A CLASS OF NONRECURSIVE SPECTRAL FILTERS AND THEIR VERY LARGE SCALE INTEGRATION IMPLEMENTATION

by

Christopher J. Zarowski

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Master of Science
in
Electrical Engineering

Winnipeg, Manitoba, 1985

A CLASS OF NONRECURSIVE SPECTRAL FILTERS AND THEIR VERY LARGE SCALE INTEGRATION IMPLEMENTATION

BY

CHRISTOPHER J. ZAROWSKI

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Christopher J. Zarowski

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Christopher J. Zarowski

ABSTRACT

This thesis studies some of the properties of a class of nonrecursive filters called discrete Fourier transform (DFT) spectrum filters, and their implementation with special purpose hardware suitable for very large scale integration (VLSI) implementation. The DFT spectrum of a real vector of N components can be filtered with an N×N matrix G_f and the result inverse transformed with the inverse DFT to get the desired signal vector. Alternatively, a different transform matrix T can be applied to the signal and the resulting spectrum filtered by a matrix G_t such that the result is the same as filtering the DFT spectrum of the vector with G_f. This is DFT spectrum filtering. Some of the properties of G_t for T = W (discrete Walsh transform), T = H (discrete Haar transform) and T = T_r (tridiagonal transform - one of the matrix factors of W and H) are described herein. It is found that DFT spectrum filtering using T = W or T = H is more efficient than using the DFT and G_f for N≤ 64, assuming a sequential processor implementation and assuming that G_f is a linear filter. Ordinarily, G_f is complex-diagonal and G_t is real and block-diagonal. The VLSI implementation of the DFT and the DWT using the radix 2, pipeline and linear systolic array (LSA) architectures is considered along with the LSA implementation of G_t. It is found that while the asymptotic area and time complexity of both architectures is essentially the same (to within a constant factor), the radix 2, pipeline structure is superior to the LSA, especially in terms of area. This assumes that we wish to implement the DFT or DWT. The G_t matrices cannot be implemented with the cascade and so require an LSA implementation. Because the LSA needs so much chip area, it is not recommended in general that DFT spectra be filtered with the use of transform T and filter G_t in the context of a special purpose hardware implementation of T or G_t using the cascade and LSA, respectively.

ACKNOWLEDGEMENTS

The author wishes to express his appreciation of the guidance provided by Professors M. Yunik, H. C. Card, and G. O. Martens in the course of this work. The helpful discussions with colleagues R. McCleod, M. Jarmasz, G. Gulak and W. Pries are also greatly appreciated. The financial support of the Natural Sciences and Engineering Research Council is gratefully acknowledged.

TABLE OF CONTENTS

Appendices pages

Chapter I

INTRODUCTION

The purpose of this thesis is to study a class of nonrecursive discrete-time filters and their implementation as digital filters using very large scale integration (VLSI) techniques. This class of filters is referred to in this thesis as the discrete Fourier transform (DFT) spectrum filters. Their purpose is to perform some desired linear filtering operation on the DFT spectrum of a real sampled signal vector (finite number of components). This is done by changing (filtering) the spectrum of the signal vector produced by a transform which is not the DFT, such as the discrete Walsh transform (DWT), discrete Haar transform (DHT), or a new transform to be introduced in this thesis, the tridiagonal transform. This approach to the problem of DFT spectrum filtering is proposed since it may lead to hardware implementations that have certain advantages in the context of VLSI.

The remainder of this chapter provides some cursory background on filtering in general and the special role played by discrete transforms in nonrecursive filtering. Chapter II introduces DFT spectrum filters as discrete-time filters. Some of their properties are examined and three special cases are examined. Chapter III considers the problem of implementing DFT spectrum filters as digital filters using special purpose hardware that is suitable for VLSI implementation. Two competing implementation techniques are evaluated here and these are the radix 2, pipeline FFT method, and the linear systolic array FFT method. The implementation of the fast Walsh transform (FWT) using these two methods is also looked at. As well, the linear systolic array implementation of DFT spectrum filters (the G_t matrices of Chapter II) is considered in this chapter. Chapter IV considers various miscellaneous topics

relating to the DFT spectrum filtering problem and is intended to be a guide to future research efforts.

1.1 FILTERING DEFINED

The definitions presented in this section are not rigorous. The intent is merely to help relate the filtering method of Chapter II to the subject of filtering in general.

Filtering means different things to different people. In its most general sense, therefore, filtering involves the modification of a signal into a more suitable form according to some criteria. The signal itself can take many forms. The signal can vary in time, vary in space, or vary in both time and space, for example. The filtering requirements can be specified in the time (or space) domain, but are usually specified in some frequency domain. The word "frequency" is intended to be more general than its common connotation of the frequency of a sinusoid, which implies that the signal under consideration has been expanded in terms of a basis of sinusoids. More will be said about this later.

Filtering can be performed in both continuous-time or discrete-time. The independent variable of a continuous-time signal may be considered to take on a continuous set of values. Discrete-time signals are defined only at discrete instants of time. When both the amplitude and time variables of a signal are discretized, a digital signal is the result.

The DFT spectrum filters of Chapter II are, strictly speaking, discrete-time filters and not digital filters because the transforms , filters and signal vectors take on values from the set of real or complex numbers. Thus, signal amplitudes are not discretized , although time is discretized. In Chapter III, hardware structures are proposed for some of the filters in Chapter II. This implies a finite word length (FWL) binary implementation, and hence the discretization of signal, transform and filter values. The resulting structures are then true digital filters.

The field of digital signal processing (DSP) can roughly be divided into the study of two main classes of filters : recursive and nonrecursive filters. This is alluded to in Rabiner [1]. Recursive filters are often described by recurrence equations. Recursive filters typically (though not necessarily) operate or are considered to operate on infinite length number sequences. The class of recursive filters is enormous and includes the classical infinite impulse response (IIR) and finite impulse response (FIR) filters, adequately described in Oppenheim and Schafer [2] and Chen [3]. Also included in this class are the wave digital (WD) filters first described by Fettweis [4], the lattice filters of Gray and Markel [5] and the adaptive filters of Widrow et al. [6]. This list is far from exhaustive. On the other hand, nonrecursive filters operate on finite length sequences only. Nonrecursive filtering usually centers around the use of discrete transforms such as the DFT or the DWT. A sampled signal vector is transformed, the resulting spectrum is altered according to some specification(s), and the altered spectrum is inverse transformed to give the desired signal vector. It is to be noted that many nonrecursive methods have recursive implementations and vice versa.

Recursive filtering, unlike nonrecursive filtering, often involves systems which employ feedback. Thus, stability considerations can play a role in recursive filter design. Nonrecursive filtering systems do not use feedback and so stability is never discussed in the context of nonrecursive filtering. Thus, the DFT spectrum filters of Chapter II are inherently stable.

Recursive filters are often used in real-time applications. In real-time DSP ,signal values are fed to the signal processor at a rate determined by the application. The signal processor must complete a given processing operation on the samples it already has before the next sample(s) become available.

Real-time recursive filters must be causal. Nonrecursive filters need not be causal as they are often employed in off-line , non-real-time, applications. Since causality imposes restrictions on the kinds of filtering operations one can perform

upon a signal, non-causal filters can do things that causal filters cannot do. Note, however, that nonrecursive filters can be used in a real-time environment and satisfy causality. As well, recursive filters can be used in a non-real-time environment.

Thus, in general, non-causal, nonrecursive filters are inherently stable and more flexible than causal recursive filters that employ feedback.

1.2 DISCRETE TRANSFORMS

This section briefly reviews discrete transforms such as the discrete Fourier transform (DFT), discrete Walsh transform (DWT), and the discrete Haar transform (DHT). Most of what follows is a mere literature review. Additional background material on these and other transforms is provided as the need arises in the chapters which follow. The DFT and its applications are extensively discussed in Oppenheim and Schafer [2],Chen [3], and Gonzalez and Wintz [7]. The applications of DFT to one-dimensional DSP is considered in [2] and [3] and the applications to digital image processing are considered in [7]. The DWT is considered in [7] as well, but more background on the DWT can be found in Ahmed, Schreiber and Lopresti [8]. The DHT is considered in [8]. It is worth noting that Ahmed et al. [8] proposes a standard terminology for the DWT and DHT.

1.2.1 The DFT

Chen [3] considers the DFT of a signal to be the truncated z-transform of that signal, evaluated on the unit circle in the complex z-plane. Thus, the z-transform of infinite sequence {h(i)}, where i = $0, \pm 1, \pm 2, \pm 3, \ldots$, and $h(i) \in \mathbf{R}$ (set of real numbers), is

$$H(z) = \sum_{i=-\infty}^{\infty} h(i)\, z^{-i} . \tag{1.1}$$

The value of H(z) on the unit circle is

$$H(e^{j\theta}) = \sum_{i=-\infty}^{\infty} h(i)e^{-ij\theta} , \tag{1.2}$$

where $\theta = \omega T$, $j = \sqrt{-1}$,ω is the frequency in radians per second, and T is the sample period in seconds. The DFT according to Chen [3] is the truncated version of (1.2), and its definition is

$$H(k) = \sum_{i=0}^{N-1} h(i)e^{-j\frac{2\pi ik}{N}} \tag{1.3}$$

where N is the number of sample points, or the number of points of the original signal sequence that have been retained.

The inverse z-transform of (1.1) is, according to Chen [3],

$$h(i) = \frac{1}{2\pi j}\oint H(z)\, z^{i-1}\, dz , \tag{1.4}$$

and if it is evaluated on the unit circle,

$$h(i) = \frac{1}{2\pi}\int_0^{2\pi} H(e^{j\theta})e^{ij\theta}\, d\theta . \tag{1.5}$$

However, for (1.3) $H(e^{j\omega T})$ exists only at $\omega = 2\pi k/NT$, $k = 0,1,\ldots,N-1$ so Chen approximates (1.5) by

$$h(i) = \frac{1}{N}\sum_{k=0}^{N-1} H(k)e^{j\frac{2\pi ik}{N}} \tag{1.6}$$

which defines the inverse DFT (IDFT).

Oppenheim and Schafer [2] avoid the approximation problems of Chen's treatment of the DFT simply by considering the expansion of an N-element vector in terms of complex exponentials $\exp\left\{j\frac{2\pi}{N}nk\right\} = e_k(n)$. This produces what Oppenheim and Schafer term the discrete Fourier series (DFS). The DFS can be used to represent discrete-time periodic sequences. Oppenheim and Schafer then show that

the DFS can represent finite duration sequences and this effectively shows that the DFT is the DFS. In so doing, a finite duration sequence is regarded as one period of an infinite duration periodic sequence.

In any case, the approach of Chen and that of Oppenheim and Schafer lead to the transform pair of (1.3) and (1.6), with (1.3) defining the DFT and (1.6) defining the inverse DFT (IDFT). These two expressions can be restated as matrix-by-vector operations. Thus,

$$\bar{H} = F\bar{h} \ , \tag{1.7a}$$

which is equivalent to (1.3), and

$$\bar{h} = F^{-1}\bar{H} \ , \tag{1.7b}$$

which is equivalent to (1.6). Clearly, $F, F^{-1} \in \mathbf{C}^{N\times N}$ (set of complex N×N matrices) and $\bar{h} = [h(0), h(1), \ldots, h(N-1)]^T, \bar{H} = [H(0), H(1), \ldots, H(N-1)]^T$. The superscript "T" means transpose. In this thesis $\bar{h} \in \mathbf{R}^N$ (set of real N-vectors) and $\bar{H} \in \mathbf{C}^N$ (set of complex N-vectors), although $\bar{h}$ can be complex as well. F is referred to as the DFT matrix and F^{-1} is the IDFT matrix.

Because the DFT and the IDFT are of such great importance in DSP, much effort has been expended over the years in finding fast ways of computing (1.7) on sequential processors via software and with special purpose hardware. The fast computation of the DFT is referred to in the literature under the general term of fast Fourier transform (FFT). The implementation of FFT algorithms using special purpose hardware is considered in Chapter III. Software methods for speeding up the computation of the DFT and the IDFT typically involve attempts at finding efficient matrix factorizations of F and F^{-1} in (1.7). For example, the Cooley-Tukey method [9], a decimation-in-time algorithm [2], effectively factorizes F (and F^{-1}) into $\log_2 N$ matrices (N is a positive integer power of two) and reduces the amount of computation from something on the order of N^2 multiplications and additions to something

on the order of $N\log_2 N$ multiplications and additions. In this case the multiplications and additions are complex. This algorithm and many of its variations is described in Oppenheim and Schafer [2]. It is also described in Chen [3] and Gonzalez and Wintz [7], and so will not be covered here.

The unit sample (impulse) function is defined as

$$\delta(n) = \begin{cases} 1 & ,if \quad n=0 \\ 0 & ,if \quad n \neq 0 \end{cases} \tag{1.8}$$

where $n \in \mathbf{Z}$ (set of integers) is the discrete-time variable. Only the class of linear shift-invariant (LSI) discrete- time systems [2,3] is of interest in this thesis. Such a system is completely described by its unit sample (impulse) response sequence {h(n)} which is the LSI system's response to δ(n). Note that the system is assumed not to have any stored energy, and {h(n)} describes only the input/output behaviour of the LSI system. If the input sequence to the system is {u(n)} then the output sequence from such a system is given by

$$y(n) = \sum_{k=-\infty}^{\infty} h(n-k)u(k)\,, \tag{1.9}$$

and this becomes

$$y(n) = \sum_{k=0}^{n} h(n-k)u(k)\,, \; n \geq 0 \tag{1.10}$$

if the system is also causal. The equations in (1.9) and (1.10) are referred to as convolution sums in [2]. Equations (1.9) and (1.10) are also referred to as linear convolutions. Note also that {u(k)} is causal as well.

Convolution of the kind in (1.10) may be done using the DFT. To start, periodic or circular convolution [2,3,7] can be performed by taking the DFT of each of the two sequences to be circularly convolved and multiplying the resulting DFT spectra of the two sequences point-by-point. The resulting sequence is then inverse transformed using the IDFT to produce the final result. Naturally, both sequences

that are being convolved must be of the same length. The linear convolution of (1.10) can be performed with the DFT in much the same way as circular convolution. However, if one sequence is of length N and the other is of length M, then both sequences must be padded with zeros until each is of length N+M-1 at least. We only consider radix 2 DFT in this thesis so N+M-1 must be a power of two. This naturally allows the Cooley-Tukey FFT to be used. The first N+M-1 points of the result of the IDFT operation will be the linear convolution of the two sequences. This method is described in [2], [3], and [7] in considerable detail. Note that the failure to pad with zeros adequately may result in a phenomenon called "wrap-around" [7]. This causes a problem analogous to aliasing. Also note that, compared with the Cooley-Tukey FFT, the direct computation of (1.10) is more efficient for $N \leq 32$, or thereabouts.

The fact that time domain convolution is equivalent to frequency domain multiplication [2] and the fact that causal LSI digital filters are describeable by (1.10) is used in Chapter II to justify a method of selecting suitable prototype Fourier gain matrices (G_f in Chapter II).

1.2.2 The DWT

The discrete Walsh transform (DWT), though less well known than the DFT, has a wide range of applications. The DWT has been used by Cheng and Liu [10] in the solution of difference equations. The Walsh functions from which the DWT is derived have been used by Corrington [11] to solve integral and differential equations approximately, and by Maqusi [12] to expand probability density functions. Pearl [13] has used linear dyadic-invariant (LDI) systems to model LSI systems. The DWT plays a role in LDI systems essentially identical to that played by the DFT in LSI systems. Tadokoro and Higuchi [14,15] have used the DWT to facilitate computation of the DFT coefficients. Their method is faster than that of Cooley-Tukey for transforms of size $N \leq 64$. This method will be considered in more detail in Chapter

II. Chan and Hsiao [16] have used the Walsh functions in the design of optimal control systems. More recently, the DWT has been proposed as a suitable substitute for the DFT in the spectral analysis of electroencephalograms, in Dzwonczyk, Howie and McDonald [17]. In most cases, the principle benefit in using the DWT rather than the DFT is that the DWT is so much easier to compute, since the DWT matrix consists entirely of +1 and -1 entries. Thus, no multiplications are involved.

The DFT can be considered as originating from the Fourier series. Where Fourier series are concerned, periodic functions are expanded in terms of the complete set of sinusoids and cosinusoids. The details of the process of Fourier expanding a function are given extensive treatment in Tolstov [18]. The Walsh functions, like the complete set of sinusoids and cosinusoids in [18], form a complete set themselves. The Walsh functions are defineable in terms of an incomplete set of functions, called the Rademacher functions. The development of the DWT from the Rademacher functions is described in Ahmed et al. [8].

The Walsh functions are rectangular functions which take on values ± 1 throughout the interval over which they are defined ,such as [0,1). The DWT is obtained simply by sampling the functions at regular intervals on [0,1). This will produce an array of ± 1 values. The resulting array is nonsingular and so if it is of dimension N×N, the column vectors making up the array will span the space $\mathbf{R}^N$.

Ahmed et al. [8] describe the three main orderings of the DWT. Different orderings of the DWT arise by rearranging the order of the rows of the DWT matrix. This reordering can be specified by a permutation matrix. The three principle orderings of the DWT are : 1) sequency or Walsh order, 2) dyadic or Paley order, 3) natural or Hadamard order [8]. Other orderings do exist, such as the cal-sal ordering (see Rao et al. [19]).

The importance of ordering is not to be underestimated. In this thesis, the Hadamard ordering is used (see Chapter II). If any of the other two main orderings were selected, the result would be chaos, although all orderings of any given

transform are equivalent to each other.

A fast algorithm for computing the DWT exists that is very much like the Cooley-Tukey FFT algorithm. It is described by Shanks [20]. In fact, this algorithm is identical to the Cooley-Tukey FFT except that it is much simpler because there is no complex arithmetic, only real additions/subtractions, in the Cooley-Tukey fast Walsh transform (FWT). The Cooley-Tukey FWT reduces the number of additions/subtractions involved in the direct computation of the DWT from on the order of N^2 to exactly $N\log_2 N$ additions/subtractions. Again, N is a power of two and is the size of the vector that is to be transformed. A program for the Shanks FWT may be found in Gonzalez and Wintz [7]. Incidentally, a program for Cooley-Tukey FFT can be found in both Chen [3] and Gonzalez and Wintz [7]. The programs are in FORTRAN.

1.2.3 The DHT

The discrete Haar transform (DHT) is also covered by Ahmed et al. [8], and some of its applications are considered by Shore [21]. The DHT, like the DWT , is obtained by sampling at regular intervals a complete set of functions, in this case they are the Haar functions. The DHT as described in Ahmed et al. [8] is not as "nice" as the DWT because the DHT matrix entries consist of numbers like $\pm\sqrt{2}$ in addition to ± 1, 0 and ± 2. For example, the DHT matrix for the case N = 8 is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix},$$

(see Ahmed et al. [8]).

Fast algorithms for the DHT, called fast Haar transforms (FHT), have been proposed by Rejchrt [22] and Ahmed, Natarajan and Rao [23]. The most useful

algorithm from the point of view of the work in Chapter II is that of Ahmed et al. [23]. This fast algorithm is very much like that of the Cooley-Tukey FFT and the FWT of Shanks [20]. Once again, the computational complexity of the DHT is reduced from on the order of N^2 additions/subtractions to on the order of $N\log_2 N$ additions/subtractions. It turns out that the Cooley-Tukey FHT in [23] requires on the order of N nontrivial multiplications. Multiplications by 0, or integral powers of two are regarded as trivial, and all other kinds of multiplication are considered non-trivial.

Chapter II

DFT SPECTRUM FILTERING USING TRANSFORMS OTHER THAN THE DFT

In this chapter the problem of filtering a discrete Fourier transform (DFT) spectrum without computing the DFT coefficients is examined. This is accomplished with the aid of a transform other than the DFT itself. In particular, the discrete Walsh [8],discrete Haar [8,21,23] and a new transform, the tridiagonal transform [24], will be considered. These transforms are used because of their computational simplicity and the ease with which they can be implemented using special purpose hardware. Transforms such as the discrete sine [25] or discrete cosine [26] transforms are not considered since they are computationally much more complex than the Walsh, Haar or tridiagonal transforms. The main concern in this chapter is with the structure of the various filter matrices that arise when transforms other than the DFT are used to filter DFT spectra. The reader should note that most of this chapter is taken from Zarowski and Yunik [27] and from Zarowski, Yunik and Martens [24].

