example the structure of the elementary building modules, the values of the module parameters and a netlist associating these modules) or even the layout of the circuit. Such tool should incorporate finite wordlength simulation capabilities, as well as detailed information about implementing the arithmetic operators. Particularly, the distributed arithmetic schemes and/or the CORDIC processor architectures to be used should be studied more carefully.

Second, identifying the applications for which the time-recursive computation is appropriate. This is a very important issue since real-time computation is an emerging need in data communication systems, especially for multidimensional data processing. The SIMO nature of computation, the local communication and the linear cost of the time-recursive architectures suggest that the time-recursive computation can play a key role in the close future.

The third research direction is concerned with the development of two additional frameworks for time-recursive computation. Both originate from modifications to the equation specifying the shift property (2.2.1). Consider substituting the linear expression at the right hand side by a nonlinear one. One will chose the nonlinearities that can be implemented in hardware in an efficient way, for example by an analog circuit. The implied questions concern the scope of the computations that can be implemented by the modified time-recursive archi-

tecture, as well as the resulted implementation cost. A second modification of (2.2.1) is obtained if we replace the difference equation by a differential equation. The result will be the continuous-time counterpart of the discrete-time framework we have presented. The implementation will involve analog circuits. The question of interest is again about the scope of this computational model and the potential applications.

APPENDIX
A

Proofs

Proof of Lemma 2.2:

1. The size of the kernel group is $M = 1$ and the one kernel is $f_0(n) = cb^n$. We have

$$f_0(n-1) = cb^{n-1} = \frac{1}{b} f(n), \quad \text{which gives} \quad r_{00} = \frac{1}{b}$$

and consequently SP is satisfied. The architecture that is implied by (2.2.3) is depicted in Fig. 2.2. Note that $f_0(0) = c$ and $f_0(N) = cb^N$.

2. We have $M = 2$ and

$$\mathbf{f}(n) = \begin{bmatrix} f_0(n) \\ f_1(n) \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} b^n \\ b^{-n} \end{bmatrix}. \tag{1.0.1}$$

So,

$$\begin{bmatrix} f_0(n-1) \\ f_1(n-1) \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} b^{n-1} \\ b^{-n+1} \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} b^{-1} & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} b^n \\ b^{-n} \end{bmatrix}$$

$$= \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} b^{-1} & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix}^{-1} \begin{bmatrix} f_0(n) \\ f_1(n) \end{bmatrix}.$$

Therefore,

$$\begin{bmatrix} r_{00} & r_{01} \\ r_{10} & r_{11} \end{bmatrix} = \frac{1}{\delta} \begin{bmatrix} -c_{01}c_{10}b + c_{00}c_{11}b^{-1} & c_{00}c_{01}(b - b^{-1}) \\ -c_{10}c_{11}(b - b^{-1}) & c_{00}c_{11}b - c_{01}c_{10}b^{-1} \end{bmatrix}, \tag{1.0.2}$$

where $\delta = c_{00}c_{11} - c_{01}c_{10}$ and consequently SP is satisfied. The architectural implementation of the kernel group is shown in Fig. 2.1.

3. The size of the kernel group is M. For the p^{th} kernel function, $0 \leq p \leq M-1$, we have $f_p(n)/c_p = n^p$. Therefore,

$$\frac{f_p(n-1)}{c_p} = \sum_{q=0}^{p} \left[\begin{pmatrix} p \\ q \end{pmatrix} (-1)^{p-q} \right] \frac{f_q(n)}{c_q},$$

which gives

$$r_{pq} = \begin{cases} \frac{c_p}{c_q} \begin{pmatrix} p \\ q \end{pmatrix} (-1)^{p-q}, & q \leq p \\ 0, & q > p \end{cases}.$$

Consequently, SP is satisfied.