2.1 PROBLEM FORMULATION

Figures 2.1 and 2.2 depict the systems of interest. The vectors $\bar{x}, \bar{x}' \in \mathbf{R}^N$ (set of real N-dimensional vectors) , with $N = 2^q$ ($q \in \mathbf{N}$ and N is the set of natural numbers) are the input and output signal vectors, respectively. Vector $\bar{x}'$ is the filtered form of vector $\bar{x}$. F is the DFT matrix and F^{-1} is the inverse DFT (IDFT) matrix, in Fig. 2.1. In Fig. 2.2 W is the discrete Walsh transform (DWT) matrix and W^{-1} is the inverse DWT (IDWT) matrix. Figure 2.2 can be generalized by replacing W by T (T transform matrix) and W^{-1} by T^{-1} (inverse T transform matrix). W is used instead of

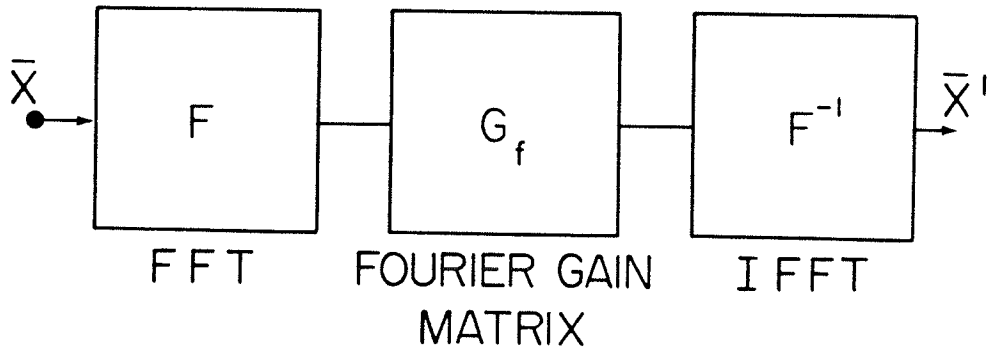

Figure 2.1: Spectral filtering with the FFT.

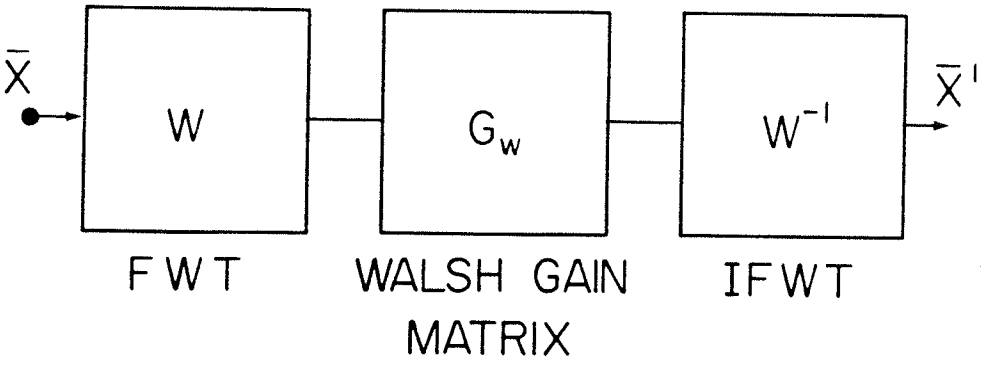

Figure 2.2: Spectral filtering with the FWT.

T in Fig. 2.2 simply for the sake of specificity. All matrices in Fig. 2.1 and Fig. 2.2 are $N \times N$ since $\bar{x}$ and $\bar{x}'$ are N-vectors. On a sequential processor the operations $F\bar{x}$ and $W\bar{x}$ would be carried out using a fast Fourier transform (FFT) [2] and fast Walsh transform (FWT) [7] algorithm, respectively, as indicated in Figs. 2.1 and 2.2. Similarly, $F^{-1}\bar{x}$ is computed using an inverse fast Fourier transform (IFFT) algorithm [2] and $W^{-1}\bar{x}$ is computed using an inverse fast Walsh transform (IFWT) algorithm [7]. Matrix G_f is the Fourier gain (filter) matrix and G_w is the Walsh gain (filter) matrix. More generally, G_t is the T transform gain (filter) matrix.

It should be obvious that the filtering operations being contemplated here occur in the frequency domain (or sequency domain [8] if the Walsh transform is being considered). Thus, $F\bar{x}$ is the frequency spectrum of $\bar{x}$, and the elements of vector $F\bar{x}$ are the DFT spectrum coefficients that must be filtered. Similarly, $W\bar{x}$ is the sequency spectrum of $\bar{x}$, and the elements of vector $W\bar{x}$ contain the DWT spectrum (sequency spectrum) coefficients. In the paragraphs which follow it will be seen that DFT spectrum filtering without the computation of the DFT of signal vector $\bar{x}$ can be accomplished in a general frequency domain by the suitable choice of a filter function in that general frequency domain. The sequency domain is an example of an alternative to the Fourier frequency domain. The Haar and tridiagonal transforms provide other alternatives. It is worth remembering that time domain convolution of two signal sequences corresponds to the frequency domain multiplication of the spectra of the two sequences [2,3,7].

One of the problems dealt with in this chapter is the computation of G_t for some transform T given some G_f, where

$$G_f = diag\,[g_0, g_1, \ldots, g_{N-1}] \tag{2.1}$$

and $g_i = g_{N-i}^*$, i = 1,2,...,N/2-1, and $g_0, g_{N/2} \in \mathbf{R}$ (set of real numbers). The asterisk (*) means complex conjugate. The g_i are called spectral gain (filter) factors. From Figs. 2.1 and 2.2, respectively,

$$\bar{x}' = F^{-1} G_f F \bar{x}, \tag{2.2a}$$

and

$$\bar{x}' = T^{-1} G_t T \bar{x}. \tag{2.2b}$$

Clearly, W has been replaced by T, W^{-1} by T^{-1}, and G_w by G_t in Fig. 2.2 in order to get equation (2.2b). For (2.2a) and (2.2b) to be equal to each other it is necessary and sufficient that

$$T^{-1} G_t T = F^{-1} G_f F, \tag{2.3a}$$

or

$$G_t = T F^{-1} G_f F T^{-1}. \tag{2.3b}$$

Thus, it is now clear that the DFT spectrum of the input signal vector can be filtered by filtering the T transform spectrum of the input signal vector. The special case of T = W will be considered in a later section of this chapter in some detail.

It is worth stating at this time that the conditions on g_i in (2.1) are sufficient to give $\bar{x}' \in \mathbf{R}^N$ if $\bar{x} \in \mathbf{R}^N$ in (2.2a). This fact follows from the properties of the DFT spectra of real signal sample sequences (see [2] and [3]). In addition, these conditions give real and block-diagonal G_w for T = W. This will be proven explicitly later on. In any case it should be clear that if, for some G_f , as in (2.1), then $\bar{x}' \in \mathbf{R}^N$ when $\bar{x} \in \mathbf{R}^N$, and it seems likely as well that $G_t \in \mathbf{R}^{N \times N}$ (set of N×N real matrices) if (2.2b) is forced to equal (2.2a) and $T \in \mathbf{R}^{N \times N}$.

Let $\Gamma_T^{-1} = FT^{-1}$ so then $\Gamma_T = TF^{-1}$. It is obvious that, since F and T are nonsingular, Γ_T is nonsingular. Thus, (2.3b) becomes

$$G_f = \Gamma_T^{-1} G_t \Gamma_T. \tag{2.4}$$

This is clearly a similarity transformation of G_t to the complex diagonal matrix G_f. Thus, the columns of Γ_T must be the eigenvectors of G_t and matrix G_f must contain

the eigenvalues of G_t. There is little doubt that an interesting study could be made of the eigenstructure of F, T, G_f, and G_t but this is beyond the scope of this thesis. It is therefore a topic for future research.

Now, consider T to be Nth order and let T be unitary ($T^{-1} = T^H = (T^*)^T = (T^T)^*$). Thus,

$$T^H T = TT^H = I_N \ , \tag{2.5}$$

where I_N is the Nth order unit matrix. Using the fact that $F = F^T$,$F^{-1} = (1/N)F^*$, $(F^{-1})^H = (1/N)F$, and $F^H = NF^{-1}$

$$G_t^H = TF^{-1}G_f^* FT^{-1}. \tag{2.6}$$

Using (2.6) and (2.3b) yields

$$G_t G_t^H = G_t^H G_t = TF^{-1}|G_f|^2 FT^{-1} \ , \tag{2.7}$$

where $|G_f|^2 = diag\left\{|g_0|^2, |g_1|^2, \ldots, |g_{N-1}|^2\right\}$ and $|g_i|$ is the magnitude of g_i. Thus, $|G_f|^2$ is a real diagonal matrix satisfying the conditions in (2.1), and so G_t cannot be unitary in general. This follows simply because $G_t G_t^H = G_t^H G_t \neq I$ (identity matrix). If G_t is not unitary in general then it cannot generally be orthogonal. In [24] it is shown that ,for the tridiagonal transform, G_t is not diagonal except in special cases.

Now partition T into four N/2th order sub-blocks T_{ij} where i,j $\in$ {1,2} as follows:

$$T = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}. \tag{2.8}$$

The sub-blocks T_{ij} occupy regions of T referred to as quadrants (see Fig. 2.3). Figure 2.3(a) shows the desired structure of G_t. In this section some of the conditions on T_{ij} are found that give the desired structure of Fig. 2.3(a).

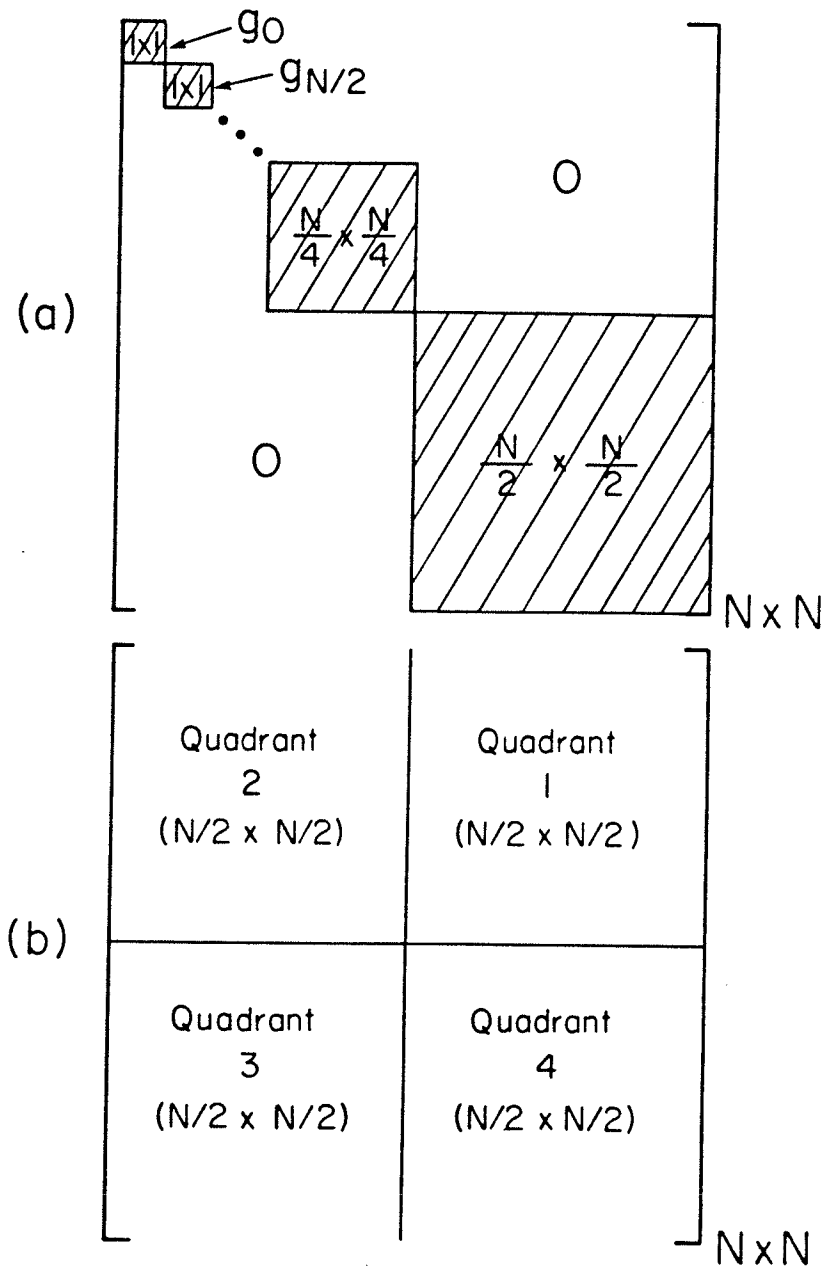

Figure 2.3: (a) Block diagonal structure of G_w. The dense submatrices (blocks) are shaded. (b) Definition of quadrants.

In [27] a matrix, called the A-matrix, is defined as

$$A = F^{-1} G_f F = [a_{nm}] \tag{2.9}$$

where $A \in C^{N \times N}$ (set of N×N complex matrices). Matrix A is the sum of N partial A-matrices $A^{(k)}$, k = 0,1,...,N-1. This is because the typical element of A is

$$a_{nm} = \frac{1}{N} \sum_{i=0}^{N-1} g_i \exp\left\{ j \frac{2\pi i}{N} (n-m) \right\}, \tag{2.10}$$

where $j = \sqrt{-1}$ (see Appendix A). Define

$$A^{(k)} = \left[a_{nm}^{(k)} \right] \tag{2.11}$$

so that

$$a_{nm}^{(k)} = \frac{g_k}{N} \exp\left\{ j \frac{2\pi k}{N} (n-m) \right\}. \tag{2.12}$$

From [27] $A^{(k)} = (g_k / N) \tilde{A}^{(k)}$. In Appendix B it is shown that

$$\tilde{A}^{(k)} = \begin{bmatrix} B & C \\ C & B \end{bmatrix} \tag{2.13}$$

with $C = (-1)^k B$. It is clear that $A^{(k)}$ is Toeplitz because $a_{nm}^{(k)}$ depends upon n-m.

Because $A = \Sigma_{k=0}^{N-1} A^{(k)}$ it follows that

$$G_{t_N}{}^{(k)} = \frac{g_k}{N} T \tilde{A}^{(k)} T^{-1} \tag{2.14}$$

which is the Nth order kth partial T transform gain matrix. Using (2.8) and (2.13) in (2.14) and the fact that $T^{-1} = T^H$, gives

$$G_{t_N}{}^{(k)} = \frac{g_k}{N} \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} B & C \\ C & B \end{bmatrix} \begin{bmatrix} T_{11}^H & T_{21}^H \\ T_{12}^H & T_{22}^H \end{bmatrix}$$

$$= \frac{g_k}{N} \begin{bmatrix} T_{11}BT_{11}^H + T_{12}CT_{11}^H + T_{11}CT_{12}^H + T_{12}BT_{12}^H & T_{11}BT_{21}^H + T_{12}CT_{21}^H + T_{11}CT_{22}^H + T_{12}BT_{22}^H \\ T_{21}BT_{11}^H + T_{22}CT_{11}^H + T_{21}CT_{12}^H + T_{22}BT_{12}^H & T_{21}BT_{21}^H + T_{22}CT_{21}^H + T_{21}CT_{22}^H + T_{22}BT_{22}^H \end{bmatrix}$$

$$= \frac{g_k}{N} \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}. \tag{2.15}$$

To get the desired block-diagonal structure for G_t it is sufficient that

$$M_{21}=M_{12}=0. \tag{2.16}$$

The expressions for M_{12} and M_{21} in (2.15) yield, respectively, upon using $C = (-1)^k B$,

$$[(-1)^k T_{21}+T_{22}]B[(-1)^k T_{11}^H+T_{12}^H]=0,$$

and

$$[(-1)^k T_{11}+T_{12}]B[(-1)^k T_{21}^H+T_{22}^H]=0.$$

These imply that

$$(-1)^k T_{11}+T_{12}=0, \tag{2.17a}$$

or

$$(-1)^k T_{21}+T_{22}=0. \tag{2.17b}$$

It is also required that

$$M_{11}\neq 0 \text{ ,or } M_{22}\neq 0. \tag{2.18}$$

The expressions for M_{11} and M_{22} in (2.15) yield, respectively,

$$[(-1)^k T_{11}+T_{12}]B[(-1)^k T_{11}^H+T_{12}^H]\neq 0,$$

or

$$[(-1)^k T_{21}+T_{22}]B[(-1)^k T_{21}^H+T_{22}^H]\neq 0.$$

These imply that

$$(-1)^k T_{11}+T_{12}\neq 0, \tag{2.19a}$$

or

$$(-1)^k T_{21}+T_{22}\neq 0. \tag{2.19b}$$

Clearly, (2.19) and (2.17) contradict each other. For example, if either one of (2.17a) or (2.17b) holds then only one of (2.19a) or (2.19b) holds, for some k. Thus, $G_{t_N}^{(k)}$ always has three identically zero quadrants for any k. In particular, quadrant-1 and quadrant-3 are always identically zero. The approach used in this thesis to yield block-diagonal G_t is to select T_{ij} so that T is orthogonal. The T_{ij} are then made to satisfy (2.17a) and (2.17b) alternately as k is even or odd. Clearly then (2.19a) and (2.19b) will never be true simultaneously. Quadrants two and four of $G_{t_N}^{(k)}$ will alternate at being zero or nonzero as k varies from even to odd (remembering that k = 0,1,...,N-1). Thus, when all N $G_{t_N}^{(k)}$ matrices are added together, G_t may acquire the structure of Fig. 2.3(a). It is important to note that certain of the possibilities for T will give a dense 2nd quadrant for G_t. The tridiagonal transform is such a T, and G_t's 2nd quadrant is as dense as its 4th quadrant when the tridiagonal transform is used. A quadrant is dense if most of its elements are nonzero. Quadrant-4 is normally dense for transforms and orderings considered in this thesis. Other transform orderings may cause G_t to take the form of Fig. 2.3(a) except that the diagonal blocks of Fig. 2.3(a) are rotated about the secondary diagonal. This causes quadrant-4 to occupy quadrant-2 and thus quadrant-2 will now normally be dense and quadrant-4 will normally be sparse. Clearly, a simple permutation transformation can be applied to put such a G_t back into the desired form of Fig. 2.3(a). If quadrant-2 is not dense then the Nth order G_t will usually contain the N/2th order G_t. This was proven to be true for T = W in [27] and is true for T = H (Haar transform matrix) as well [24]. It is not true for T = $T_{\bar{I}}$ (tridiagonal transform matrix) [24]. These facts will be demonstrated later in this chapter. Quadrant-2 will usually not be dense if , for even k, $(-1)^k T_{11} + T_{12} \neq cI$ (c is a complex scalar constant). Conditions yielding a non-dense 2nd quadrant will be more precisely stated when the special cases for T are considered. Conditions that cause the N/2th order

G_t to be contained in the 2nd quadrant of the Nth order G_t will also be more precisely specified.

2.2 PREVIOUS AND RELATED WORK

Before proceeding with a study of the structure of G_t for the special cases of T = W, T = H, and T = T_r (tridiagonal transform), it is desirable to briefly review the contributions of other researchers to the problem of DFT spectrum filtering without using the DFT and the related problem of finding DFT spectrum coefficients with the aid of transforms other than the DFT.

It turns out that very little work has been done on the problem of filtering DFT spectra without computing the DFT spectrum coefficients. The only publication on this subject known to the author of this thesis is by Kahveci and Hall [28]. Kahveci and Hall empirically examined the problem of filtering the DFT spectrum of a real vector with T = W. They briefly considered the problem of filtering the DFT spectrum of a two-dimensional signal as well. However, they made no effort at all to analyze the structure of G_w (= G_2 in [28]) theoretically. This lead to certain errors on their part in their understanding of G_w's structure as described in their paper. Thus, this thesis represents an attempt to correct and extend the work of Kahveci and Hall.

While very little work has been done to date on the problem of filtering DFT spectra without computing the DFT, much more has been done on the related problem of computing DFT spectral coefficients by the use of transforms other than the DFT. Efforts in this area appear to center around the basic problem of computing discrete Fourier series (essentially the same as DFT [2]) coefficients using the Walsh transform. Early efforts are due to Siemens and Kitai [29] and Blachman [30].

More recently, Tadokoro and Higuchi [14-15] developed an improved version of the method due to Siemens and Kitai [29]. Tadokoro and Higuchi compute a_k and b_k in

$$f(i)=a_0+\sum_{k=1}^{(N/2)-1}(a_k \cos k\theta_i + b_k \sin k\theta_i)+b_{N/2}\sin(N/2)\theta_i \qquad (2.20)$$

where N = 2^q ($q \in N$), f(i) (i=1,2,...,N) are the N samples of the continuous-time signal f(t) (t is time), a_k and b_k are Fourier coefficients, and

$$\theta_i = \frac{2\pi i}{N} - \frac{\pi}{N}.$$

Equation (2.20) is actually the inverse DFT. This fact is demonstrated in Appendix C, but was not shown by Tadokoro and Higuchi [14-15].

The coefficients a_k and b_k are computed using the Walsh transform as follows. Let $\bar{x}$ be the signal vector of N components and let $\bar{X}$ be the Fourier coefficient vector, where

$$\bar{X} = \left[a_0\ b_1\ a_1\ b_2\ a_2\ \cdots\ b_{\frac{N}{2}-1}\ a_{\frac{N}{2}-1}\ b_{\frac{N}{2}}\right]^T.$$

Let C be a conversion factor matrix with

$$C = FW^{-1} \qquad (2.21)$$

so

$$\bar{X} = CW\bar{x}. \qquad (2.22)$$

Thus, $\bar{X}$ is computed by first determining the DWT of $\bar{x}$ (using the FWT [7]) and then applying the conversion matrix C to the resulting vector $W\bar{x}$. Tadokoro and Higuchi derive expressions for the elements of C and they used the Walsh or sequency ordered W matrix [8] in their derivations. Do not confuse the use of C in (2.21) and (2.22) with the use of C in (2.13).

The use of the Walsh transform and a conversion matrix to compute the Fourier coefficients is motivated by the fact that this operation is accompanied by an overall reduction in the amount of computation for $N \leq 64$ [14]. Computation is reduced with respect to the use of the Cooley-Tukey FFT algorithm (described in

[2,3,7]). In particular, the number of multiplications is reduced. This can represent a dramatic increase in the processing speed on a sequential processor. This is especially true if the processor does not possess a hardware multiplier. The method of Tadokoro and Higuchi has the added advantage that not all of the Fourier coefficients need be calculated. The Cooley- Tukey FFT algorithm, and indeed most FFT algorithms, require that all DFT coefficients be computed.

It is important to realize that there are other ways of computing the DFT spectrum, and therefore of convolving two time sequences. Many are more efficient computationally than either the method of Cooley- Tukey or of Tadokoro and Higuchi. For example, the fastest method of computing the DFT is Winograd's algorithm [31]. However, fast algorithms such as Winograd's algorithm tend to be much more complex in terms of implementation than algorithms such as the Cooley-Tukey algorithm or the Tadokoro and Higuchi algorithm. It is a fact that computationally suboptimal but simple algorithms may be more cost-effective than computationally optimal but complex algorithms. Thus, simple methods such as that of Tadokoro and Higuchi continue to be of interest.

The more general problem of using the Walsh transform to compute the coefficients of transforms other than the DFT has been considered by Jones,Hein and Knauer [32], Venkatraman,Kanchan,Rao and Srinivasan [33], and Kwak,Srinivasan and Rao [34]. Jones et al. [32] show that any transform in the class of even-odd transforms (EOT) can be expressed in terms of any other transform in that class via a conversion factor matrix. The discrete Walsh, sine and cosine transforms are examples of this class of transforms. Jones et al. mainly considered the discrete cosine transform. Venkatraman et al. [33] consider a wide variety of discrete transforms such as the C-matrix transform (CMT), which is an approximation of the discrete cosine transform (DCT). They also consider the discrete sine transform (DST), and the discrete Legendre transform (DLT) as well as others. Their study is short and completely empirical. It deals with transforms of low order

($N \leq 32$). Computation of the CMT and DCT using the discrete Walsh transform is given separate consideration in Kwak et al. [34], and again this is an empirical study of transforms of small order ($N \leq 32$). Once again the motivation for using one transform (here the DWT) for computing another transform is to save on computation.

2.3 SPECIAL CASES FOR TRANSFORM T

It is now appropriate to consider three special cases for transform T. These are T = W (discrete Walsh transform), T = H (discrete Haar transform) and $T = T_r$ (tridiagonal transform). It will be seen that these transforms give rise to filter matrices with a very regular structure. In some cases, the resulting structures are candidates for very large scale integration (VLSI) implementation. This subject will be explored in Chapter III.

2.3.1 Walsh Transform

For T = W equation (2.3b) can be rewritten as

$$G_w = \frac{1}{N} W F^{-1} G_f F W, \tag{2.23}$$

where $W^{-1} = (1/N)W$. It is important to note that the Hadamard (natural) ordered discrete Walsh transform [8] is being used in this thesis. If $W = [w_{nm}]$ then

$$w_{nm} = (-1)^{\sum_{k=0}^{q-1} b_k(n) b_k(m)}, \tag{2.24}$$

where $b_k(n)$ is the kth bit of the binary representation of n and modulo-2 arithmetic is used in the exponent of (2.24). Remember that $N=2^q$. This expression for a typical element of W is taken from Gonzalez and Wintz [7]. Since $A = F^{-1}G_f F$ and $A = \Sigma_{k=0}^{N-1} A^{(k)}$ it is possible to write

$$G_w = \frac{1}{N} WAW = \frac{1}{N} W \left(\sum_{k=0}^{N-1} A^{(k)} \right) W = \sum_{k=0}^{N-1} \frac{1}{N} W A^{(k)} W = \sum_{k=0}^{N-1} G_w^{(k)}, \tag{2.25}$$

where

$$G_w^{(k)}=\left[\gamma_{nm}^{(k)}\right]=\frac{1}{N}WA^{(k)}W \tag{2.26}$$

is the kth partial Walsh gain (filter) matrix. In Appendix D it is shown that the typical element of $G_w^{(k)}$, namely $\gamma_{nm}^{(k)}$, is given by

$$\gamma_{nm}^{(k)}=\frac{1}{N^2}g_k\sum_{r=0}^{N-1}\sum_{p=0}^{N-1}(-1)^{\sum_{l=0}^{q-1}[b_l(n)b_l(r)+b_l(p)b_l(m)]}\exp\left(j\frac{2\pi k}{N}(r-p)\right). \tag{2.27}$$

This expression can be interpreted as the typical element of the two-dimensional discrete Walsh transform of the kth partial A-matrix (see equation 3.5-32 in [7]). This fact is significant since a simple and fast algorithm for the computation of two-dimensional discrete Walsh transforms is available and is fully described in Gonzalez and Wintz [7]. Rewrite the right most equation in (2.26) as

$$\frac{1}{N}WA^{(k)}W=\frac{1}{N}\left\{W\left[WA^{(k)}\right]^T\right\}^T. \tag{2.28}$$

From (2.28) it is possible to see that $G_w^{(k)}$ can be computed in a computationally efficient manner by taking the FWT of each column of $A^{(k)}$ and saving the resultant matrix $WA^{(k)}$. Next, the FWT of the rows of the intermediate result $WA^{(k)}$ are taken and then this result is scaled by factor 1/N and transposed. This yields the final result. From this discussion it should be clear that the Hadamard ordered DWT is self-inverse and symmetric, except for the scale factor 1/N.

Thus, an efficient method for the computation of G_w off-line exists. However, the method is too slow for on-line signal processing applications. In practice, the gain matrix G_w would be computed off-line and saved for use in an on-line signal processing application. This statement is true for G_t generally. The conditions under which this approach is likely to be worthwhile will be considered in the sequel. The preceding off-line procedure is taken from Zarowski and Yunik [27].

It has been stated that the condition $g_i = g^*_{N-i}$ (i = 1,2,...,(N/2)-1) and $g_0, g_{\frac{N}{2}} \in \mathbf{R}$ will yield a real G_w and a conjecture was proposed which implies that this is generally true if $T \in \mathbf{R}^{N \times N}$, and (2.2b) is forced to equal (2.2a). This will now be proven explicitly for T = W (originally proven in [27]). From (2.12),

$$a_{nm}^{(N-k)} = \frac{g_{N-k}}{N} \exp\left\{ j \frac{2\pi(N-k)}{N}(n-m) \right\}$$

$$= \frac{g_{N-k}}{N} \exp\left\{ -j \frac{2\pi k}{N}(n-m) \right\}. \tag{2.29}$$

Thus, (2.29) and (2.12) are related as

$$a_{nm}^{(k)} = a_{nm}^{*(N-k)} \tag{2.30}$$

provided $g_k = g^*_{N-k}$ for k = 1,2,...,(N/2)-1. This implies that $A^{(k)} = A^{*(N-k)}$. It is necessary that g_0 and $g_{\frac{N}{2}}$ be real so that $G_w^{(0)}$ and $G_w^{(N/2)}$, are real.

From (2.26),

$$G_w^{*(N-k)} = \frac{1}{N} W A^{*(N-k)} W. \tag{2.31}$$

Since $G_w^{(N-k)} = [v_{nm} + u_{nm} j]$ then $G_w^{*(N-k)} = [v_{nm} - u_{nm} j] = G_w^{(k)}$ so $G_w^{(k)} + G_w^{(N-k)} = 2[v_{nm}]$. Thus, when the partial Walsh gain matrices, $G_w^{(k)}$ and $G_w^{(N-k)}$ for k = 1,2,...,(N/2)-1, are added together, a real matrix is produced. Adding $G_w^{(0)}$ and $G_w^{(N/2)}$ to this matrix yields a real G_w matrix. It is important that $G_w^{(0)}$ and $G_w^{(N/2)}$ be real themselves because of the structure of G_w. This will become evident in the remainder of this subsection.

It is now desirable to prove that G_w has the structure depicted in Fig. 2.3(a), that is, G_w is block-diagonal. This proof is taken from [27]. Once again it is important to note that the Hadamard ordered DWT is used here. The use of another ordering will yield a structure different from, but equivalent to, that depicted in Fig.

2.3(a). Equivalent structures are related by simple permutation transformations. The Hadamard ordered DWT [7] is generated by the order recursion

$$W_{2N}=\begin{bmatrix} W_N & W_N \\ W_N & -W_N \end{bmatrix}, \tag{2.32}$$

where W_N is the Nth order W ,W_{2N} is the 2Nth order W, and $W_1 = +1$. It is convenient to rewrite (2.26) as

$$G_{w_{2N}}{}^{(k)}=\frac{1}{2N}W_{2N}A_{2N}^{(k)}W_{2N}, \tag{2.33}$$

where $A_{2N}^{(k)}$ is the 2Nth order kth partial A-matrix. Using (2.13) and the fact that $A^{(k)}=(g_k/N)\tilde{A}^{(k)}$,

$$A_{2N}^{(k)}=\frac{g_k}{2N}\begin{bmatrix} B & C \\ C & B \end{bmatrix}=\frac{g_k}{2N}\tilde{A}_{2N}^{(k)}. \tag{2.34}$$

Substituting (2.32) and (2.34) into (2.33) gives

$$\begin{aligned} G_{w_{2N}}{}^{(k)} &= \frac{g_k}{4N^2}\begin{bmatrix} W_N & W_N \\ W_N & -W_N \end{bmatrix}\begin{bmatrix} B & C \\ C & B \end{bmatrix}\begin{bmatrix} W_N & W_N \\ W_N & -W_N \end{bmatrix} \\ &= \frac{g_k}{4N^2}\begin{bmatrix} 2W_N(B+C)W_N & 0 \\ 0 & 2W_N(B-C)W_N \end{bmatrix} \\ &= \frac{g_k}{2N^2}\begin{bmatrix} (1+(-1)^k)W_N B W_N & 0 \\ 0 & (1-(-1)^k)W_N B W_N \end{bmatrix}, \end{aligned} \tag{2.35}$$

where $C = (-1)^k B$ has been used. From (2.35) it is easy to see that quadrant-1 and quadrant-3 of $G_{w_{2N}}{}^{(k)}$ are always identically zero. As well, quadrant-2 is identically zero when k is odd and quadrant-4 is identically zero when k is even (k = 0,1,2,...,2N-1 here).

The set $G=\{g_i \mid i=0,1,...,N-1\}$ represents samples of some desired filter function $H(e^{j\theta})$ and $j=\sqrt{-1}$. $H(e^{j\theta})$ is sampled at N points corresponding to

$$\theta_i = \frac{2\pi i}{N}, \tag{2.36}$$

so $g_i = H(e^{j\theta_i})$. The function $H(e^{j\theta})$ is the system function H(z) [2,3], a rational function of z, evaluated on the unit-circle in the complex z-plane. If we start with N samples of $H(e^{j\theta})$ and then double the number to 2N, the N new samples will fall in between the N old samples (as in a perfect shuffle of a deck of cards). The new samples of g_i will correspond to odd values of i. Thus, these new samples will affect the value of $G_{w_{2N}}^{(k)}$ only in quadrant-4. Similarly, the values of g_i for even i will only affect quadrant-2 of $G_{w_{2N}}^{(k)}$. However, g_i for i even is one of the N old samples corresponding to an Nth order G_f filter matrix and thus quadrant-2 of $G_{w_{2N}}$ must contain G_{w_N}. This naturally leads to the structure of Fig. 2.3(a) for G_w.

Since quadrant-2 of an Nth order G_w contains the N/2th order G_w, and since g_i with i odd always affects only the fourth quadrant, it is possible to deduce the diagonal block that any particular g_i will affect. The block structure of G_w for N = 32 is shown in Fig. 2.4 and Table 2.1 contains a list of the diagonal blocks in Fig. 2.4 and the g_i values that affect them. Note that block B4 contains all g_i with odd i for N = 32, block B3 contains all g_i with odd i for N = 16, block B2 contains all g_i with odd i for N = 8, and so on. From this it may be seen that g_0 only affects element (0,0) of matrix G_w and $g_{N/2}$ only affects element (1,1) of G_w. No other g_i affects these two elements of G_w. Thus, $G_w^{(0)}$ and $G_w^{(N/2)}$ must be real since, if they were complex, adding them to the remaining $G_w^{(k)}$ would not give a real G_w because elements (0,0) and (1,1) would be complex. This implies that $g_0, g_{N/2} \in \mathbf{R}$ is required. The diagrams of Fig. 2.4 and Fig. 2.3(a) both illustrate this.

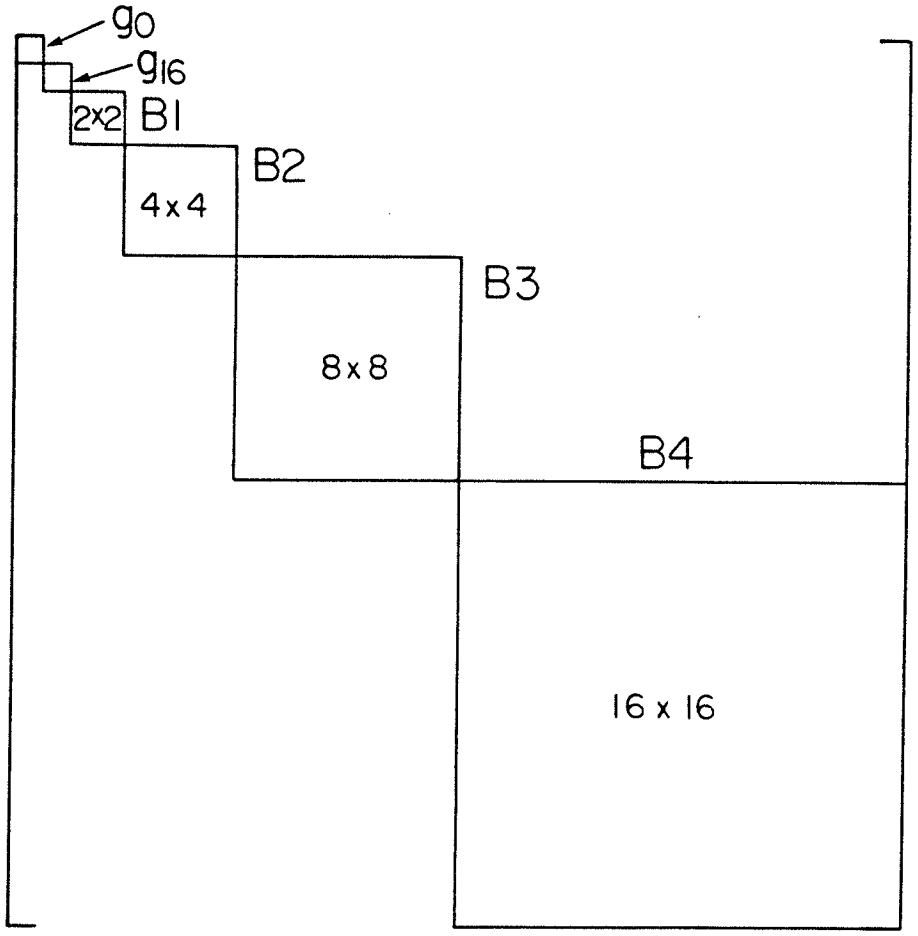

Figure 2.4: The block structure of G_w for N = 32 showing diagonal blocks B1, B2, B3 and B4 (see Table 2.1).

Block	g_i that the block is a function of
B1	g_8, g_{24}
B2	$g_4, g_{12}, g_{20}, g_{28}$
B3	$g_2, g_6, g_{10}, g_{14}, g_{18}, g_{22}, g_{26}, g_{30}$
B4	$g_1, g_3, g_5, \cdots, g_{13}, g_{15}, \cdots, g_{27}, g_{29}, g_{31}$

Table 2.1: The diagonal blocks of G_w for N = 32 showing which g_i values they are a function of (see Fig. 2.4).

The method of generating set G described above needs some justification. A digression to explain this method is therefore in order. To begin, it can be shown that a linear time-invariant [3] (linear shift-invariant [2]) and causal filter that is relaxed at discrete-time instant n=0 can be described by

$$y(n) = \sum_{m=0}^{n} h(n-m)u(m), \quad n=0,1,2,... \tag{2.37}$$

where {y(n)} is the filter output sequence and {u(n)} is the input signal sequence. The sequence {h(n)} is the impulse response sequence of the filter and may be of infinite duration. Equation (2.37) is the linear discrete-time convolution of {u(n)} with {h(n)} to get {y(n)}. Taking the z-transform of (2.37) yields

$$Y(z)=H(z)U(z), \tag{2.38}$$

as shown in [3], where Y(z), H(z) and U(z) are the z-transforms of the sequences {y(n)}, {h(n)} and {u(n)} , respectively. Clearly, H(z) is the system function of the filter characterized by the impulse response sequence {h(n)}. Since the filter is causal and linear time-invariant, H(z) is a proper rational function [3]. A rational function is said to be proper if the degree of the denominator is equal to or greater than the degree of the numerator.

To produce a discrete-time sequence {u(n)}, a continuous-time signal u(t) is sampled every τ seconds (sampling period). If the signal is band-limited and has a bandwidth of f_c then $\tau < 1/(2f_c)$ is required to prevent aliasing [2,3,7]. It is shown in [3] that the steady-state response of filter H(z) to an appropriately sampled sinusoidal sequence of frequency ω radians per second is fully characterized by $H(e^{j\omega\tau})=A(\omega)e^{j\phi(\omega)}$ where $A(\omega)$ is the amplitude response of the filter and $\phi(\omega)$ is the phase response. The sinusoid will be attenuated (or amplified) by amount $A(\omega)$ and phase-shifted by amount $\phi(\omega)$. Filters are ordinarily designed according to how they affect a sinusoidal input sequence. Thus, $H(e^{j\omega\tau})=H(e^{j\theta})|_{\theta=\omega\tau}$ is a suitable definition for the spectrum of a signal sequence or the characteristics of a filter [3]. It is important to note that $H(e^{j\theta})$, when sampled according to (2.36), will satisfy the conditions on g_i as stated in (2.1). This is clear from reading [2,3,7]. Thus, at least H(z) bears consideration as a suitable prototype for G_f and hence for G_w (or G_t in general).

There is one more issue surounding the use of H(z) as a prototype filter for G_f. It is true that $H(e^{j\theta})$ produces a continuous but periodic spectrum. The question is: Is it legitimate to sample $H(e^{j\theta})$ in order to get G_f ? The answer is yes because the DFT spectrum of $\bar{x}$ exists only at discrete frequencies and $\bar{x}$ has only a finite number of harmonics. Vector $\bar{x}$ is obtained from a discrete- time signal sequence {x(n)}, truncated after N samples, and the discrete-time signal sequence is obtained by sampling a continuous- time signal , x(t), fast enough ($\tau < 1/(2f_c)$). Thus, $\bar{x}$ should be a suitable approximate representation of both the discrete- time and continuous-time sequences. If $H(e^{j\theta})$ represents the desired filter operation that we wish to perform on {x(n)} then clearly, to filter $\bar{x}$, those harmonics possessed by $\bar{x}$ and affected by $H(e^{j\theta})$ are all that is needed in order to filter $\bar{x}$. It is obvious that these harmonics occur at θ_i in (2.36). Thus, $g_i=H(e^{j\theta_i})$. Therefore, the method described above for obtaining G_f is legitimate.

It is to be noted that W satisfies the requirements associated with (2.17) and (2.19) described at the end of section 2.1. W is clearly orthogonal, except for a scale

factor. Equations (2.17a) and (2.17b) alternate at being zero or nonzero as k varies from even to odd. It has been suggested that a necessary condition for quadrant-2 not to be dense is $(-1)^k T_{11}+T_{12} \neq cI$. Clearly, this condition holds for T = W. Since B is in all quadrants of $\tilde{A}^{(k)}$ (see (2.13)), it has essentially the same structure as $\tilde{A}^{(k)}$ except that if $\tilde{A}^{(k)}$ is of order 2N then B is effectively of order N since only even k affect quadrant-2 of $G_{w_{2N}}{}^{(k)}$. Thus, $W_N B W_N$ can be expected to possess the same structure as (2.35),assuming even k. This of course has proven to be true from the preceding discussions. Thus, to avoid a dense second quadrant (in (2.35)), it should have a form such as $(1+(-1)^k)cW_N B W_N$, or more generally $(1+(-1)^k)cT_N B T_N{}^H$ for 2Nth order T.