4. Suppose the kernel groups $\mathbf{f}(\cdot) = [f_0(\cdot)\ f_1(\cdot)\ \cdots\ f_{P-1}(\cdot)]^T$ and $\mathbf{g}(\cdot) = [g_0(\cdot)\ g_1(\cdot)\ \cdots\ g_{M-1}(\cdot)]^T$ satisfy SP. We will have $\mathbf{f}(n-1) = \mathbf{R}_f\mathbf{f}(n)$ and $\mathbf{g}(n-1) = \mathbf{R}_g\mathbf{g}(n)$, where $\mathbf{R}_f$ and $\mathbf{R}_g$ are constant matrices of proper dimensionality. For the union of the two kernel groups $\{\{f_p(\cdot)\}_{p=0}^{P-1}, \{g_p(\cdot)\}_{p=0}^{M-1}\}$ we have

$$\begin{bmatrix} \mathbf{f}(n-1) \\ \mathbf{g}(n-1) \end{bmatrix} = \begin{bmatrix} \mathbf{R}_f & 0 \\ 0 & \mathbf{R}_g \end{bmatrix} \begin{bmatrix} \mathbf{f}(n) \\ \mathbf{g}(n) \end{bmatrix},$$

and consequently, SP is satisfied.

5. We will consider the special case of two kernel groups that both are of size $M = 2$, $[f_0(\cdot)\ f_1(\cdot)]^T$ and $[g_0(\cdot)\ g_1(\cdot)]^T$. The cartesian product kernel group is $[f_0(\cdot)g_0(\cdot)\ f_1(\cdot)g_0(\cdot)f_0(\cdot)g_1(\cdot)\ f_1(\cdot)g_1(\cdot)]^T$. Since (2.2.1) holds for the two kernel groups we have

$$f_p(n-1) = r_{p0}f_0(n) + r_{p1}f_1(n), \quad p = 0, 1$$

and

$$g_q(n-1) = s_{q0}g_0(n) + s_{q1}g_1(n), \quad q = 0, 1$$

for some constant coefficients r_{pq}, s_{pq}, $p, q = 0, 1$. From the above we get

$$f_p(n-1)g_q(n-1) =$$
$$r_{p0}s_{q0}f_0(n)g_0(n) + r_{p0}s_{q1}f_0(n)g_1(n) + r_{p1}s_{q0}f_1(n)g_0(n) + r_{p1}s_{q1}f_1(n)g_1(n)$$

for $p, q = 0, 1$. Therefore, SP is satisfied by the cartesian product kernel group. Obviously, the procedure described here applies for the cartesian product of kernel groups with arbitrary sizes.

Proof of Lemma 2.3: The proof consists of two steps: First we show that all the discrete mapping operators of finite length can be expressed as linear combinations of exponential functions. Second, we show that such expressions can be implemented in a time-recursive way:

Given a mapping operator of length N we can express the coefficients of this operator as linear combinations of exponential kernel functions by taking the N-point DFT of the coefficients.

From Lemma 2.2, Statement 1, every exponential kernel function can be implemented recursively. From Lemma 2.2, Statement 4, the set of all the necessary exponential kernel functions can be implemented recursively. From the linearity property, the linear combination of the exponential kernel functions can be implemented recursively.

Proof of Lemma 2.5: We will proceed with this proof by showing that there are algorithms for the following computations:

1. Compute $\{\mathbf{A}, \mathbf{b}\}$ based on knowledge of $\mathbf{R}$ and $\mathbf{f}(0)$.
2. Compute $\{\mathbf{R}, \mathbf{f}(0)\}$ based on $\{\mathbf{A}, \mathbf{b}\}$.
3. Compute $\{\mathbf{A}, \mathbf{b}\}$ based on $\{\mathbf{f}(-1), \mathbf{f}(-2), \cdots, \mathbf{f}(-M), \gamma_1, \gamma_2, \cdots, \gamma_M\}$.
4. Compute $\{\mathbf{f}(-1), \mathbf{f}(-2), \cdots, \mathbf{f}(-M), \gamma_1, \gamma_2, \cdots, \gamma_M\}$ based on $\{\mathbf{A}, \mathbf{b}\}$.

The first two algorithms are straightforward implications of relation (2.2.9). Note the implicit nonsingularity assumption we have made for the matrix $\mathbf{R}$.