In Zarowski, Yunik and Martens [24] it is shown that the diagonal blocks of G_w have the form

$$\begin{bmatrix} X & Y \\ -Y & X \end{bmatrix}. \tag{2.39}$$

Also, the $G_w^{(k)}$ blocks have a structure like that of (2.39). This is an interesting property since it is known [35] that an isomorphism exists between the group of nonzero matrices of the form

$$\begin{bmatrix} a & b \\ -b & a \end{bmatrix}, \tag{2.40}$$

under matrix multiplication ($a, b \in \mathbf{R}$, $a \neq 0$, and $b \neq 0$ simultaneously), and the group of nonzero complex numbers under ordinary complex multiplication. Clearly, X and Y are square submatrices in (2.39). This structure may have practical significance in that it may signal the presence of an efficient factorization of matrix G_w, or of matrix G_t in general. This possibility is under investigation (see Chapter IV). It has been shown that G_t is not unitary. Good [36] and Andrews and Caspari [37] considered the factorization of certain kinds of unitary matrices, such as W and F. Since G_t is not unitary even when T itself is unitary, it is possible to conclude that the method of Good and of Andrews and Caspari is not useful here.

Let

$$B = \begin{bmatrix} Q & P \\ R & Q \end{bmatrix}, \tag{2.41}$$

and

$$W_N = \begin{bmatrix} W_{N/2} & W_{N/2} \\ W_{N/2} & -W_{N/2} \end{bmatrix}. \tag{2.42}$$

This allows the submatrix $W_N B W_N$ in (2.35) to be written as

$$W_N B W_N = \begin{bmatrix} W_{N/2}(2Q+P+R)W_{N/2} & W_{N/2}(R-P)W_{N/2} \\ -W_{N/2}(R-P)W_{N/2} & W_{N/2}(2Q-P-R)W_{N/2} \end{bmatrix}, \tag{2.43}$$

where (2.41) and (2.42) have been used. Comparing (2.43) with (2.39) yields

$$Y = W_{N/2}(R-P)W_{N/2}. \tag{2.44}$$

It is now necessary to show that R + P = 0 for odd k (k = 0,1,...,2N-1). If R + P = 0 for odd k this will give the diagonal blocks the structure of (2.39).

B is Hermitian (see Appendix B) so $P = R^H$. Let $P = [p_{nm}]$, $R = [r_{nm}]$ so $r_{nm} = p^*_{mn}$ and if $p_{nm} = \alpha_{nm} + \beta_{nm} j$ then $p^*_{mn} = \alpha_{mn} - \beta_{mn} j = r_{nm}$. It is possible to write $B = [b^{(k)}_{nm}]$ and $b^{(k)}_{nm}$ is

$$b^{(k)}_{nm} = \exp\left\{ j\frac{\pi k}{N}(n-m) \right\}, \tag{2.45}$$

(2Nth order $G_w^{(k)}$). Considering quadrant-1 of B in quadrant-2 of (2.35) gives

$$b^{(k)}_{nm} = \exp\left\{ j\frac{\pi k}{N}(n-r-\frac{N}{2}) \right\} \tag{2.46}$$

where m = r + (N/2) and n,r = 0,1,2,...,(N/2)-1. Thus, $p_{nm} = \exp\left\{ j\frac{\pi k}{N}(n-m-\frac{N}{2}) \right\}$ and n,m = 0,1,2,...,(N/2)-1 (changing the indexing notation somewhat), which yields for odd k,

$$\alpha_{nm} = cos\left[\frac{k\pi}{N}(n-m-\frac{N}{2})\right] = sin\left[\frac{k\pi}{2}\right] sin\left[\frac{k\pi}{N}(n-m)\right], \tag{2.47a}$$

$$\beta_{nm} = sin\left[\frac{k\pi}{N}(n-m-\frac{N}{2})\right] = -sin\left[\frac{k\pi}{2}\right] cos\left[\frac{k\pi}{N}(n-m)\right]. \tag{2.47b}$$

Clearly, $\beta_{nm} = \beta_{mn}$ and $\alpha_{nm} = -\alpha_{mn}$. Thus,

$$\begin{aligned} R+P &= [r_{nm}+p_{nm}] \\ &= [p^{*}_{mn}+p_{nm}] \\ &= [\alpha_{mn}-\beta_{mn}j+\alpha_{nm}+\beta_{nm}j] \\ &= 0 \end{aligned} \tag{2.48}$$

We can be more specific about the structure of $W_N B W_N$ in (2.43) above. Start by using the fact that

$$\begin{aligned} R-P &= [\alpha_{mn}-\beta_{mn}j-\alpha_{nm}-\beta_{nm}j] \\ &= -2[\alpha_{nm}+\beta_{nm}j] \\ &= -\left[\exp\left\{-j\frac{\pi k}{2}\right\}\right]\left[2\exp\left\{j\frac{\pi k}{N}(n-m)\right\}\right] \\ &= \pm j\left[2exp\left\{j\frac{\pi k}{N}(n-m)\right\}\right], \end{aligned} \tag{2.49}$$

where the expression for p_{nm} and odd k are used. Thus, the structure of a diagonal block in $G_{w_{2N}}{}^{(k)}$ can be more precisely stated than in (2.39) as

$$W_N B^{(k)} W_N = \begin{bmatrix} X^{(k)} & \pm jX^{(k)} \\ \mp jX^{(k)} & X^{(k)} \end{bmatrix} \tag{2.50}$$

because $Q=[q_{nm}]$ and $q_{nm}=\exp\left\{j\frac{\pi k}{N}(n-m)\right\}$ and B has been replaced by $B^{(k)}$, and X by $X^{(k)}$ to emphasize their dependence upon k.

It has been shown, by running a modification of program GWGEN (see Appendix G), that if g_k is real then $G_w^{(k)}$ is composed of elements which are strictly real or strictly imaginary (N $\leq$ 128). Let D_1 be Hermitian-Toeplitz and D_2 be skew-Hermitian-Toeplitz, and both matrices are 2 × 2. It is easily shown that

$$W_2 D_1 W_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha_1 & \alpha_2+\beta_2 j \\ \alpha_2-\beta_2 j & \alpha_1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$= 2\begin{bmatrix} \alpha_1+\alpha_2 & -\beta_2 j \\ \beta_2 j & \alpha_1-\alpha_2 \end{bmatrix}, \tag{2.51}$$

$$W_2 D_2 W_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \beta_1 j & \alpha_2+\beta_2 j \\ -\alpha_2+\beta_2 j & \beta_1 j \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$= 2\begin{bmatrix} (\beta_1+\beta_2)j & -\alpha_2 \\ \alpha_2 & (\beta_1-\beta_2)j \end{bmatrix}. \tag{2.52}$$

The proof that the elements of $G_w^{(k)}$ are strictly real or strictly imaginary (g_k is real here) appears to rest on the truth of (2.51) and (2.52). The full proof will not be presented here.

Figure 2.5 depicts more of the details of the structure of G_w for N=32. The '0' characters indicate the elements of G_w that are zero. Characters 'X' and 'Z' indicate elements which are generally nonzero. If it is assumed that $g_i \in \mathbf{R}$ for all i and $g_i = g_{N-i}$ for i = 1,2,...,(N/2)-1 then it has been observed that the 'Z' characters of Fig. 2.5 all go to zero and so the 'Z' characters in Fig. 2.5 can be replaced by '0' characters. Because a typical element of $G_w^{(k)}$ is strictly real or strictly imaginary when $g_k \in \mathbf{R}$, the structure of (2.50) suggests that half of all the elements in the diagonal blocks of G_w might be zero. This is because when all $G_w^{(k)}$ are added together the imaginary parts of $G_w^{(k)}$ must vanish. This disappearance is assured by the fact that $G_w^{(k)} = G_w^{*(N-k)}$. However, it must still be shown that exactly half of the elements in a block are real and half are imaginary. Furthermore, to prove that the structure of Fig. 2.5 holds for all N, it must be shown that the 'Z' characters correspond to

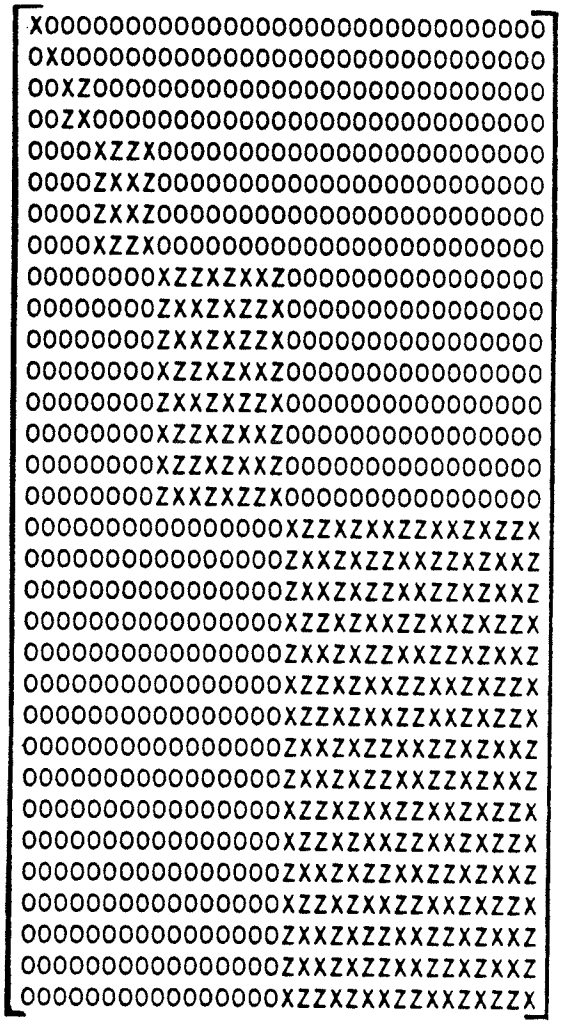

Figure 2.5: The structure of G_W for N = 32 showing more of the detail of the structure of the diagonal blocks.

imaginary numbers. These proofs have yet to be worked out.

Because of the structure of Fig. 2.3(a), any T transform in general which yields this structure will produce a G_t that causes the system of Fig. 2.2 to require A_t additions/subtractions and M_t multiplications to process vector $\bar{x}$ into vector $\bar{x}'$. This compares with the A_f additions/subtractions and M_f multiplications to process vector $\bar{x}$ into vector $\bar{x}'$ using the system of Fig. 2.1 (Cooley-Tukey FFT [2,3,7]). By additions , subtractions and multiplications are meant real additions, subtractions and multiplications. From [3] it is known that the number of complex additions for the Cooley-Tukey type FFT algorithm is, to a close approximation, $N \log_2 N$ and the number of complex multiplications is $\frac{1}{2}N \log_2 N$. Thus, FFT requires $2N \log_2 N$ real additions and $2N \log_2 N$ real multiplications, and IFFT has the same computational requirement. There are 4N real multiplications associated with premultiplying the complex vector $F\bar{x}$ by the complex diagonal matrix G_f. There are 2N real additions/subtractions associated with this operation as well. Thus, the amount of computation involved in the use of the spectral filtering system of Fig. 2.1 is

$$A_f = 4N \log_2 N + 2N , \tag{2.53a}$$

and

$$M_f = 4N \log_2 N + 4N . \tag{2.53b}$$

The FWT and IFWT both require $N\log_2 N$ additions/subtractions. Using this fact and the structure of G_w gives

$$A_w \le 2N\log_2 N + \left[2+12+\cdots+\frac{N}{8}\left(\frac{N}{8}-1\right)+\frac{N}{4}\left(\frac{N}{4}-1\right)+\frac{N}{2}\left(\frac{N}{2}-1\right)\right] , \tag{2.54a}$$

and

$$M_w \le \left[2^2+4^2+\ldots.+\left(\frac{N}{8}\right)^2+\left(\frac{N}{4}\right)^2+\left(\frac{N}{2}\right)^2\right]+2 , \tag{2.54b}$$

where $N \geq 4$ and subscript "w" indicates that $A_t = A_w$, and $M_t = M_w$. Inequality is used in (2.54a) and (2.54b) because these expressions provide acceptable bounds on the computation for both the Haar and Walsh transforms. As well, for $g_i \in \mathbf{R}$ for all i, the computation associated with operation $G_w \bar{y}$ ($\bar{y} = W\bar{x}$) appears to be halved ('Z' characters go to '0' characters in Fig. 2.5). Thus, for $g_i \in \mathbf{R}$, i=0,1,...,N-1,

$$A_w \leq 2N\log_2 N + \tfrac{1}{2}\left[2+12+\cdots+\frac{N}{8}\left(\frac{N}{8}-1\right)+\frac{N}{4}\left(\frac{N}{4}-1\right)+\frac{N}{2}\left(\frac{N}{2}-1\right)\right], \qquad (2.55a)$$

and

$$M_w \leq \tfrac{1}{2}\left[2^2+4^2+\cdots+\left(\frac{N}{8}\right)^2+\left(\frac{N}{4}\right)^2+\left(\frac{N}{2}\right)^2\right]+2, \qquad (2.55b)$$

which represents a tighter bound for the case $g_i \in \mathbf{R}$ for all i.

Table 2.2 shows the values of A_f, M_f, A_w, and M_w for N=4 to N=128 using expressions (2.53) and (2.54). It is therefore evident that the system of Fig. 2.2 is computationally more efficient than that of Fig. 2.1 for $N \leq 64$ if $g_0, g_{N/2} \in \mathbf{R}$ but $g_i = g^*_{N-i}$ (i = 1,2, ... , N/2 - 1). Similarly, if $g_i \in \mathbf{R}$ for all i, then the system of Fig. 2.2 is computationally more efficient than that of Fig. 2.1 for $N \leq 128$ (based on comparing (2.55) with (2.53)). These computational comparisons assume the implementation of the system of Figs. 2.1 and 2.2 on a sequential processor.

	Fourier	Domain	Walsh	Domain
N	A_f	M_f	A_w	M_w
4	40	48	18	6
8	112	128	62	22
16	288	320	198	86
32	704	768	630	342
64	1664	1792	2070	1366
128	3840	4096	7126	5462

Table 2.2: A comparison of the computational requirements for spectral filtering in the Fourier and Walsh domains.

For examples of partial Walsh gain matrices (case N=8) the reader should consult Appendix E. Appendix F contains an example of G_w for N=16. The prototype is a first order Butterworth filter.

2.3.2 Haar Transform

The use of T = H (discrete Haar transform (DHT)) and $T^{-1} = H^{-1}$ (inverse discrete Haar transform (IDHT)) will now be considered. This subject was first considered by Zarowski, Yunik and Martens [24]. In this case G_t becomes G_h, the Haar gain matrix, and $G_t^{(k)}$ becomes $G_h^{(k)}$, the kth partial Haar gain matrix. Equation (2.3b) can be rewritten as

$$G_h = HF^{-1}G_f FH^{-1} . \tag{2.56}$$

In Ahmed et al. [23] the 2Nth order orthonormal Hadamard Haar transform is given as

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ 2^{k/2}I_N & -2^{k/2}I_N \end{bmatrix}, \tag{2.57}$$

where k = $\log_2 N$ here and H_N is the Nth order orthonormal Hadamard Haar transform. However, for the purposes of this thesis, an orthogonal Hadamard Haar transform of 2Nth order is defined as

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ I_N & -I_N \end{bmatrix} . \tag{2.58}$$

It will replace (2.57) in what follows. It is easy to see that

$$\Lambda_{2N}^{-1} = H_{2N} H_{2N}^T = \begin{bmatrix} 2H_N H_N^T & 0 \\ 0 & 2I_N \end{bmatrix} . \tag{2.59}$$

Matrix Λ_{2N} is a real and diagonal normalizing factor matrix with elements that are integer powers of two. Thus,

$$H_{2N}^{-1} = H_{2N}^T \Lambda_{2N} . \tag{2.60}$$

By using (2.58) instead of (2.57), the integer powers of $\sqrt{2}$ in (2.57) may be avoided. As an example, $\Lambda_8^{-1} = diag\{8,8,4,4,2,2,2,2\}$. Note that, strictly speaking, (2.59) and (2.60) only show that H_{2N} has a right inverse (2.60). However, H_{2N} is nonsingular and therefore invertible. Thus, it possesses a unique inverse, and it is a basic principle of linear algebra that this inverse is both a right and a left inverse for H_{2N}. Thus, (2.60) is certainly the inverse of H_{2N}. Using (2.60) in (2.56) yields

$$G_{h_{2N}} = H_{2N} F^{-1} G_f F H_{2N}^T \Lambda_{2N} \tag{2.61}$$

and

$$G_{h_{2N}}^{(k)} = H_{2N} A_{2N}^{(k)} H_{2N}^T \Lambda_{2N}$$

$$= \frac{g_k}{2N} H_{2N} \tilde{A}_{2N}^{(k)} H_{2N}^T \Lambda_{2N} . \tag{2.62}$$

There is no "nice" formula such as (2.24) for a typical element of H which can be used to give an expression for the typical element of $G_h^{(k)}$ as in (2.27). However, as a practical matter, this causes no problem. Equation (2.28) defines an off-line method for computing $G_w^{(k)}$ and hence G_W. Similarly, the first equality of (2.62) can be rewritten , without the 2N subscripts, as

$$G_h^{(k)}=HA^{(k)}H^T\Lambda=(\Lambda H\,(HA^{(k)})^T)^T\ . \tag{2.63}$$

It has been shown for the special case of T = W, in subsection 2.3.1, that the conditions on g_k given in (2.1) are sufficient to give $G_w \in \mathbf{R}^{N\times N}$. The reader should have noticed that the proof was actually independent of T (assuming $T \in \mathbf{R}^{N\times N}$). Thus, it is clear that the conditions on g_k in (2.1) will give $G_h \in \mathbf{R}^{N\times N}$. In effect then, the conjecture that $G_t \in \mathbf{R}^{N\times N}$ is true, if the conditions on g_k in (2.1) hold, and if (2.2b) is forced to equal (2.2a).

It is now necessary to show that G_h has the block-diagonal structure of Fig. 2.3(a). Using (2.58) and (2.13) in (2.62) gives

$$\begin{aligned}
G_{h_{2N}}{}^{(k)} &= \frac{g_k}{2N}\begin{bmatrix} H_N & H_N \\ I_N & -I_N \end{bmatrix}\begin{bmatrix} B & C \\ C & B \end{bmatrix}\begin{bmatrix} H_N^T & I_N \\ H_N^T & -I_N \end{bmatrix}\Lambda_{2N} \\
&= \frac{g_k}{N}\begin{bmatrix} H_N(B+C)H_N^T & 0 \\ 0 & B-C \end{bmatrix}\Lambda_{2N} \\
&= \frac{g_k}{N}\begin{bmatrix} (1+(-1)^k)H_N B H_N^T & 0 \\ 0 & (1-(-1)^k)B \end{bmatrix}\Lambda_{2N}\ .
\end{aligned} \tag{2.64}$$

Thus, since (2.64) is similar to (2.35), by arguments identical to that associated with (2.35), G_h has the block-diagonal structure of Fig. 2.3(a). Furthermore, Fig. 2.4 and Table 2.1 accurately specify the structure of G_h for N=32, as g_k with k odd once again only affects quadrant-4 and g_k with k even only affects quadrant-2.

The Haar domain analogue of (2.43) is simply matrix B, since only k odd is of interest. From (2.41) it is possible to write

$$B^{(k)}=\begin{bmatrix} Q^{(k)} & P^{(k)} \\ R^{(k)} & Q^{(k)} \end{bmatrix}, \tag{2.65}$$

to emphasize the k-dependence of B and its submatrices. Thus,

$$\begin{aligned} P^{(k)} &= [\alpha_{nm}^{(k)}+\beta_{nm}^{(k)}j] , \\ R^{(k)} &= [\alpha_{nm}^{(k)}-\beta_{nm}^{(k)}j] , \\ P^{(2N-k)} &= [\alpha_{nm}^{(k)}-\beta_{nm}^{(k)}j] , \\ R^{(2N-k)} &= [\alpha_{nm}^{(k)}+\beta_{nm}^{(k)}j] , \end{aligned} \tag{2.66}$$

where $P = R^H$ has been used. Now, $\beta_{nm}^{(k)} = \beta_{mn}^{(k)}$ and $\alpha_{nm}^{(k)} = -\alpha_{mn}^{(k)}$ so

$$P^{(k)} + P^{(2N-k)} = 2[\alpha_{nm}^{(k)}] ,$$

and

$$R^{(k)} + R^{(2N-k)} = 2[\alpha_{nm}^{(k)}] = -2[\alpha_{nm}^{(k)}] .$$

Thus, $B^{(k)} + B^{(2N-k)}$ has the form of (2.39), meaning that each diagonal block of G_h has this form. However, because typical elements of $B^{(k)}$ (B for short) are as in (2.45), its elements are not generally strictly real or strictly imaginary. Hence, $G_h^{(k)}$ cannot generally have strictly real or strictly imaginary components. The consequence of this seems to be that G_h has not got the structure depicted in Fig. 2.5. That is, when $g_k \in \mathbf{R}$ for all k, the 'Z' characters rarely go to '0' characters. Thus, suitable bounds on the computation associated with the use of the system of Fig. 2.2 with T=H are obtained using (2.54) but not (2.55). This implies that Table 2.2 holds for T=H as well as for T=W.