For the computation in 3 we follow four steps: First, compute the quantities $\mathbf{f}(n),\ n = 0, 1, \cdots, M-1$ based on $\mathbf{f}(n),\ n = -1, -2, \cdots, -M$ and (2.2.12). Since $\mathbf{f}(n) = \mathbf{A}^n\mathbf{b}$, the controllability matrix specified by the unknown quantities $\{\mathbf{A}, \mathbf{b}\}$ will be [27]

$$\mathcal{C} = \left[\mathbf{b}\ \mathbf{A}\mathbf{b}\ \cdots\ \mathbf{A}^{M-1}\mathbf{b}\right] = \left[\mathbf{f}(0)\ \mathbf{f}(1)\ \cdots\ \mathbf{f}(M-1)\right].$$

Second, by using relation (2.2.11) find the controller canonical form system matrix $\mathbf{A}_c$ and output vector $\mathbf{b}_c$ specified in (2.1.3). So, the controllability matrix of the controller canonical form is obtained:

$$\mathcal{C}_c = \left[\mathbf{b}_c\ \mathbf{A}_c\mathbf{b}_c\ \cdots\ \mathbf{A}_c^{M-1}\mathbf{b}_c\right].$$

Third, compute the matrix $\mathbf{T}$ that defines the similarity transform

$$\{\mathbf{A}_c, \mathbf{b}_c\} \longrightarrow \{\mathbf{A} = \mathbf{T}^{-1}\mathbf{A}_c\mathbf{T},\ \mathbf{b} = \mathbf{T}^{-1}\mathbf{b}_c\} \tag{1.0.3}$$

by using the relation [27]

$$\mathbf{T} = \mathcal{C}_c\mathcal{C}^{-1}.$$

Forth, the quantities $\{\mathbf{A}, \mathbf{b}\}$ are computed by the relations specified in (1.0.3).

The computation in 4 is as follows: From knowledge of $\{\mathbf{A}, \mathbf{b}\}$ obtain the corresponding pair in controller canonical form $\{\mathbf{A}_c, \mathbf{b}_c\}$ [27]. The desired coefficients $\gamma_1, \gamma_2, \cdots, \gamma_M$ can be obtained from the elements of the first row of the matrix $\mathbf{A}_c$ by using (2.2.11). The initial values $\mathbf{f}(-1), \mathbf{f}(-2), \cdots, \mathbf{f}(-M)$ can be obtained by simply evaluating the expression $\mathbf{f}(n) = \mathbf{A}^n\mathbf{b}$ for $n = -1, -2, \cdots, -M$.

Proof of Lemma 2.6: We will consider the special case of $M = 3$. The proof can be easily generalized for arbitrary values of M.

One can verify that the transfer functions from the input to $\hat{X}_0(t)$, $\hat{X}_1(t)$ and

$\hat{X}_2(t)$ in Fig. 2.3 respectively are

$$-f_0(0)z^{-N} + f_0(N), \quad -f_1(0)z^{-N} + f_1(N) \quad \text{and} \quad -f_2(0)z^{-N} + f_2(N).$$

Consequently, from the $\mathcal{Z}$ transform of (2.3.1) we get

$$\hat{X}_p(z) = D_p\hat{X}_0(z) \quad \text{or} \quad -f_p(0)z^{-N}+f_p(N) = D_p\left[-f_0(0)z^{-N} + f_0(N)\right], \quad p = 1, 2.$$

Since this is true for every z in some open interval, the latter implies

$$\begin{bmatrix} f_p(0) \\ f_p(N) \end{bmatrix} = D_p \begin{bmatrix} f_0(0) \\ f_0(N) \end{bmatrix} \tag{1.0.4}$$

for $p = 1, 2$, or equivalently

$$\frac{f_p(0)}{f_0(0)} = \frac{f_p(N)}{f_0(N)} \quad \text{or} \quad \frac{f_0(N)}{f_0(0)} = \frac{f_p(N)}{f_p(0)}, \quad p = 1, 2,$$

which in turn is equivalent to (2.3.2).

Proof of Lemma 2.7: If we have $b = e^{j\frac{k\pi}{N}}$ one can verify that (2.3.2) holds with ratio value $1/S = (-1)^k$, by simply substituting the above expression of b in (2.3.3).

On the other hand, suppose that (2.3.2) is satisfied by a kernel group specified by (2.3.3) with $b = e^{j\beta}$. If $1/S$ is the value of the ratio in (2.3.2), then the latter implies:

$$c_{p0}e^{j\beta N} + c_{p1}e^{-j\beta N} = \frac{1}{S}(c_{p0} + c_{p1}). \quad p = 0, 1.$$

The left hand side expression can also be written as

$$c_{p0}(\cos\beta N + j\sin\beta N) + c_{p1}(\cos\beta N - j\sin\beta N) =$$

$$\cos\beta N(c_{p0} + c_{p1}) + j\sin\beta N(c_{p0} - c_{p1}), \qquad p = 0, 1.$$

Therefore we have either $c_{p0} = c_{p1}$, $p = 0, 1$ or $\beta = j\frac{k\pi}{N}$. Since the first condition

yields $c_{00}c_{11} - c_{01}c_{10} = 0$, the alternative must be true.

In turn, the above result implies

$$\frac{1}{S} = \cos \beta N = \cos k\pi = (-1)^k.$$

Proof of Lemma 2.8: Let $X_p(t)$, $t = 0, 1, \cdots$ be the output data of the mapping operation defined by the operator

$$[f_p(0)\ f_p(1)\ \cdots\ f_p(N-1)].$$

From (2.2.3) we get

$$X_p(t) = \sum_{q=0}^{M-1} r_{pq} \left[X_q(t-1) + \hat{X}_q(t)\right], \quad p = 0, 1, \cdots, M-1, \quad t = 1, 2, \cdots \tag{1.0.5}$$

where

$$\hat{X}_q(t) = -f_q(0)x(t-N) + f_q(N)x(t), \quad q = 0, 1, \cdots, M-1. \tag{1.0.6}$$

Consider the unilateral $\mathcal{Z}_+$ transform, defined as

$$X(z) = Z_+\{x(t)\} = \sum_{t=0}^{+\infty} x(t)z^{-t}. \tag{1.0.7}$$

Since

$$Z_+\{x(t-m)\} = z^{-m}X(z), \quad \text{for every integer} \quad m > 0, \tag{1.0.8}$$

the $\mathcal{Z}_+$ transform of (1.0.5) and (1.0.6) gives

$$X_p(z) = \sum_{q=0}^{M-1} r_{pq} \left[z^{-1}X_q(z) + \hat{X}_q(z)\right], \quad p = 0, 1, \cdots, M-1, \tag{1.0.9}$$

where

$$\hat{X}_q(z) = \left[-f_q(0)z^{-N} + f_q(N)\right] X(z), \quad q = 0, 1, \cdots, M-1. \tag{1.0.10}$$

From (1.0.9) we have

$$\sum_{q=0, q\neq p}^{M-1} r_{pq} z^{-1} X_q(z) + (-1 + r_{pp} z^{-1}) X_p(z)$$

$$= -\sum_{q=0}^{M-1} r_{pq} \hat{X}_q(z) = -X(z) \sum_{q=0}^{M-1} r_{pq} \left[-f_q(0)z^{-N} + f_q(N)\right],$$

$p = 0, 1, \cdots, M-1$. If we solve the above system of equations for $X_p(z), p = 0, 1, \cdots, M-1$, we obtain

$$X_p(z) = H_p(z) X(z), \quad p = 0, 1, \cdots, M-1,$$

where $H_p(z)$ can be brought into the form specified in Lemma 2.8 after a few algebraic manipulations.

Proof of Lemma 2.9: First, we define the $\mathcal{Z}_N$ transform of a discrete time function $f(n)$ over the time segment $\{0, \cdots, N-1\}$

$$\mathcal{Z}_N\{f(n)\} = \sum_{n=0}^{N-1} f(n) z^{-n}. \tag{1.0.11}$$

This variation of the $\mathcal{Z}$ transform is appropriate for the frequency domain representation of the kernel functions we consider here, since these functions are defined on a bounded segment of the time axes. On the other hand, we will use the unilateral $\mathcal{Z}_+$ transform as the frequency domain representation of the input signal $x(t)$ and the output signal $X(t)$, since these signals are defined on the semi-infinite sequence of time instances $t = 0, 1, \cdots$.