Note that B is Toeplitz. This implies that G_h will be Toeplitz in its diagonal blocks (block-Toeplitz). Such a property will be seen to have important implications in a VLSI context. This will be shown in Chapter III (section 3.4).

For examples of partial Haar gain matrices (case N=8) the reader should consult Appendix E. Appendix F contains an example of G_h for N=16. The prototype is a first order Butterworth filter.

2.3.3 Tridiagonal Transform

Now we consider the last special case for transform T, and this is $T = T_r$, the tridiagonal transform, and its corresponding inverse, T_r^{-1}. The gain matrix G_t becomes G_{t_r} (tridiagonal transform gain matrix) and $G_t^{(k)}$ becomes $G_{t_r}^{(k)}$ (kth partial tridiagonal transform gain matrix). The properties of G_{t_r} and $G_{t_r}^{(k)}$ were first studied by Zarowski, Yunik and Martens [24]. Equation (2.3b) can be rewritten as

$$G_{t_r} = T_r F^{-1} G_f F T_r^{-1} . \tag{2.67}$$

The 2Nth order tridiagonal transform T_r is defined as

$$T_{r_{2N}} = \frac{1}{\sqrt{2}} \begin{bmatrix} I_N & I_N \\ I_N & -I_N \end{bmatrix}, \tag{2.68}$$

and it is clearly self-inverse and symmetric. It may be readily seen that T_r is actually one of the $\log_2 N$ matrix factors making up the fast Walsh transform (Hadamard order) algorithm. This can be seen in Fig. 2.6 for N=8. In addition, it is one of the $\log_2 N$ matrix factors of the fast Haar transform (orthonormal or orthogonal Hadamard Haar transform [23]) algorithm, and of the decimation-in-frequency FFT [2] algorithm. What follows in this subsection appears to be the first time that this matrix factor has been considered as a valid transform in its own right.

To start, equation (2.62) may be rewritten as

$$G_{t_{r_{2N}}}^{(k)} = T_{r_{2N}} A_{2N}^{(k)} T_{r_{2N}}$$

$$= \frac{g_k}{2N} T_{r_{2N}} \tilde{A}_{2N}^{(k)} T_{r_{2N}} . \tag{2.69}$$

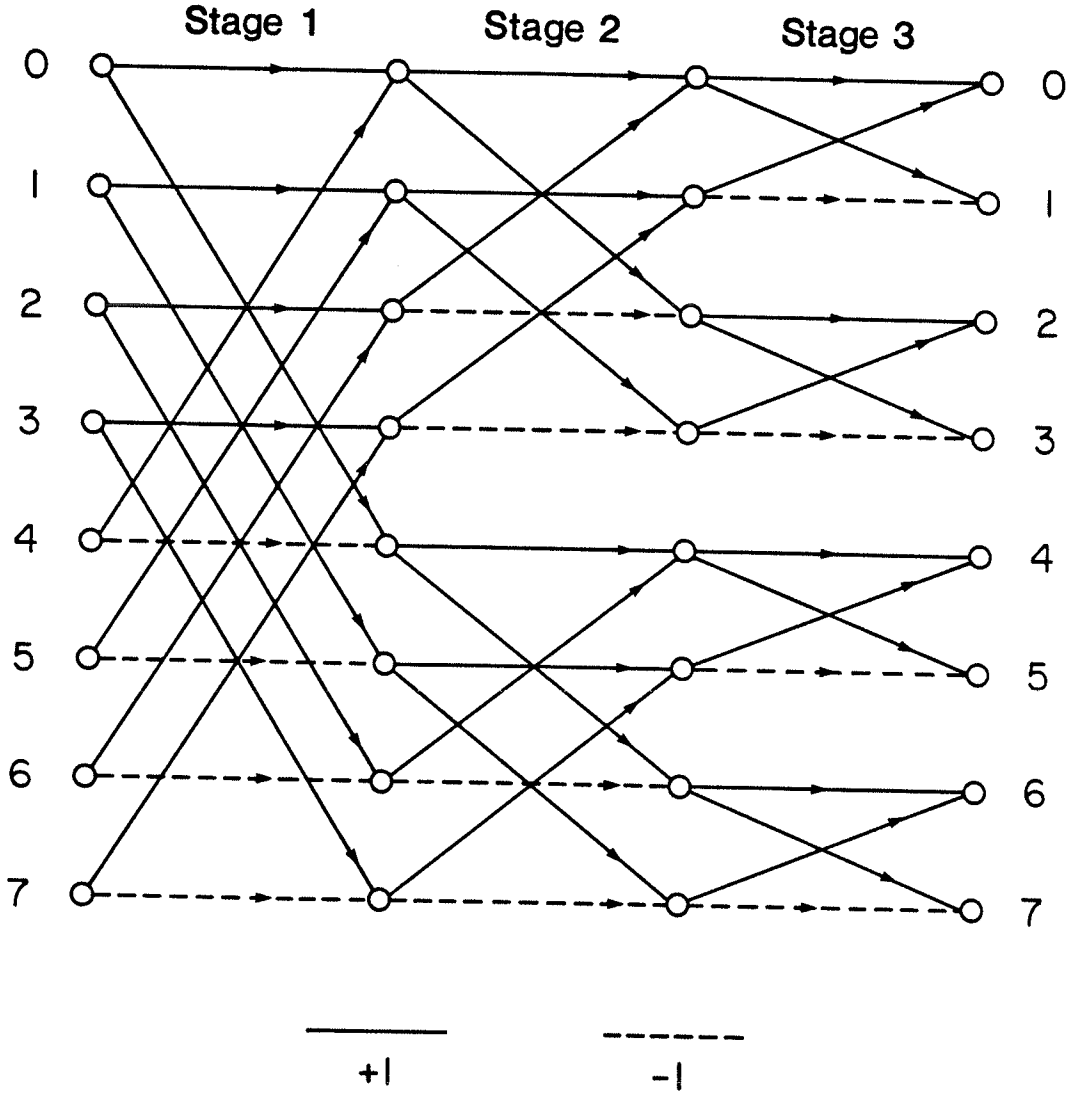

Figure 2.6: The order N = 8 Hadamard ordered DWT butterfly diagram.

The off-line procedure for computing $G_{t_r}^{(k)}$ and hence G_{t_r} may be described , using (2.69) without the 2N subscripts, by

$$G_{t_r}^{(k)} = (T_r(T_r A^{(k)})^T)^T \ . \tag{2.70}$$

Once again, the conditions on g_k in (2.1) are sufficient to cause $G_{t_r} \in \mathbf{R}^{N\times N}$ since T_r is a real matrix and (2.2b) is forced to equal (2.2a).

G_{t_r} also has a block-diagonal structure, except that quadrant-2 is dense. Substituting (2.13) and (2.68) into (2.69) gives

$$G_{t_{r_{2N}}}^{(k)} = \frac{g_k}{4N}\begin{bmatrix} I_N & I_N \\ I_N & -I_N \end{bmatrix}\begin{bmatrix} B & C \\ C & B \end{bmatrix}\begin{bmatrix} I_N & I_N \\ I_N & -I_N \end{bmatrix}$$

$$= \frac{g_k}{2N}\begin{bmatrix} B+C & 0 \\ 0 & B-C \end{bmatrix}$$

$$= \frac{g_k}{2N}\begin{bmatrix} (1+(-1)^k)B & 0 \\ 0 & (1-(-1)^k)B \end{bmatrix} . \tag{2.71}$$

It is clear that g_k for odd k only affects quadrant-4 and g_k for even k only affects quadrant-2. Furthermore, $G_{t_{r_{2N}}}^{(k)}$ is Hermitian and block-Toeplitz because B is Hermitian and Toeplitz. Thus, G_{t_r} will be Toeplitz in each of its two diagonal blocks. From arguments associated with $G_h^{(k)}$ it is true as well that G_{t_r} does not possess the structure of Fig. 2.5 (the 'Z' characters do not become '0' characters for real g_k in general). It is clear that quadrant-4 of G_{t_r} will have the form of (2.39), when k is odd of course. What about quadrant-2, where k is now even ?

Using the expression for p_{nm} following (2.46), and with k even, p_{nm} becomes

$$p_{nm} = (-1)^{k/2}\exp\left\{j\frac{\pi k}{N}(n-m)\right\}, \tag{2.72}$$

so

$$\alpha_{nm} = (-1)^{k/2} \cos\left[\frac{\pi k}{N}(n-m)\right], \tag{2.73a}$$

and

$$\beta_{nm} = (-1)^{k/2} \sin\left[\frac{\pi k}{N}(n-m)\right]. \tag{2.73b}$$

Thus, $r_{nm} = p^{*}_{mn} = \alpha_{mn} - \beta_{mn} j = \alpha_{nm} + \beta_{nm} j$ so R=P for even k values. Therefore, (2.65) is now

$$B^{(k)} = \begin{bmatrix} Q^{(k)} & P^{(k)} \\ P^{(k)} & Q^{(k)} \end{bmatrix}, \tag{2.74}$$

which implies that the 2nd quadrant of G_{t_r} has the form

$$\begin{bmatrix} X & Y \\ Y & X \end{bmatrix}, \tag{2.75}$$

and not the form of (2.39).

It is obvious that equations (2.54) and (2.55) do not hold for $T=T_r$. Because of the structure of G_{t_r}, (2.54) must be rewritten as

$$A_{t_r} \leq N + \frac{N^2}{2} \tag{2.76a}$$

and

$$M_{t_r} \leq \frac{N^2}{2}. \tag{2.76b}$$

Use has been made of the fact that T_r requires N/2 additions and N/2 subtractions in (2.76a). The $\sqrt{2}$ scale factor is not counted in (2.76b).

For examples of partial tridiagonal gain matrices (case N=8) the reader should consult Appendix E. Appendix F contains an example of G_{t_r} for N=16. The prototype is a first order Butterworth filter.

Finally, Appendix G contains programs, written in PASCAL, for the computation of the partial gain matrices, $G_w^{(k)}$, $G_h^{(k)}$, and $G_{t_r}^{(k)}$.

Chapter III

THE VLSI IMPLEMENTATION OF DFT SPECTRUM FILTERS

This chapter looks at the problem of constructing DFT spectrum filters, of the kind discussed in Chapter II, with special purpose hardware of a kind amenable to very large scale integration (VLSI) implementation. This is in contrast with Chapter II where only the sequential processor implementation of DFT spectrum filters was mentioned. There the comparison was based upon the number of additions/subtractions and multiplications needed to compute the filtered vector $\bar{x}'$ from the input vector $\bar{x}$. The comparison was with respect to the Cooley-Tukey FFT algorithm [2,3,7]. The two implementation methods considered here will be the radix 2, pipeline FFT [38] hardware algorithm, and the linear systolic array [39] hardware algorithm, which is a general purpose method for postmultiplying a matrix by a vector. The relative merits of implementing FFT and FWT with both hardware structures will be examined. The linear systolic array implementation of G_t will be considered as well, for the special cases of T that were considered in the previous chapter. Architectures such as the shuffle-exchange (SE) graph [40,41] and the cube-connected cycles (CCC) [40,42] organization will not be considered because, although they are fast structures, they require enormous amounts of integrated circuit (chip) area and so are not very practical, especially when hardware multipliers are needed. This issue will be discussed somewhat more fully in section 3.5. It is to be noted that only architectures will be considered here and not the details of implementation. Thompson's complexity theory for VLSI [43] is used to make the comparison between radix 2, pipeline FFT and the linear systolic array FFT on the basis of asymptotic area and time complexity. Thus, some review of Thompson's theory is

included in this chapter. It will be seen that the radix 2, pipeline FFT and the linear systolic array FFT have the same asymptotic area and time complexity, except that the radix 2, pipeline FFT is more area efficient by a constant factor than the linear systolic array FFT. However, the linear systolic array FFT is more readily cascadeable and designable than the radix 2, pipeline FFT, though perhaps only marginally.

3.1 THOMPSON'S VLSI COMPLEXITY THEORY

This section contains a brief review of Thompson's VLSI complexity theory [43]. The purpose of this section is to state some of the results of Thompson's theory and some associated terminology.

Thompson's VLSI complexity theory [43] is concerned with the relationship between the speed and size of VLSI circuits. Thompson's VLSI circuit model allows the determination of upper and lower bounds on the growth rates of chip area (A), and the time needed to solve a problem (T) versus the problem size (N). Thompson used the FFT and sorting as examples of the application of his theory. The first step in the process of determining speed-size relationships is the formation of the VLSI circuit model.

Thompson's theory is of a graph theoretic nature. The VLSI circuit is modeled by a collection of nodes and wires referred to as a communication graph.

There are three different categories of nodes: source nodes, sink nodes and switching nodes. Inputs to the computation are stored in the source nodes. The output values of a computation are collected at the sink nodes. Switching nodes may perform computations on information obtained from other switching nodes or from the source nodes. In turn, this information is passed either to other switching nodes or to the sink nodes. It is possible for a node to act as a source, sink or switching node simultaneously, or to act as some other combination of the three possible categories of nodes.

In Thompson's thesis [43] a set of assumptions about the communication graph and the nodes and wires that it is composed of are stated. There are eight formal assumptions associated with the lower bound proofs and an additional seven assumptions associated with the upper bound proofs. Only the eight lower bound proof assumptions are of any interest here. Thompson [44] used them to study the asymptotic area and time complexity of many of the competing methods of implementing the FFT in special purpose hardware. This includes the radix 2, pipeline FFT, which is referred to as the cascade, and the linear systolic array FFT, which is referred to as the N-cell DFT. The eight lower bound assumptions are summarized in [44] and so will not be repeated here. However, some of their consequences will be pointed out as the need arises.

Crucial to the formation of lower bound proofs is the concept that all communication graphs possess a minimum bisection width. The minimum bisection width of a communication graph is the smallest number of edges that must be removed in order to disconnect one half of the nodes from the other half. It is also required in the definition that half of the source nodes lie on either side of the bisection. Thompson [43] represents the minimum bisection width symbolically using the Greek letter ω. It is intuitively obvious that if ω is large then the graph is large, and hence the VLSI circuit is also large. However, the circuit will be fast as well since the available bandwidth is larger.

One of the main consequences of the lower bound assumptions and the concept of minimum bisection width,ω, is that

$$A \geq \frac{\omega^2}{4}, \tag{3.1}$$

and another important consequence is that

$$T = \Omega(N \log N)/\omega . \tag{3.2}$$

These results combine to give the optimal AT^2 metric

$$AT^2 = \Omega(N^2 \log^2 N) . \tag{3.3}$$

More generally, if $\omega = \theta(N^{1/2})$, then for $0 \leq x \leq 1$,

$$AT^{2x} = \Omega(N^{1+x} \log^{2x} N) . \tag{3.4}$$

N, for example, may be the number of components in the vector that is to be transformed using DFT or DWT, or the number of values that are to be sorted. The expressions of the form "$\theta()$","$\Omega()$" and "$O()$" are formally defined in Appendix H. All of (3.1) to (3.4) are proven formally in Thompson [43]. Result (3.1) is completely general (within the limits imposed by the lower bound assumptions). However, results (3.2) to (3.4) were proven formally only in the special cases of the DFT and sorting, although (3.2) to (3.4) are more generally applicable than the special cases of DFT and sorting would suggest. The significance of the optimal AT^2 complexity metric is that it proves the existence of a fundamental lower limit on the growth in complexity of a VLSI chip. A limit asymptotically lower than that, say $AT^2 = \Omega(N \log N)$, is not physically possible. Equation (3.4) is a more general expression of this fact than (3.3).

Equation (3.3) can be rewritten as

$$AT^2 = cN^2 \log^2 N \text{ , for } N \geq N_0 , \tag{3.5}$$

where c is called the technology dependent constant (TDC). $N \geq N_0$ should be interpreted to mean "for N sufficiently large" in (3.5). Given two circuits which solve the same problem and which also possess the same asymptotic area and asymptotic time complexity taken separately, this implies that they have the same AT^2 metric, to within a constant factor. It is possible to decide which circuit is the best by examining c, the TDC, in the resulting expressions for AT^2. The better circuit possesses the smaller c. However, unless c is very different (perhaps even orders of magnitude) for both circuits it may be necessary to look at the asymptotic area and time complexity , and their respective TDCs, separately in order to decide which circuit is the

best. Thus, the AT^2 metric by itself is not usually adequate for choosing the best circuit. Better measures of performance are obtained by looking at A-versus-N and T-versus-N separately and by comparing their respective TDCs. This is the problem faced when comparing the linear systolic array FFT method with the radix 2, pipeline FFT method. Thompson [44] has shown that both have the same A metric, T metric and AT^2 metric , to within a constant factor. Thus, only the TDCs of all three metrics can allow us to choose which method is the best in terms of area and time complexity.

3.2 THE RADIX 2 PIPELINE FFT (CASCADE)

This section describes the radix 2, pipeline FFT method. Much of the description is taken from Rabiner and Gold [38]. Thompson [44] referred to this structure simply as the cascade and this is what it will usually be called from now on except where confusion is likely to arise. Thompson [44] compared the A,T, and AT^2 metrics of this structure with other parallel and/or pipelined special purpose hardware structures for the FFT. Thompson's results for the cascade will be presented in this section along with estimates of the technology dependent constant (TDC) values assuming a particular fabrication technology. In addition, the cascade implementation of FWT and FHT will be treated briefly. The tridiagonal transform will be seen as merely one stage in the cascade. The only other structure of interest is the linear systolic array FFT, to be discussed in the next section. For the A,T, and AT^2 metrics of structures other than the cascade and linear systolic array FFT, the reader must consult Thompson [44].

The cascade FFT method is a technique for implementing the decimation-in-frequency (DIF) FFT algorithm (see Oppenheim and Schafer [2]), using $\log_2 N$ processing stages. The method is originally due to Groginsky and Works [45].

Figure 3.1 depicts the DIF FFT butterfly diagram for N = 8. In this figure, $W = \exp\left\{-j\frac{2\pi}{N}\right\}$ in general. Note that there are essentially three ($=\log_2 8$) stages in

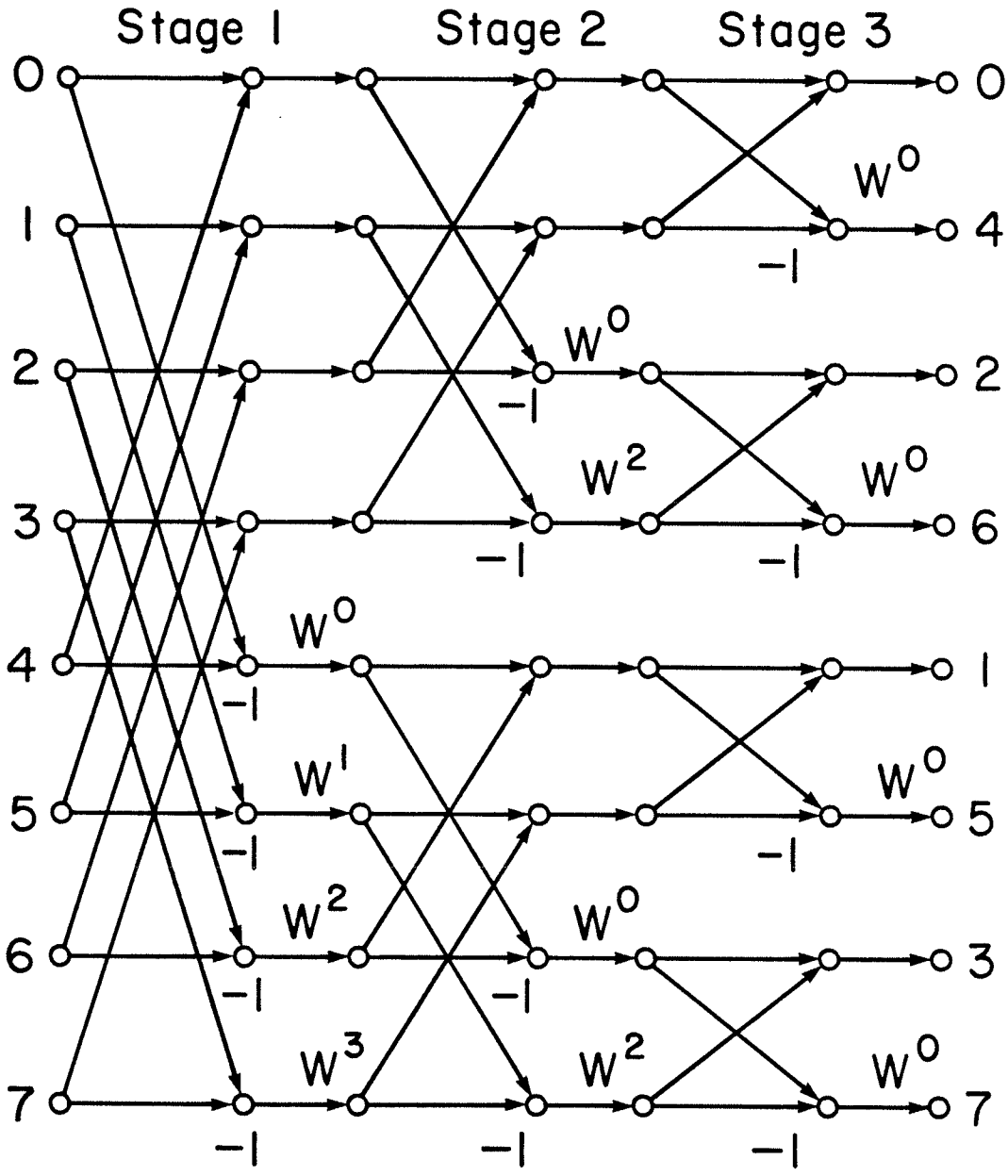

Figure 3.1: The butterfly diagram for the N = 8 point decimation-in-frequency (DIF) fast Fourier transform (FFT) algorithm (taken from Oppenheim and Schafer [2]).

the butterfly diagram. In general, there will be $\log_2 N$ stages. In terms of implementing the DIF FFT on a sequential processor, the amount of computation is the same as that for the Cooley-Tukey FFT. That is, O(NlogN) complex operations are needed. Also note that DIF FFT produces the DFT spectrum in bit-reversed-order (BRO) (see [2]). Cooley-Tukey FFT ordinarily gives output in natural order.