Let $F(z) = \mathcal{Z}_N\{f_p(n)\}$. Based on (1.0.11) we can show that

$$\mathcal{Z}\{f_p(n-1)\} = z^{-1}F(z) + f_p(-1) - z^{-N}f_p(N-1) \quad \text{and}$$

$$\mathcal{Z}\{f_p(n-2)\} = z^{-2}F(z) + f_p(-2) + z^{-1}f_p(-1) - z^{-N}f_p(N-2) - z^{N-1}f_p(N-1). \tag{1.0.12}$$

Also, we have

$$\tilde{F}(z) = z^{-N+1}F(z^{-1}), \tag{1.0.13}$$

where

$$\tilde{F}(z) = \mathcal{Z}_N\{\tilde{f}_p(n)\} \quad \text{and} \quad \tilde{f}(n) = f_p(N-1-n), \quad n = 0, 1, \cdots, N-1.$$

By taking the $\mathcal{Z}_N$ transform of both sides of (2.3.8), using (1.0.12) and solving for $F(z)$, we obtain:

$$F(z) = \frac{f_p(0) + \gamma_2 f_p(-1)z^{-1} - z^{-N}\left[f_p(N) + \gamma_2 f_p(N-1)z^{-1}\right]}{1 - \gamma_1 z^{-1} - \gamma_2 z^{-2}}. \tag{1.0.14}$$

From (2.2.2) we have

$$X(t+N-1) = \sum_{n=0}^{N-1} f_p(n)x(t+n),$$

or equivalently

$$y(t) = \sum_{n=0}^{N-1} x(t-n)\tilde{f}(n) \tag{1.0.15}$$

where $y(t) = X(t+N-1)$. By taking the $\mathcal{Z}_+$ transform of both sides of (1.0.15) and using (1.0.8) we obtain:

$$Y(z) = \sum_{n=0}^{N-1} \tilde{f}(n)\left[z^{-n}X(z)\right] = X(z)\sum_{n=0}^{N-1} \tilde{f}(n)z^{-n} = X(z)\tilde{F}(z).$$

By substituting (1.0.13) we get

$$Y(z) = z^{-N+1}F(z^{-1})X(z),$$

and therefore, the transfer function we were after is

$$H(z) = z^{-N+1}F(z^{-1}). \tag{1.0.16}$$

If we substitute the expression (1.0.14) of $F(z)$ in the above we obtain the transfer function specified in (2.3.9).

Proof of Lemma 2.10: One can verify that

$$\mathcal{Z}_N\{f_p(n-q)\} = z^{-q}F(z) + \sum_{n=1}^{q}\left[f_p(-n)z^{-q+n}f_p(N-n)z^{-N-q+n}\right], \tag{1.0.17}$$

where the $\mathcal{Z}_N$ transform is defined by (1.0.11) and $F(z) = \mathcal{Z}_N\{f_p(n)\}$. By taking the $\mathcal{Z}_N$ transform of (2.2.12),using (1.0.17) and solving for $F(z)$ we obtain:

$$F(z) = \frac{\sum_{q=1}^{M}\gamma_q\left[\sum_{n=1}^{q}f_p(-n)z^{-q+n} - z^{-N}\sum_{n=1}^{q}f_p(N-n)z^{-N-q+n}\right]}{1-\sum_{q=1}^{M}\gamma_q z^{-k}}.$$

By substituting this expression in (1.0.16) we obtain (2.3.11).

Proof of Lemma 3.1: Consider the $\mathcal{Z}_N$ transform of a discrete time function $f(n)$ over the time segment $\{0,\cdots,N-1\}$

$$\mathcal{Z}_N\{f(n)\} = \sum_{n=0}^{N-1}f(n)z^{-n}.$$

This variation of the $\mathcal{Z}$ transform is appropriate for the frequency domain representation of the kernel functions we consider in this paper, since these functions are defined on a bounded segment of the time axes. Let $\mathbf{Y}(z)$ be the $\mathcal{Z}_+$ transform of $\mathbf{y}(Nt+n)$:

$$\mathbf{Y}(z) = \sum_{n=0}^{\infty}\mathbf{y}(Nt+n)z^{-n}$$

One can show that

$$\mathcal{Z}_+\{\mathbf{y}(Nt+n+1)\} = z\left[\mathbf{Y}(z) - \mathbf{y}(Nt+0)\right]. \tag{1.0.18}$$

By taking the unilateral $\mathcal{Z}_+$ transform of both sides of (3.2.11), substituting (1.0.18) and solving for $\mathbf{Y}(z)$ we obtain:

$$\mathbf{Y}(z) = \left(\mathbf{I} - z^{-1}\mathbf{R}^{-1}\right)^{-1} y(Nt),$$

or

$$\mathbf{Y}(z) = \left(\mathbf{I} - z^{-1}\mathbf{R}^{-1}\right)^{-1} y(N(t-1)+N). \tag{1.0.19}$$

From (3.2.9) we get

$$\mathbf{y}(N(t-1)+N) = \mathbf{f}(N)X(0, N(t-1)).$$

By substituting the above expression in (1.0.19) we can derive the transfer function $H(z)$ specified in Lemma 3.1 after a few simple algebraic manipulations.

REFERENCES

[1] Anderson, L.A. and Yau, H.C and Manry, M.T., Recursive Approximation of the Energy Spectral Density, IEEE Trans. on Signal Processing. Vol. 40, No. 12, pp. 3059-3062, Dec. 1992.

[2] Image Coding Using Wavelet Transform, Antonini, M. and Barlaud, M. and Mathieu, P. and Daubechies, I., IEEE Trans. on Signal Processing, 1992.

[3] Beraldin, J.A. and Aboulnasr, T. and Steenhart, W., Efficient one-dimensional systolic array realization of the discrete Fourier transform, IEEE Trans. on Circuits and Systems, Vol. 36, pp. 95-100, 1989.

[4] Beraldin, J.A. and Steenhart, W., Oveflow analysis of a fixed-point implementation of the Goertzel algorithm, IEEE Trans. on Circuits and Systems, Vol. 36, pp. 322-324, 1989.

[5] Chihara, T.S., An Introduction to Orthogonal Polynomials, Gordon and Breach Science Pub., New York, 1978.

[6] Babic, H. and Temes, G.C., Optimum Low-Order Windows for Discrete Fourier Transform Systems, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-24, No. 6, pp. 512–517, Dec. 1976.

[7] Bitmead, R.R. and Anderson, B.D.O., Adaptive Frequency Sampling Filters, IEEE Trans. on Circuits and Systems, Vol. 28, No. 6, pp. 524-534, June 1981.

[8] Canaris, J., A VLSI Architecture for the Real-Time Computation of Discrete Trigonometric Transforms, Journal of VLSI Signal Processing, Vol. 5, No. 1, pp. 95-104, Jan. 1993.

[9] Chiu, C.T. and Liu, K.J.R., Real-Time Parallel and Fully Pipelined Two-Dimentional DCT Lattice Structures with Application to HDTV Systems, IEEE Trans. on Circuits and Systems for Video Technology, Vol. 2, No. 1, pp. 25-37, March 1992.

[10] Clark, G.A. and Soderstrand, M.A. and Johnson, T.G., Transform Domain Adaptive Filtering Using a Recursive DFT, Proc. IEEE ISCAS, pp. 1113-1116, June 1985.

[11] Cooley, J.W. and Tukey, J.W., An algorithm for machine computation of complex Fourier series, Math. Comput., Vol. 19, pp. 297-301, 1965.

[12] Dentino, M. and McCool, J. and Widrow, B., Adaptive Filtering in the Frequency Domain, Proceedings of the IEEE, Vol. 66, No. 12, pp. 1658-1659, Dec. 1978.

[13] Duhamel, P., Implementation of Split-Radix FFT Algorithms for Complex, Real, and Real-Symmetric Data, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-34, No. 2, pp. 285-295, April 1986.

[14] Frantzeskakis, E. and Baras, J.S. and Liu, K.J.R., Time-Recursive Computation and Real-Time Parallel Architectures, Part I: Framework, submitted to IEEE Trans. on Signal Processing, July 1993.

[15] Frantzeskakis, E. and Baras, J.S. and Liu, K.J.R., Time-Recursive Computation, Part II: Methodology and Application on QMF Banks and ELT, submitted to IEEE Trans. on Signal Processing, July 1993.