Figure 3.2 depicts the cascade architecture for N=8. In other words, the structure of Fig. 3.2 implements the butterfly diagram shown in Fig. 3.1. Figure 3.3 depicts the timing diagram associated with the structure of Fig. 3.2.

Notice the basic components of the cascade architecture depicted in Fig. 3.2. There are switching units (also called commutators in the literature) and these are labelled SWi. There are delays (a form of memory) which are labelled z^{-k} and there are butterfly computer units which are labelled BCi, where $i \in \{1,2,...,\log_2 N\}$. In addition, there are $\log_2 N$ coefficient memory units associated with each BCi unit. The coefficient memory units are used to store the complex factors W^p. Note that these factors are trivial in the last stage because p=0 at this stage. Thus, there is no real need for coefficient storage at BC3.

The integers 0,1,...,7 in the timing diagram of Fig. 3.3 represent the data points x(0),x(1),...,x(7), respectively. The basic clocking interval of the cascade is the sample period. The first switching unit, SW1, feeds the points 0,1,2,3 into the four unit delay z^{-4} when SW1 is high in Fig. 3.3. The remaining input points are switched to the lower branch of the circuit when SW1 is low. Clearly, there is a four time unit delay before BC1 can start computation. Output from BC1 appears four time units after the first data point was entered. The output appears at points B and C in Fig. 3.2. The switching units after SW1 operate somewhat differently. In Fig. 3.3, when SW2 is low, data flows from points D to G and E to F. When SW2 is high, data flows from points D to F and E to G. The operation of SW3 is essentially the same. When SW3 is low, data flows from J to M and K to L, and when SW3 is high the data flows from J to L and from K to M. The final DFT (DIF) values appear in pairs

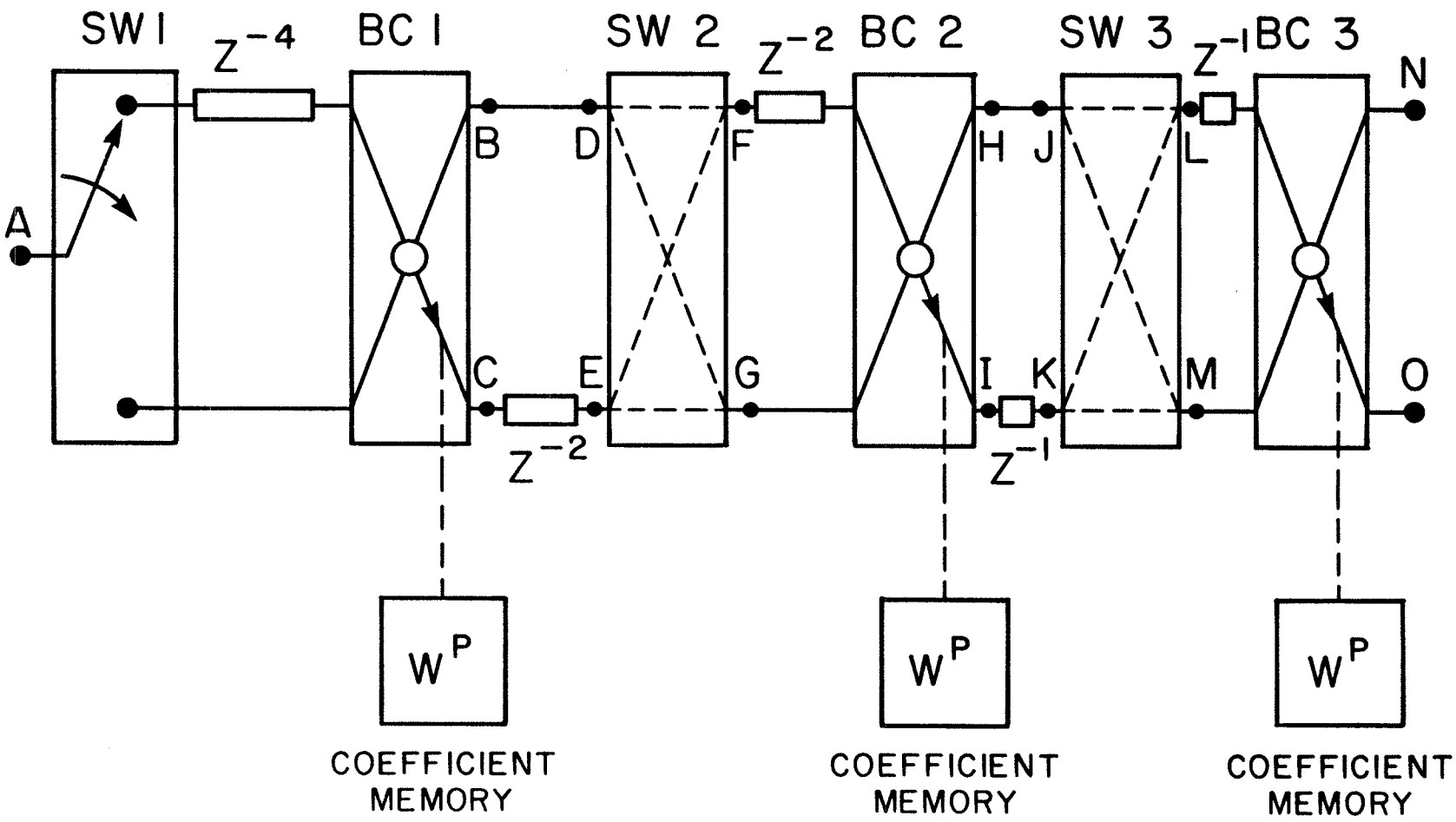

Figure 3.2: The radix 2, pipeline FFT structure for the N = 8 point DIF FFT (adapted from Rabiner and Gold [38]).

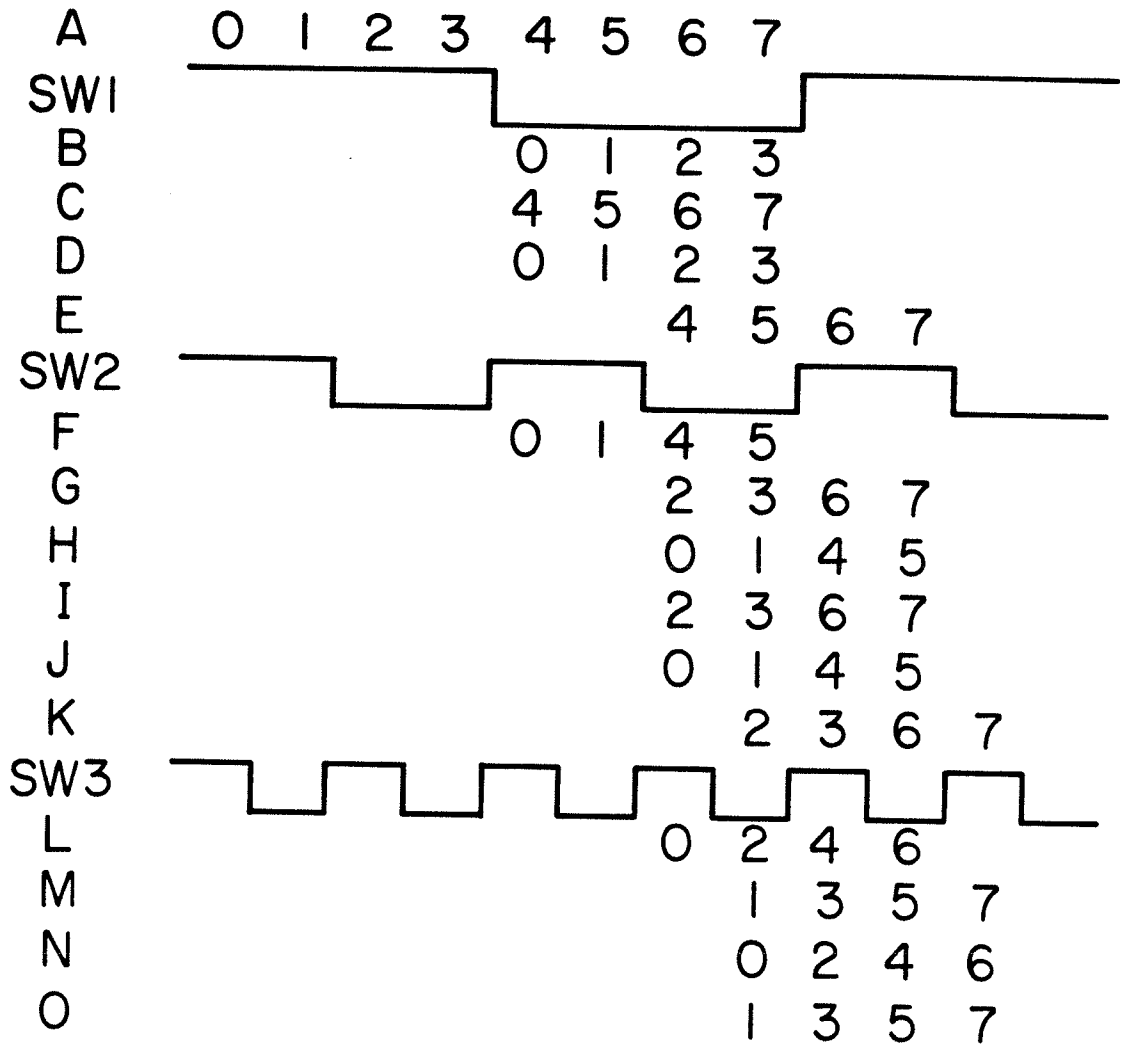

Figure 3.3: The timing diagram for the N = 8 point, radix 2, pipeline FFT structure of Figure 3.2.

at points N and O in Fig. 3.2. The output is in BRO. Thus, for the last two lines of Fig. 3.3 concerning the N and O outputs, the following correspondences hold :

$$\begin{array}{llll} 0 \rightarrow X(0) & 2 \rightarrow X(2) & 4 \rightarrow X(1) & 6 \rightarrow X(3) \\ 1 \rightarrow X(4) & 3 \rightarrow X(6) & 5 \rightarrow X(5) & 7 \rightarrow X(7) \end{array}$$

Therefore, from an inspection of Fig. 3.2 and Fig. 3.3 the operation and properties of the cascade can be summarized as follows:

1. The delay of a given stage is half as long as the delay in the preceding stage.
2. The system is only 50% efficient in the sense that each BCi is operational only 50% of the time.
3. Each switching unit operates at twice the frequency of its predecessor.
4. The basic time unit of the cascade is the sampling period.
5. The output data appear in pairs and in BRO.
6. The cascade is a pipelined structure as one N-point vector after another can be fed into the structure.

A more complete description of the cascade can be found in Rabiner and Gold [38] where an N=16 point cascade is used as an example. In Rabiner and Gold [38] a scheme for increasing the cascade's efficiency to 100% , in the sense of point 2 above, is proposed. This method involves a more sophisticated switching and buffering scheme.

Many variations on the cascade concept are possible. Rabiner and Gold [38] describe a radix 4, pipeline FFT hardware algorithm and compare it to the radix 2, pipeline FFT. The structure is more complex than the cascade just described and will not be presented in this thesis.

Many of the variations on the cascade of Fig. 3.2 often center around the manner in which multiplications are to be performed. The stored-product method of Peled and Liu [46] has been suggested as a means of implementing the cascade in Liu

and Peled [47]. Despain [48] suggests using Volder's [49] CORDIC complex multiplier structure to implement radix 2, and higher radix, pipeline FFT circuits.

The cascade of Fig. 3.2 can be used to implement the DFT spectrum filtering scheme of Fig. 2.1. The first block of Fig. 2.1 (labelled F) is replaced by the cascade structure. There are two complex multipliers needed to implement the Fourier filter block (labelled G_f in Fig. 2.1). Each of the two outputs from the first cascade goes to one of these multipliers. A G_f filter coefficient store is attached to the remaining inputs at each multiplier. The multiplier outputs of the G_f block feed to the input buffer memory of the second cascade, which implements the IFFT (F^{-1} block in Fig. 2.1). Naturally, allowance is made for the fact that the output of the first cascade is in BRO, that the output values from this cascade appear in pairs, and that IDFT is not exactly the same as DFT. The second cascade can take on exactly the same structure as the first cascade if both the input values to the second cascade and the output values from it are conjugated. These conjugation operations effectively correspond to the IDFT. The implementation just described is clearly time-domain convolution by the multiplication of DFT spectra (see Chapter I). More details on the use of the cascade in this type of operation (DFT spectrum filtering) can be found in Rabiner and Gold [38].

Having described the structure and operation of the cascade, it is now worth considering its area-time complexity. According to Thompson [44], the cascade has the following asymptotic complexity metrics,

$$A_c = \Omega(N \log N), \tag{3.6a}$$

$$T_c = \Omega(N \log N), \tag{3.6b}$$

$$D_c = \Omega(N \log^2 N), \tag{3.6c}$$

and

$$A_c T_c^2 = \Omega(N^3 \log^3 N), \tag{3.6d}$$

where A_c is the area complexity, T_c is the processing time complexity, and D_c is the delay complexity. The logN factors arise as a result of Thompson's assumptions concerning the problem size (N), the number of distinct values that any one of the N inputs may take on (P), and the word length (M). Clearly, there are P^N distinct problem instances, which Thompson assumed to occur with equal likelihood. Thompson also assumed that logP = θ(logN) so M = c $\log_2$N bits, since some assumption about the relationship between M and N is needed in order to avoid having A, T and hence AT^2, depend explicitly upon M. In addition, Thompson [44] assumes that the multiply-add of a BC takes O(logN) time, and O(logN) area.

A different version of (3.6a)-(3.6c) will now be presented. The asymptotic results will contain no explicit logN factors, though they are present implicitly if Thompson's assumptions in [44] are allowed to hold. The time complexity of the cascade is characterized by

$$T_c = \frac{3}{2}N - 1 = 1 + \frac{N}{2} + \sum_{i=2}^{\log_2 N} \frac{N}{2^{i-1}} \quad , N \geq 8 \tag{3.7a}$$

$$D_c = N \quad . \tag{3.7b}$$

Thus, $T_c = \Omega(N)$, $D_c = \Omega(N)$. The area complexity is approximated reasonably well by

$$A_c = 2\,(T_c - 1)a_d M + (2\alpha_{ca}(M) + \alpha_{cm}(M))\log_2 N + 4a_{tg} M + 8a_{tg} M\,(\log_2 N - 1)$$

$$+ a_{co}\log_2 N + 2a_m M \sum_{i=1}^{\log_2 N} \frac{N}{2^i} \quad , \tag{3.8}$$

and

$$\alpha_{ca}(M) = 2(M-1)a_{fa} + 2a_{ha} \quad , \tag{3.9a}$$

$$\alpha_{rm}(M) = (M-1)^2(a_{fa} + a_g) + 2(M-1)(a_{ha} + a_g) \quad , \tag{3.9b}$$

so

$$\alpha_{cm}(M) = 4\alpha_{rm}(M) + \alpha_{ca}(M) . \qquad (3.9c)$$

The symbols above are defined as follows :

N = number of transform points (a positive integer power of two),

M = number of bits in the real or imaginary part of a number,

a_d = 1-bit delay element area,

a_{fa} = 1-bit full adder area,

a_{ha} = 1-bit half adder area,

a_g = area of a 2-input nand gate plus an inverter (and gate),

a_m = area of a 1-bit memory cell (could be RAM or ROM),

a_{tg} = area of a transmission gate,

a_{co} = area of control circuitry per stage ($\log_2$N stages),

α_{ca}(M) = area of a complex adder,

α_{rm}(M) = area of a real array multiplier (see Hamacher et al. [50]),

α_{cm}(M) = area of a complex multiplier.

In the formula for A_c in (3.8) no allowance has been made for the interconnecting wire area or the area taken up by power supply and ground lines or I/O pads. This causes no real problem since it is the lower bound on the true area that we are after. The expressions for α_{ca}(M), α_{rm}(M), and α_{cm}(M) do not allow for the area taken up by the sign-change circuitry. Such circuitry depends upon M but not N. Clearly, its omission weakens the lower bound on the true area estimate somewhat. Note that a_{tg} is the area of a transmission gate. This suggests the use of CMOS technology. In fact, when estimates of the above constants are made later in this section, the source of the estimates will be from a particular static CMOS technology. More will be said on this at the proper time. It is now necessary to describe the terms making up equation (3.8)

above. This description is as follows:

$2(T_c - 1)a_d M$ = delay area term ,

$(2\alpha_{ca}(M) + \alpha_{cm}(M))\log_2 N$ = butterfly computer (BC) area term,

$4a_{tg}M$ = area term for SW1,

$8a_{tg}M(\log_2 N - 1)$ = area term for SWi (i> 1),

$a_{co}\log_2 N$ = control area term,

$2a_m M \sum_{i=1}^{\log_2 N} \frac{N}{2^i}$ = coefficient memory term.

Figure 3.4 illustrates the structure of some of the subsystems making up the cascade. The areas of these structures give rise to some of the area terms in (3.8) above. These subsystems are the butterfly computing units (BCi), and the switching units (SWi). Figure 3.4(a) shows the computational structure of the BCi. There are two complex adders and there is a single complex multiplier. The structure is simply that of a typical butterfly from Fig. 3.1. Figure 3.4(b) shows the first switching unit, SW1, and is composed of $2\hat{M}$ = 4M transmission gates. The area due to inverters for buffering and control is neglected in the area term for SW1 in (3.8). Figure 3.4(c) shows SWi (i> 1). It is composed of $4\hat{M}$ = 8M transmission gates. Once again, the area due to inverters for control and buffering is neglected in the area term for SWi (i> 1) in (3.8). From (3.10) below these omissions have no effect at all on the asymptotic behaviour of A_c (meaning $\lim_{N\to\infty} A_c$ is independent of a_{tg}).

Using the fact that $\sum_{i=1}^{p} 2^{-i} = 1-2^{-p}$ and using (3.7a) and (3.9) in (3.8) allows (3.8) to be expanded as,

$$A_c = (3a_d + 2a_m)MN$$

$$+ \left[4(a_{fa}+a_g)M^2 + (8a_{ha}+8a_{tg}-2a_{fa})M + a_{co}-2a_{fa}-2a_{ha}-4a_g\right]\log_2 N$$

$$- (4a_d+4a_{tg}+2a_m)M \ , \qquad (3.10)$$

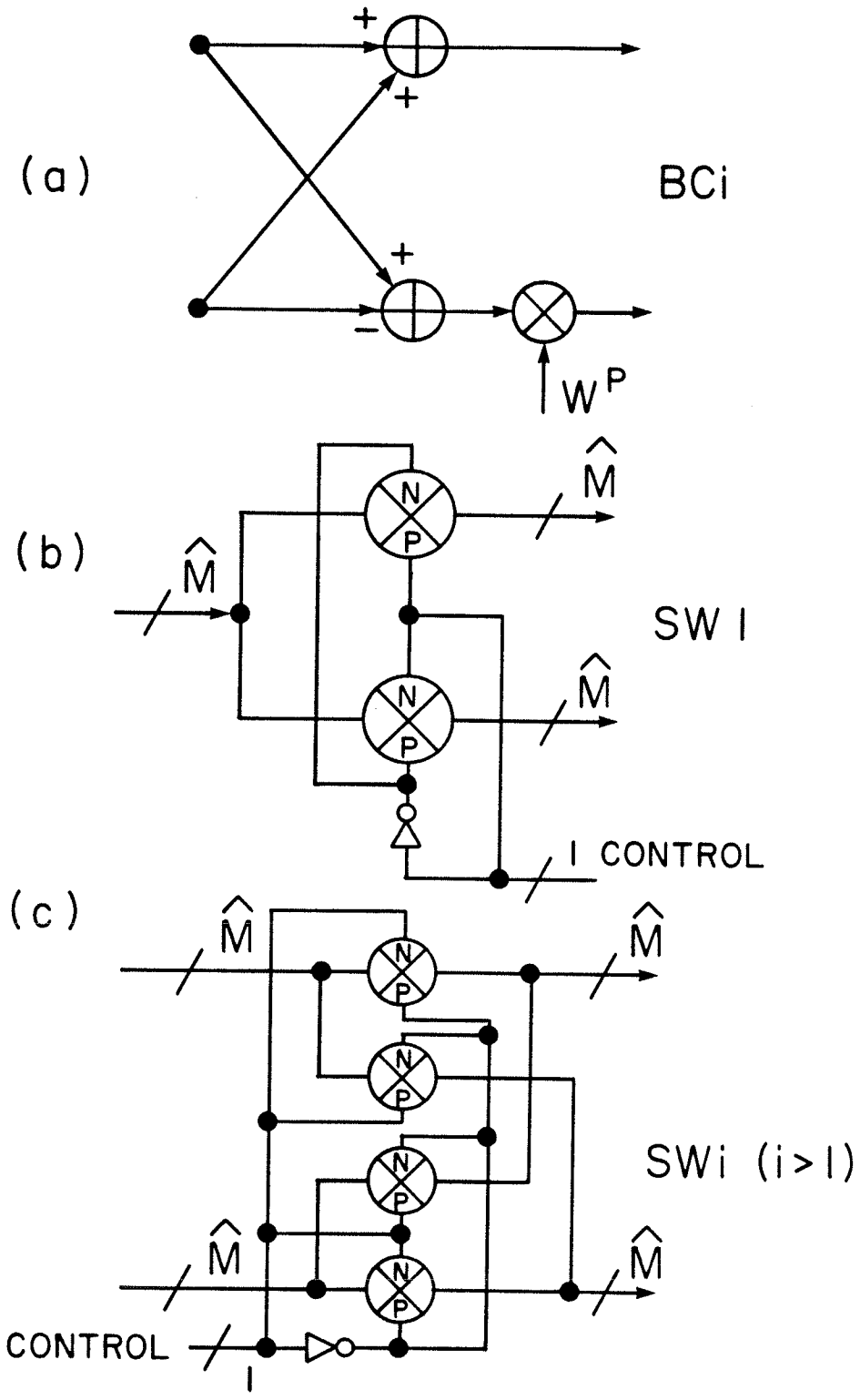

Figure 3.4: The proposed structure of the BCi and SWi that make up the N-point, radix 2, pipeline FFT circuit of Figure 3.2.

and clearly $A_c = \Omega(N)$. It should be clear that the technology dependent constant (TDC) associated with T_c is 3/2 and the TDC associated with A_c is $(3a_d + 2a_m)M$. Thus, in asymptotic terms, the area of the cascade is mainly memory since a_d and a_m are constants which specify storage area.

Now it is worth estimating the constants (a_d,a_m, etc.) for a particular static CMOS technology. The fabrication process of interest is Northern Telecom's CMOS1B process. This process allows minimum device dimensions of 5μm. It is a p-well CMOS technology (n-devices sit in wells of p-type silicon). There is only a single layer of metal and a single layer of polysilicon. Table 3.1 below gives estimates for a_d,a_g,a_m,a_{fa},a_{ha}, and a_{tg} based on the sizes of actual circuits designed for fabrication with the CMOS1B process here at the University of Manitoba. Complete descriptions of the cells, from which the data of Table 3.1 was obtained, will eventually become available in print. The cells were designed by a number of graduate students as part of the requirements for a graduate course on VLSI design presented in the fall term of 1984 here at the University of Manitoba.

Parameter	Cell Dimension (μm × μm)	Area (μm^2)
a_g	130 × 120 (est.)	15,600
a_e	92 × 152	13,984
a_{ha}	201 × 167	33,567
a_{fa}	470 × 199	93,530
a_{tg}	53 × 64	3,392
a_d	161 × 300	48,300
a_m	118 × 120	14,160

Table 3.1: Values for the area parameters based upon some CMOS standard cell dimensions.

The entry for a_g in Table 3.1 is an estimate based upon the size of an inverter (59 × 106 μm) and a two-input nand gate (69 × 111 μm). The entry for a_m is the area of a six- transistor static RAM cell. This cell is actually much larger than the best commercially available cells since commercial cells often use smaller device sizes and two layers of metal. The overhead associated with buffering, sense amplifiers, address decoding, etc., is not included in the figure for a_m. The entry for a_d is based upon the area of a D flip-flop that does not have a preset or clear. The figure for a_{tg} in Table 3.1 above does not include the area taken up by the ground line which must run through the cell in order to ground out the p-well. In what follows, a_{co} will be assumed to be zero. Parameter a_e is the area of a two-input exclusive-or gate (used in section 3.3).

Using the parameters of Table 3.1 above, it is now possible to write (3.10) as

$$A_c = 173{,}000MN + [437{,}000M^2 + 109{,}000M - 317{,}000]log_2 N - 235{,}000M \quad (\mu m^2). \qquad (3.11)$$

This gives a TDC for A_c of the cascade FFT of 173,000M μm^2.

The cascade of Fig. 3.2 can be used to implement the FWT or even the FHT. The tridiagonal transform requires only one stage of the cascade for all N and so is especially simple to implement. Indeed, the tridiagonal transform is probably too simple to be of any real value in an application. This will be seen more clearly later on. In implementing the FHT, it must be possible to have the butterfly computers turned off and data allowed to flow through unaffected in stages 2 to $\log_2 N$, during certain time periods defined by the structure of the FHT butterfly diagram. A butterfly diagram for the Cooley-Tukey type FHT is to be found in [23]. This butterfly is structurally similar to the DIF FFT butterfly and Hadamard order FWT butterfly diagrams. No more will be said on this subject in this thesis.

It is possible to evaluate A_c for the FWT cascade. To this end, equation (3.8) can be rewritten as,

$$A_c = (T_c - 1)a_d M + 2\alpha_{ra}(M)\log_2 N + 2a_{tg}M + 4a_{tg}M(\log_2 N - 1) + a_{co}\log_2 N \ , \qquad (3.12)$$

where $\alpha_{ra}(M) = (M-1)a_{fa} + a_{ha}$ (area of an M-bit real adder). The terms making up the expression for A_c in (3.12) are as follows:

$(T_c - 1)a_d M$ = delay area term,

$2\alpha_{ra}(M)\log_2 N$ = butterfly computer (BC) area term,

$2a_{tg}M$ = area term for SW1,

$4a_{tg}M(\log_2 N - 1)$ = area term for SWi (i> 1),

$a_{co}\log_2 N$ = control area term.

Note that there is no coefficient memory term in (3.12). Thus, the butterfly computer unit depicted in Fig. 3.4(a) reduces to two real adders and the complex multiplier becomes redundant since there are no multiplier coefficients in the FWT butterfly (see Fig. 2.6). The area taken up by switching units is cut in half because only real numbers are being handled by the cascade, implying M-bit rather than 2M-bit words. The FWT cascade implements the butterfly diagram of Fig. 2.6, as has

already been suggested, and so the output appears in natural (Hadamard) order in contrast with the BRO of the DIF FFT cascade output.

Assuming $a_{co} = 0$,(3.12) can be reduced, using (3.7a) and the expression for $\alpha_{ra}(M)$, to

$$A_c = \frac{3}{2}a_d MN + [(2a_{fa}+4a_{tg})M + 2(a_{ha}-a_{fa})]log_2 N - 2(a_{tg}+a_d)M \ , \qquad (3.13)$$

and upon using the values in Table 3.1 above,

$$A_c = 72{,}500MN + [201{,}000M - 120{,}000]log_2 N - 103{,}000M \quad (\mu m^2) \ . \qquad (3.14)$$

3.3 LINEAR SYSTOLIC ARRAY FFT (N-CELL DFT)

In the present section the linear systolic array FFT will be discussed in a manner similar to the discussion of the cascade in the previous section. Much of the discussion to follow is taken from Mead and Conway [39] and Thompson [44]. The discussion of linear systolic arrays in Mead and Conway [39] is taken from original work by Kung and Leiserson [51]. Thompson [44] discusses the A,T, and AT^2 metrics for the linear systolic array FFT, or N-cell DFT as it is called in [44]. These results will be presented here. Bridges et al. [52] discuss the linear systolic array implementation of the FWT and FHT and their results will be summarized here as well. A comparison of the N-cell DFT and the cascade of the previous section will be made in a later section of this chapter.

The linear systolic array (LSA) [39,40] is a parallel-pipelined architecture for postmultiplying a matrix by a vector. The cascade is similar in this regard except that it is much more specialized. It can only postmultiply a matrix by a vector if the matrix has a particular structure. The LSA can handle arbitrary matrices and so is much more generally applicable than the cascade. However, our concern lies with using the LSA to implement the FFT, FWT and FHT. The LSA implementation of

the G_t filters of Chapter II will be treated in the next section.

The LSA consists of a linear array of what are called inner product step processors (IPSPs). The structure is depicted in Fig. 3.5. In Fig. 3.5 a 3×3 matrix, $A = [a_{ij}]$, is postmultiplied by vector $\bar{x}=[x_1,x_2,x_3]^T$ in order to give the solution vector $\bar{y}=[y_1,y_2,y_3]^T$. For vectors $\bar{x}=[x_1, \ldots ,x_N]^T$ and $\bar{y}=[y_1, \ldots ,y_N]^T$ the LSA implements the recurrences,

$$y_i^{(1)} = 0 ,$$

$$y_i^{(j+1)} = y_i^{(j)} + a_{ij} x_j ,$$

$$y_i = y_i^{(N+1)} ,$$

where A is an N×N band matrix of band width w. In Fig. 3.5 w=5. Note in Fig. 3.5 that the coefficients along each diagonal of A are fed into one and only one IPSP. Each IPSP is composed of three registers (R_x,R_a, and R_y), a complex adder and a complex multiplier.

The y_i, which are initially zero, move to the left while the x_i move to the right and the a_{ij} move down. All data movement is synchronized. If the system clock period is τ then the data points x_i are clocked in every 2τ (this would be the sample period) time units. IPSPs are active once every 2τ time units. The first output value, y_1, appears in $w\tau$ time units. The time to process all points is $(2N+w)\tau$ time units. More details of the operation and timing of the LSA are to be found in Mead and Conway [39].

The diagram of Fig. 3.5 implies that a coefficient memory is needed to hold each a_{ij}, in particular, if FFT is to be performed. In [39] a method of generating the a_{ij} for FFT on-the-fly is proposed. The operation of this LSA FFT circuit is described in [39]. It is enough to know that each IPSP needs an extra register (called R_t in [39]) and that the middle IPSP must store W (and only W needs to be stored). In addition, at each even-numbered time step the middle IPSP performs five complex

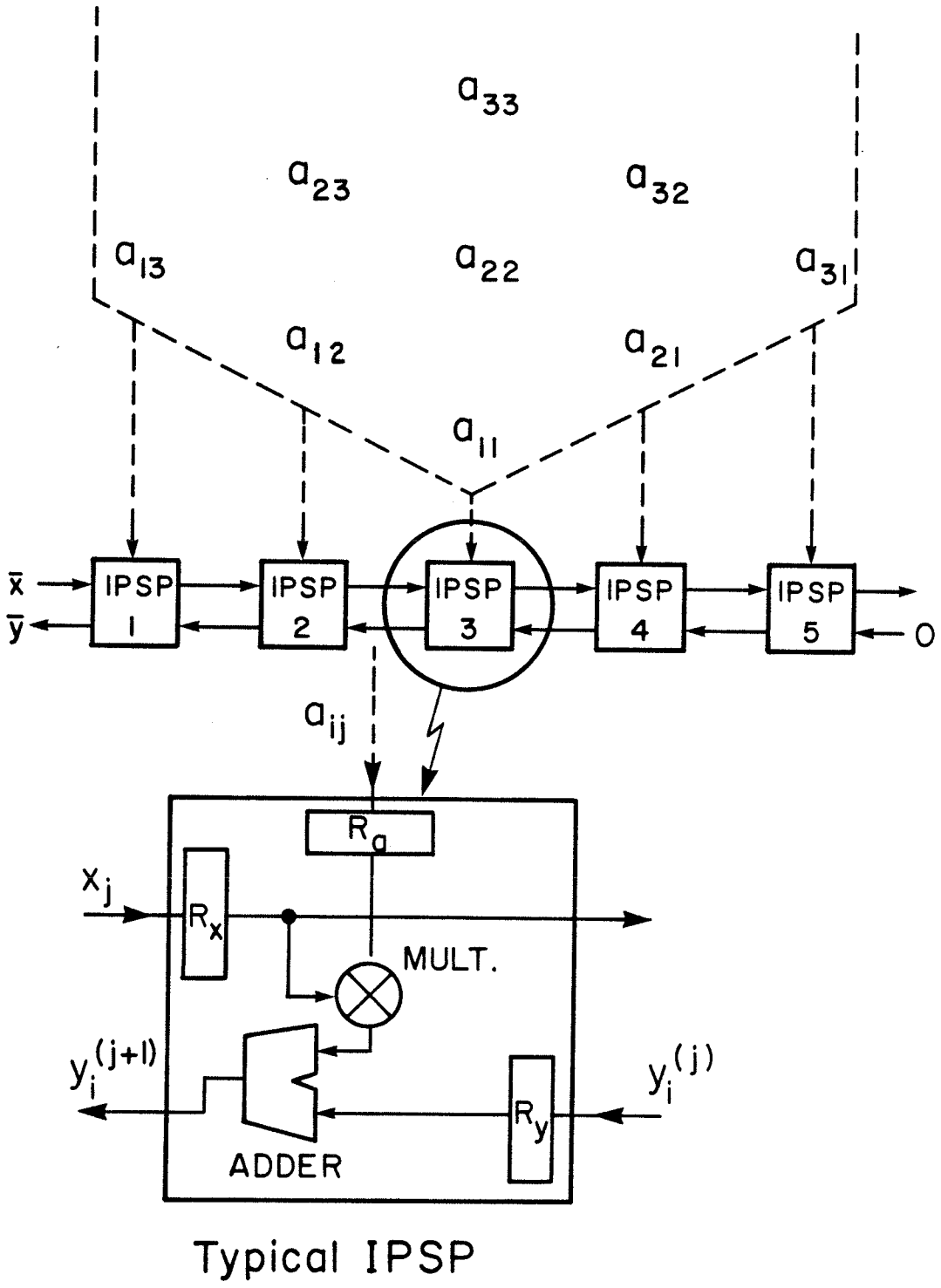

Figure 3.5: The linear systolic array (LSA) and a typical inner product step processor (IPSP).

mutliplications instead of just the two performed by the other IPSPs. This fact and the order in which the complex multiplications are performed by the middle IPSP allows one to assume that four complex multipliers are needed by the middle IPSP. It is this form of LSA FFT that will be used to obtain an expression for A_s, the LSA FFT area. A structure such as this has a delay of 2N-1 and a processing time of 4N-1 clock cycles.

Bridges et al. [52] developed a method of generating a_{ij} on-chip for the FWT and the FHT, and proposed a method of generating the a_{ij} for the FFT on-chip as well. Each IPSP has a co-processor associated with it (matrix element co-processor or MECP) dedicated to the task of producing the appropriate matrix coefficients. The method is effective only if the matrix coefficients are dependent upon the row and column indices of the matrix, and this is clearly the case for the DWT, DHT and DFT matrices. The structure they propose is very cascadeable.

From the preceding discussion and from Mead and Conway [39] the operation and properties of the LSA FFT can be summarized as follows:

1. The delay of any given stage is one clock cycle.
2. Each IPSP is the same as any other IPSP except for the middle IPSP.
3. The data is clocked in every second clock cycle.
4. The output data appear one point at a time every second clock cycle after a delay determined by the number of IPSPs and are in an order determined by the ordering of the matrix (for the LSA FFT above the natural order is produced).
5. The matrix coefficients are generated on alternate clock cycles (every second cycle).

6. Each IPSP is operational only 50% of the time and so the LSA FFT is only 50% efficient.

7. The LSA FFT is a pipelined structure for the same reason that the cascade is.

According to Thompson [44], the N-cell DFT has the following asymptotic complexity metrics,

$$A_s = \Omega(N \log N), \tag{3.15a}$$

$$T_s = \Omega(N \log N), \tag{3.15b}$$

$$D_s = \Omega(N^2 \log N), \tag{3.15c}$$

and

$$A_s T_s^2 = \Omega(N^3 \log^3 N), \tag{3.15d}$$

where A_s is the area complexity, T_s is the processing time complexity, and D_s is the delay complexity. Note that the expression for D_s given by Thompson can't be correct. A more plausible expression is $D_s = \Omega(N\log^2 N)$, the same as for the cascade, although even the $\log^2 N$ factor is suspect as well (logN seems more likely). Unfortunately, Thompson [44] does not explain how he arives at his delay complexity expressions. Once again, the logN factors arise because of Thompson's assumptions on multiplier complexity and the relationship between problem size and word length (see the paragraph containing (3.6) in the preceding section).

It has already been said that

$$T_s = 4N - 1, \tag{3.16a}$$

and

$$D_s = 2N - 1, \tag{3.16b}$$

and so it is clear that $T_s = \Omega(N)$, and $D_s = \Omega(N)$. The area complexity for the N-

cell DFT can be approximated by

$$A_s = a_{co}w + 8a_dMw + (w-1)(\alpha_{cm}(M)+\alpha_{ca}(M)) + (4\alpha_{cm}(M)+\alpha_{ca}(M)) + 2Ma_m , \quad (3.17)$$

where the parameters a_m,a_d,a_{co},M,α_{cm}(M), and α_{ca}(M) are defined as in section 3.2 and w is the matrix band width. The area terms are defined as follows:

$8a_dMw$ = delay (register) area term,

$(w-1)(\alpha_{cm}(M) + \alpha_{ca}(M))$ = IPSP arithmetic hardware area term (excludes middle IPSP),

$4\alpha_{cm}(M) + \alpha_{ca}(M)$ = middle IPSP arithmetic hardware area term,

$2Ma_m$ = area needed to store W in the middle IPSP,

$a_{co}w$ = control area per IPSP term.

Again, no allowance is made for interconnecting wire area, the area taken up by power and ground lines or I/O pads. In what follows, a_{co} for the N-cell DFT is assumed to be zero.

The expression for (3.17) can be expanded into

$$A_s = [8(a_{fa}+a_g)M^2+(16a_d+16a_{ha}-8a_{fa})M-8(a_g+a_{ha})]N + 8(a_{fa}+a_g)M^2 + (16a_{ha}+2a_m-14a_{fa}-8a_d)M + (6a_{fa}-8a_g-14a_{ha}) , \quad (3.18)$$

using the definitions from section 3.2. If the values in Table 3.1 are used then (3.18) becomes

$$A_s = [873{,}000M^2+562{,}000M-393{,}000]N + 873{,}000M^2-1{,}130{,}000M-33{,}600 \ (\mu m^2) , \quad (3.19)$$

and so $A_s = \omega(N)$.

Using the structure described in Bridges et al. [52], it is possible to get an expression for the area of the N-cell DWT. Thus,

$$A_s = A_{s_{IPSP}} + A_{s_{MECP}} , \tag{3.20}$$

where $A_{s_{IPSP}}$ is the area due to the IPSP, and $A_{s_{MECP}}$ is the area due to the MECP, the circuit which generates the matrix elements used by the IPSP. Thus,

$$A_{s_{IPSP}} = 3a_d Mw + \alpha_{ra}(M)w$$

$$= [(6a_d + 2a_{fa})M + 2(a_{ha} - a_{fa})]N - (3a_d + a_{fa})M + a_{fa} - a_{ha} , \tag{3.21a}$$

where $3a_dMw$ is the IPSP register area term and $\alpha_{ra}(M)w$ is the IPSP arithmetic hardware area term (only a single real adder). As well,

$$A_{s_{MECP}} = 2a_d \log_2 N + w\log_2 N\,[4a_{tg} + 3a_g + 2a_e] + a_{co}w$$

$$= (8a_{tg} + 6a_g + 4a_e)N\log_2 N + (2a_d - 4a_{tg} - 3a_g - 2a_e)\log_2 N , \tag{3.21b}$$

where $a_{co}w$ is an area term which allows for any extra control logic not included in the rest of the MECP area expression, and again we have assumed that a_{co} is zero. The term $2a_d\log_2N$ is the area due to the two counters (assumed to be ripple counters) which drive the MECPs and the term $w\log_2N[4a_{tg}+3a_g+ 2a_e]$ accounts for the area taken up by the MECPs themselves. A complete description of the MECPs can be found in [52]. Thus, (3.20) becomes

$$A_s = (8a_{tg} + 6a_g + 4a_e)N\log_2 N + [(6a_d + 2a_{fa})M + 2(a_{ha} - a_{fa})]N$$

$$+ (2a_d - 4a_{tg} - 3a_g - 2a_e)\log_2 N - (3a_d + a_{fa})M + a_{fa} - a_{ha} , \tag{3.22}$$

and so $A_s = \Omega(N\log_2N)$. The $\log_2N$ factor arises because each MECP contains $\log_2N$ common function blocks (see [52] for a definition of this term) and there are N MECPs. Using the parameters of Table 3.1 gives

$$A_s = 177{,}000N\log_2 N + [477{,}000M - 120{,}000]N$$

$$+ 8{,}260\log_2 N - 238{,}000M + 60{,}000 \ (\mu m^2) . \qquad (3.23)$$

in place of (3.22).