[16] Frantzeskakis, E. and Baras, J.S. and Liu, K.J.R., Time-Recursive Architectures and Wavelet Transform, Proc. IEEE ICASSP, pp. 445-448, 1993.

[17] Frantzeskakis, E. and Baras, J.S. and Liu, K.J.R., Time-Recursive Computation and Real-Time Parallel Architectures, with Application on the Modulated Lapped Transform, Proc. SPIE, International Symposium on Optical Applied Science and Engineering, San Diego, July 1993.

[18] Frantzeskakis, E. and Karathanasis, H., On Computing the 2-D Modulated Lapped Transform in Real-Time, 1993 IEEE Workshop on VLSI Signal Processing, Oct. 1993.

[19] Goertzel, G., An algorithm for the evaluation of finite trigonometric series, Amer. Math. Monthly, Vol. 65, pp. 34-35, 1958.

[20] Harris, F.J., On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform, Proceedings IEEE, Vol. 66, pp. 51-83, Jan. 1978.

[21] Hayes, J.P., Computer Architecture and Organization, McGraw-Hill,Inc., New York, 2nd Edition, 1988.

[22] Hou, H.S., A Fast Recursive Algorithm for Computing the Discrete Cosine Transform, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-35, No. 10, pp. 1455-1461, Oct. 1987.

[23] Yu Hen Hu, CORDIC-Based VLSI Architectures for Digital Signal Processing, IEEE Signal Processing Magazine, Vol. 9, No. 3, pp. 18-35, July 1992.

[24] Jayant, N., Signal Compression: Technology Targets and Research Directions, IEEE Journal on Selected Areas in Communications, Vol. 10, No. 5, pp. 796-818, June 1992.

[25] Jayant, N., Digital Coding of Wideband Audio, IEEE ICASSP 1993, Tutorial No3 (Lecture Notes).

[26] Jaynant, N.S. and Noll, P., Digital Coding of Waveforms, Prentice Hall, Englewood Cliffs, NJ, 1984.

[27] Kailath, T., Linear Systems, Prentice Hall, London, 1980.

[28] Knowles, G., VLSI Architecture for the Discrete Wavelet Transform, Electronics Letters, Vol. 26, No. 15, pp. 1184–1185, July 1990.

[29] Kung, S.Y., Multivariable and Multidimentional Systems: Analysis and Design, PhD Thesis, Stanford University, June 1977.

[30] Lee, B.G., A New Algorithm to Compute the Discrete Cosine Transform, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-32, No. 6, pp. 1243–1245, Dec. 1984.

[31] LeGall, D.J., The MPEG Video Compression Algorithm, Comm. ACM, Vol. 34, No. 4, pp. 46-58, April 1991.

[32] Liu, K.J.R., Novel Parallel Architectures for Short Time Fourier Transform, To appear in IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, 1993.

[33] Liu, K.J.R. and Chiu, C.T., Unified Parallel Lattice Structures for Time-Recursive Discrete Cosine/Sine/Hartley Transforms, IEEE Trans. on Signal Processing, Vol. 41, No. 3, pp. 1357-1377, May 1993.

[34] Liu, K.J.R. and Chiu, C.T. and Kolagolta, R.K. and Jaja, J.F., Optimal Unified Architectures for the Real-Time Computation of Time-Recursive Discrete Sinusoidal Transforms, Submitted to IEEE Trans. on Circuits and Systems for Video Technology, 1992.

[35] Mallat, S.G., A Theory for Multiresolution Signal Decomposition: The Wavelet Representation, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 11, No. 7, pp. 674-693, July 1989.

[36] Malvar, H.S., Lapped Transforms for Efficient Transform/Subband Coding, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-38, No. 6, pp. 969-978, June 1990.

[37] Malvar, H.S. and Staelin, D.H., The LOT: Transform Coding Without Blocking Effects, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-37, No. 4, pp. 553-559, April 1989.

[38] Malvar, H.S., Extended Lapped Transforms: Properties, Applications, and Fast Algorithms, IEEE Trans. on Signal Processing, Vol. 40, N0. 11, pp. 2703-2714, Nov. 1992.

[39] Malvar, H.S., Signal Processing with Lapped Transforms, Artech House,Inc., Boston 1992.

[40] Marchall, F.C.,III and Temes, G.C., Binary Windows for the Discrete Fourier Transform, Proceedings of the IEEE, pp. 1370-1371, Sep. 1975.

[41] Marshall, D.F. and Jenkins, W.K. and Murphy, J.J., The Use of Orthogonal Transforms for Improving Performance of Adaptive Filters, IEEE Trans. on Circuits and Systems, Vol. 36, No. 4, pp. 474–484, April 1989.

[42] Murthy, N.R. and Swamy, M.N.S., On the Computation of Running Discrete Cosine and Sine Transforms, IEEE Trans. on Signal Processing, Vol. 40, No. 6, pp. 1430–1437, June 1992.

[43] Narayan, S.S. and Peterson, A.M. and Narasimha, M.J., Transform Domain LMS Algorithm, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-31, No. 3, pp. 609–615, June 1983.

[44] Nussbaumer, H.J., Fast Fourier Transform and Convolution Algorithms, Springer, Berlin, 1981.

[45] Nussbaumer, H.J., Pseudo QMF Filter Bank, IBM Tech. Disclosure Bulletin, Vol. 24, pp. 3081-3087, Nov. 1981.

[46] Oppenheim, A.V. and Schafer, R.W., Discrete-Time Signal Processing, Prentice Hall,Inc., Englewood Cliffs, NJ, 1989.

[47] Papoulis, A., Signal Analysis, McGraw-Hill,Inc, New York, 1977.

[48] Rioul, O. and Duhamel, P., Fast Algorithms for Discrete and Continuous Wavelet Transforms, IEEE Trans. on Information Theory, Vol. 38, No. 2, pp. 569-586, March 1992.

[49] Rubino, E.M. and Malvar, H.S., Improved Chen-Smith Image Coder, Proc. IEEE ISCAS, 1993.

[50] Edited by Ruskai, M.B. et al., Wavelets and their Applications, Johnes and Barlett Publishers, Inc., Boston, 1992.

[51] Shynk, J.J., Frequency-Domain and Multirate Adaptive Filtering, IEEE Signal Processing Magasine, Vol. 9, No. 1, pp. 14-37, Jan. 1992.

[52] Smith, S.G. and White, S.A., Hardware Approaches to Vector Plane Rotation, Proc. IEEE ICASSP, pp. 2128–2131, 1988.

[53] Ullman, J., Computational Aspect of VLSI, Computer Science Press, Rockville, MD, 1984.

[54] Vaidyanathan, P.P., Multirate Filters and Filter Banks, Prentice Hall, Eglewood Cliffs, NJ, 1993.

[55] Vaidyanathan, P.P. and Doganata, Z., The Role of Lossless Systems in Modern Digital Signal Processing: A Tutorial, IEEE Trans. on Education, Vol. 32, No. 3, pp. 181-197, Aug. 1989.

[56] Vargas, L.F.C. and Malvar, H.S., ELT-Based Wavelet Coding of High-Fidelity Audio Signals, Proc. IEEE ISCAS, 1993.

[57] Vetterli, M. and Naussbaumer, H., Simple FFT and DCT Algorithms with Reduced Number of Operations, Signal Processing, Vol. 6, No. 4, pp. 267-278, Aug. 1984.

[58] Wallace, G.K., Overview of the JPEG Still Picture Compression Algorithm, Comm. ACM, Vol. 34, No. 4, pp. 30-44, April 1991.

[59] Wang, Z., Fast Algorithms for the Discrete W Transform and for the Discrete Fourier Transform, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-32, No. 4, pp. 803-816, Aug. 1984.

[60] White, S.A., Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review, IEEE Signal Processing Magasine, Vol. 6. No. 1, pp. 4-19, July 1989.

[61] Yip, P. and Rao, K.R., On the Shift Property of DCT's and DST's, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-35, No. 3, pp. 404–406, March 1987.

[62] Young, R. and Kingsbury, N., Motion Estimation using Lapped Transforms, Proc. IEEE ICASSP, pp. III 261-264, March 1992.