3.4 LINEAR SYSTOLIC ARRAY IMPLEMENTATIONS OF G_t

This section briefly considers some aspects of the implementation of the G_t filters of Chapter II for the special cases of G_w, G_h, and G_{t_r}. It will be seen that the structure of G_t has a significant influence on the asymptotic area complexity and hence the implementability of the filters.

Remember that the cascade is used for postmultiplying a matrix by a vector in Ω (N) time, provided that the matrix has the proper structure. Unfortunately, the G_t matrices do not appear to possess the proper structure. Thus, we are forced to implement them using the linear systolic array (LSA). The resulting LSA will consist of w=N-1 IPSPs for any of the G_ts of interest, and will have a processing time of 3N-1 clock cycles. In addition, it is important to note that due to the complexity of (2.3b), the on-chip generation of the G_t matrix coefficients is not feasible or advantageous, at least at present. This rules out the structure of Bridges et al. [52] as a viable means of implementing the G_t filters.

Matrix G_w has $O(N^2)$ distinct elements and therefore the area required to store these coefficients will grow as $\Omega(N^2)$. This implies that the LSA implementation of G_w has an asymptotic area complexity of $\Omega(N^2)$. Thus, G_w cannot be readily implemented with an LSA because of the large coefficient storage requirement.

Matrix G_h has O(N) distinct elements and so it is clear that the asymptotic area complexity of the LSA implementation of G_h is Ω (N). Since the diagonal blocks of G_h are Toeplitz, at most $\log_2 N + 1$ distinct nonzero coefficients need to be stored with each IPSP. Thus, G_h is easier to implement with a linear systolic array than G_w.

Matrix G_{t_r} possesses only two distinct diagonal blocks and both are Toeplitz. Thus, at most two distinct nonzero matrix coefficients need to be stored with each IPSP. The area due to coefficient storage grows as $\Omega(N)$ and so the asymptotic area complexity of the LSA implementation of G_{t_r} is $\Omega(N)$ as well. Thus, matrix G_{t_r} is more readily implemented than G_h with a linear systolic array.

The linear convolution of sequence {u(n)} with the filter function impulse response sequence {h(n)} is given in (1.10). If {u(n)} is an N_1-point sequence and {h(n)} is an N_2-point sequence then (1.10) can be expressed in matrix form as

$$\begin{bmatrix} y(0) \\ y(1) \\ \cdot \\ \cdot \\ \cdot \\ y(N_1-1) \end{bmatrix} = \begin{bmatrix} h(0) & 0 & \cdots & 0 \\ h(1) & h(0) & \cdots & 0 \\ h(2) & h(1) & \cdots & 0 \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ h(N_2-1) & h(N_2-2) & \cdots & 0 \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 0 & 0 & \cdots & h(0) \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \cdot \\ \cdot \\ x(N_1-1) \end{bmatrix}, \tag{3.24}$$

where it is assumed that $N_1 > N_2$. To implement this linear convolution with an N-point DFT it is clear that $N \geq N_1 + N_2 - 1$ (see Chapter I,section 1.2,subsection 1.2.1). Similarly, the operation of (3.24) could be carried out with an order N G_t matrix provided that H(z) for {h(n)} could be found. Note, however, that $N_2 < N$ and the matrix in (3.24) is Toeplitz and of band width N_2. Thus, the operation in (3.24) is most efficiently implemented directly with an LSA rather than by using an LSA implementation of the equivalent G_t matrix. This raises the question of whether or not DFT spectrum filtering with the use of the G_t matrices is worthwhile. This matter will be treated in the next section.

3.5 COMPARISONS

In this section the results of the four preceding sections will be evaluated in order to ascertain the conditions under which it may be worthwhile to implement DFT spectrum filters using T and G_t rather than F and G_f.

Thompson [43] has shown that the AT^2 complexity of the FFT must grow at least as fast as $\Omega(N^2 \log^2 N)$ (see (3.3)) and so it should be obvious that any algorithm which achieves this lower bound on AT^2 growth should be optimal in some sense. Thompson [44] has shown that the FFT network, perfect shuffle (or shuffle exchange (SE)) network, cube-connected cycles (CCC) and the mesh are all AT^2 optimal in the sense of satisfying (3.3). On the other hand, the cascade and the N-cell DFT have AT^2 complexities of $\Omega(N^3 \log^3 N)$ and so are suboptimal. Why have we chosen to reject the AT^2 optimal designs in favor of the AT^2 suboptimal designs in the considerations of this thesis? The reasons can be summarized as follows:

1. The AT^2 optimal designs have a high asymptotic area complexity, which is why they are fast structures. This high area requirement makes the implementation of these algorithms difficult in practice, even with the best available technology.
2. In a sense, the AT^2 optimal structures are too fast. The fastest is the FFT network with $T = \Omega(\log N)$ and the slowest is the mesh with $T = \Omega(\sqrt{N})$. These time complexities are superior to the $\Omega(N)$ time complexity of the cascade and N-cell DFT. However, it is usually the case in practice that the data to be transformed appear only one component at a time (O(N) time). Thus, the improvement in processing time due to the use of AT^2 optimal structures is not going to be fully utilized. It is therefore wasteful to use these structures in many real-time applications.
3. The AT^2 optimal structures expect all of the data points to be available simultaneously. If N is of even moderate size this implies a chip with a very large number of pins. Chips with a large number of pins are simply not practical.

For the convenience of the reader, equations (3.11),(3.19),(3.14) and (3.23), respectively, are reproduced here as follows:

$$A_c(DFT) = 173{,}000MN + [437{,}000M^2 + 109{,}000M - 317{,}000]log_2N - 235{,}000M \quad (\mu m^2), \quad (3.11)$$

$$A_s(DFT) = [873{,}000M^2 + 562{,}000M - 393{,}000]N$$

$$+873{,}000M^2 - 1{,}130{,}000M - 33{,}600 \quad (\mu m^2), \quad (3.19)$$

$$A_c(DWT) = 72{,}500MN + [201{,}000M - 120{,}000]log_2N - 103{,}000M \quad (\mu m^2), \quad (3.14)$$

$$A_s(DWT) = 177{,}000Nlog_2N + [477{,}000M - 120{,}000]N$$

$$+8{,}260log_2N - 238{,}000M + 60{,}000 \quad (\mu m^2). \quad (3.23)$$

Table 3.2 below shows a table of values for (3.11),(3.19),(3.14) and (3.23) versus N for M = 8,12, and 16 bits.

M	N	$A_c(DFT)$ (mm^2)	$A_s(DFT)$ (mm^2)	$A_c(DWT)$ (mm^2)	$A_s(DWT)$ (mm^2)
8	16	130	1000	14	69
8	32	190	2000	25	140
8	64	260	3900	45	300
8	128	370	7800	84	630
8	256	580	15000	160	1300
8	512	960	31000	310	2700
8	1024	1700	61000	610	5600
12	16	290	2300	22	98
12	32	380	4400	38	200
12	64	510	8600	68	420
12	128	710	17000	130	870
12	256	1000	34000	240	1800
12	512	1600	68000	460	3700
12	1024	2800	140000	910	7500
16	32	650	7700	51	270
16	64	850	15000	91	550
16	128	1100	30000	170	1100
16	256	1600	60000	320	2300
16	512	2400	120000	620	4700
16	1024	4000	240000	1200	9500

Table 3.2: Cascade and N-cell DFT and DWT areas versus N for M = 8,12, and 16 bits. Areas are in square millimeters.

The processing time and delay complexities of (3.7) and (3.16) can be restated, respectively, as follows:

$$T_c = \frac{3}{2}N - 1 , \tag{3.7a}$$

$$D_c = N , \tag{3.7b}$$

$$T_c = 4N - 1 , \tag{3.16a}$$

$$D_c = 2N - 1 . \tag{3.16b}$$

From a comparison of the entries in Table 3.2 with the time complexity results of (3.7) and (3.16) it is obvious that the cascade has a superior area and time performance. Strictly speaking, the AT and AT^2 metrics should be displayed before such a conclusion is drawn. However, the differences between the cascade and the LSA are so enormous that this is totally unnecessary. The processing time of the cascade is less than half that of the LSA and the delay is half that of the LSA. The area required by the cascade for a given N and M is substantially less than that required by the LSA for the same N and M, and this is clearly true for a wide range of N and M. This is particularly true of the comparison between A_c(DFT) and A_s(DFT). The difference in the size of these two numbers is especially great because of the presence of complex multipliers. The cascade uses $O(\log_2 N)$ of them but the LSA needs O(N) complex multipliers, an extravagant requirement. Thus, strictly on the basis of area and time complexity the cascade outperforms the LSA by a very wide margin. This is true of both DFT and DWT implementations.

Why should the LSA be so inefficient at implementing the DFT and DWT (and no doubt the DHT as well) ? The reason is simply that the DFT and DWT matrices have a highly regular structure that can be used to advantage in the design of special purpose hardware with O(N) time complexity. This fact leads naturally to the cascade. The LSA , on the other hand, can be best applied to matrix-by-vector product

problems where the matrix has no significant regularity in its structure. Since the DFT and DWT matrices have a great deal of structure the LSA is fundamentally inappropriate for implementing such matrix-by-vector operations, strictly on the basis of area and time efficiency.

Under what conditions, if any, is it desireable to pay the large area penalty involved in the use of the LSA DFT and DWT ? Given a cascade DFT implementation and an N-cell DFT implementation for a particular N and M, if both designs fit onto a single chip, then the N-cell DFT implementation may be preferable to the cascade DFT implementation. The same might be said for the DWT and DHT. This may be so because the design costs for the N-cell DFT should be lower than that of the cascade. The N-cell DFT is composed of only a very few distinct standard cell types, since each stage of the array is essentially the same as any other stage. For example, there are only two distinct kinds of IPSP cell for the N-cell DFT in Mead and Conway [39, pp. 289-291]. The cascade, on the other hand, has many different cell types. A $\log_2 N$ stage cascade needs $\log_2 N$ different cells. However, the cascade cells are actually quite similar to each other, differing only in the number of delay elements per cell (stage). The N-cell DFT seems to be somewhat more cascadeable than the cascade. This is because the N-cell DFT's IPSPs simply butt together when the cascading of stages is required. The cascade input switching unit (SW1 in Fig. 3.2) must be disabled and bypassed before extra stages can be added to the front end of the cascade. Another consideration is that the LSA chip needs four data ports, if the on-chip coefficient generation scheme in Mead and Conway [39] is used, since each IPSP has four I/O ports and stages must be added to both ends of the LSA. The cascade can do with only three ports (two input, one output). Thus, for a large M, it may be that the cascade is preferable to the LSA since it needs one less I/O port and so needs fewer pins. For small to moderate M this fact is likely to be of little importance.

Thus, it seems that the N-cell DFT may have a marginal advantage over the cascade in terms of designability and cascadeability, although it seems likely that an N-cell DFT is feasible in practice only if it can fit onto a single chip. However, from Table 3.2, this will likely be the case only for small N. The technology on which the values in Table 3.2 are based is very primitive by today's standards (for example, TRW Inc. has a commercial 1μm CMOS process) and so in the future it is possible that the N-cell DFT may be preferable to the cascade DFT as device dimensions shrink to below 1μm and the die size and number of wiring layers increases.

It is probably quite important that the N-cell DFT actually fit onto a single chip. This is at least due to the fact that the cascade DFT accomodates an increase in N from 2^k to 2^{k+1} more readily than does the LSA. In going from $N=2^k$ to 2^{k+1} only one new stage needs to be added to the cascade but the number of IPSP cells must double in order to accomodate the increase in N. This is because the number of cascade stages grows as $\log_2 N$ rather than growing as N in the case of the LSA.

Another very important consideration that will almost certainly count against any suggestion of using the LSA for computing the DFT is the question of yield. If on a wafer there are η fatal flaws per unit of area, the probability that a chip of area A will work can be characterized qualitatively by an expression such as

$$P_0(\eta A) = e^{-\eta A} .$$

Thus, as A goes up, yield goes down for a given η. For a given N, M and η it is clear that the yield for an N-cell DFT chip is likely to be lower (perhaps substantially lower) than for an equivalent cascade DFT chip. This will inevitably increase the cost of production and so reduce the viability of the N-cell DFT in relation to the viability of the cascade DFT. Such yield problems could very easily nullify any benefit that the N-cell DFT has in terms of cascadeability or designability. Clearly, this problem may not be as severe in the case of the N-cell DWT since the area difference between it and the cascade DWT is not as great.

Testability is a consideration as well. However, it seems that neither structure has any special advantage over the other in this regard. Clearly though, this issue should be looked at more closely.

Thus, one can safely conclude from the above discussion that, for the forseeable future at least, the cascade is much better suited to transform implementation than the LSA. Furthermore, from the preceding sections, the G_t matrix filters can only be implemented with the LSA and this is more complex than implementing the filter G_f. Thus, the DFT spectral filtering scheme of Fig. 2.1 is better than that of Fig. 2.2 provided F and F^{-1} are implemented using the cascade DFT. The best way to implement the system of Fig. 2.2 is to implement W and W^{-1} with cascade structures and G_w with an LSA. Clearly then, the system of Fig. 2.2 will be slower and more area consuming than that of Fig. 2.1, in general.

The system of Fig. 2.2 is likely to be better than that of Fig. 2.1 only if the DWT has already been implemented and it is desired to add the capability of doing DFT spectrum filtering. This assumes a sufficiently small N and M to enable fitting G_w onto a single chip. Also, some types of G_f filter give G_w filters of very narrow band width. Clearly, such narrow band width filter matrices may be very easy to implement, and so may be effective competition for the system of Fig. 2.1. The resulting G_w matrix may even be much narrower in band width than that of the matrix in (3.24).

It may be possible to roundoff the coefficients of a G_w filter (or more generally a G_t filter) to such an extent the the IPSPs have a very simple structure and so a large N G_w filter might fit on a single chip. This would cause the system of Fig. 2.2 to be viewed more favorably. The extent to which filter coefficients may be rounded off depends largely upon the "tightness" of the filter specifications. Clearly, such a possibility as this requires much more investigation.

Last of all, it is important to note that if G_t could be factored in such a way that it could be implemented as a cascade or cascade- like structure, then this would

likely make the system of Fig. 2.2 competitive with that of Fig. 2.1 for a much wider range of conditions. The search for an efficient factorization is now of paramount importance.

Chapter IV

MISCELLANEOUS CONSIDERATIONS

This chapter briefly treats a few topics related to DFT spectrum filtering that may be of interest to future researchers. Various open questions are presented in the four sections which follow along with some potential methods for answering them. The topics presented are believed to be of the most immediate concern. The first topic is the problem of finding a computationally efficient factorization of G_t. The second topic concerns the sensitivity of matrix G_t to coefficient rounding and the roundoff noise properties of the cascade and linear systolic array (LSA) architectures. The third topic concerns the problem of how to do two-dimensional DFT spectrum filtering. The fourth topic suggests a study of the properties of G_t for more general T transforms than those considered in this thesis. It is suggested that the representation theory of groups may be useful in this regard.

4.1 FACTORIZING MATRIX G_t

It is fair to say that finding a computationally efficient way of factorizing matrix G_t is the most pressing problem at this time. This follows from the considerations of Chapter III where it is most clearly seen that the existence or otherwise of an efficient factorization for G_t will have an enormous impact on the practicality of the spectral filtering techniques described in this thesis. This is especially true where special purpose hardware implementations are concerned, but it is also true of sequential processor implementations.

It would be very surprising, given the results of Chapter II, to find out that G_t has no efficient factorization in general. A "trivial" factorization of G_t could simply

involve a factorization of the matrices in (2.3b), but this is useless since such a factorization would reduce to the direct use of the filtering scheme of Fig. 2.1 which is described by equation (2.2a). The question is: Is there an alternative to factorizing the matrices of (2.3b) that is more meaningful and useful ?

In the special case of $G_t = G_h$ the 4×4 diagonal block of G_h has the form

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ -x_4 & x_1 & x_2 & x_3 \\ -x_3 & -x_4 & x_1 & x_2 \\ -x_2 & -x_3 & -x_4 & x_1 \end{bmatrix}. \tag{4.1}$$

It is possible to rearrange the rows and columns of (4.1) to get

$$\begin{bmatrix} x_1 & x_3 & x_2 & x_4 \\ -x_3 & x_1 & -x_4 & x_2 \\ -x_4 & x_2 & x_1 & x_3 \\ -x_2 & -x_4 & -x_3 & x_1 \end{bmatrix}. \tag{4.2}$$

Notice that the 2×2 submatrices making up the quadrants of (4.2) can be considered as rotation operators in the same sense as

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{4.3}$$

is a rotation operator in the Cartesian xy-plane. Since the 2×2 submatrices are like rotation operators in the sense of (4.3), a 25% reduction in the number of multiplications is possible , since Winograd [53] has shown how to do the operation

$$\begin{bmatrix} x_1 & -x_2 \\ x_2 & x_1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1y_1 - x_2y_2 \\ x_2y_1 + x_1y_2 \end{bmatrix} \tag{4.4}$$

with only three multiplications and five additions/subtractions. Thus, the number of multiplications needed to compute (4.1) can be reduced from 16 to 12. The structure exemplified by (4.1) and (4.2) appears to hold for diagonal blocks of higher order. Perhaps even greater reductions in the number of multiplications are possible for blocks of higher order.

The structure of the diagonal blocks of G_h is similar in some respects to that of the matrix in (3.24). Thus, the fast linear convolution algorithm of Toom-Cook described in McClellan and Rader [54] may be helpful. Consider the matrix-by-vector product

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_1 & a_2 \\ a_5 & a_4 & a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$= \begin{bmatrix} a_1 & 0 & 0 \\ a_4 & a_1 & 0 \\ a_5 & a_4 & a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & a_2 & a_3 \\ 0 & 0 & a_2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}. \tag{4.5a}$$

The first 3×3 matrix in (4.5a) clearly has the same form as a typical diagonal block of G_h even though it is 3×3. The first term in the second line of (4.5a) is a simple linear convolution which can certainly be solved with the Toom-Cook algorithm [54]. The second term can be reduced to an equivalent form,

$$\begin{bmatrix} a_2 & 0 \\ a_3 & a_2 \end{bmatrix} \begin{bmatrix} x_3 \\ x_2 \end{bmatrix} \tag{4.5b}$$

and the resulting vector can be time-reversed to yield the proper ordering of vector components. Clearly, (4.5b) is also a simple linear convolution. It could be called an anticausal convolution. Thus, matrix-by-vector products involving the diagonal blocks of G_h reduce to an N-point and an N-1-point linear convolution. It is known that the Toom-Cook procedure can perform a N-point linear convolution in as few as 2N-1 nontrivial multiplications. In fact, Winograd [53,54] has proven that this is the minimum possible number. Thus, postmultiplying an order N block of G_h by an N-component vector should actually require at least 4(N-1) nontrivial multiplications. Thus, we would expect that computation should be reduced from N^2 multiplications to as little as 4(N-1) multiplications according to Winograd's theory [53]. We have proven that if N=4, at least 12 multiplications are needed. This shows that (4.1) cannot be reduced in complexity any further than by the method suggested in the

preceding paragraph.

Therefore, the Toom-Cook algorithm and Winograd's complexity theory give some hope of finding a minimal multiplication algorithm. However, the resulting algorithm may be exceedingly complex and messy, especially for large N, and therefore may be of little use in practice.

4.2 COEFFICIENT SENSITIVITY AND ROUNDOFF EFFECTS

What are the effects of rounding off the coefficients in matrix G_t ? What are the roundoff noise properties of the cascade and the LSA architectures ? The answers to these questions are important in any practical implementation of the DFT spectrum filters described in this thesis.

One of the essential features of a linear spectrum filter matrix in some general frequency domain is that the matrix is strictly diagonal. This is because a linear filter merely scales and phase shifts a particular spectral component. The nondiagonal nature of G_t for almost any G_f indicates that G_t is really a type of nonlinear filter. Thus, a linear filter in one kind of frequency domain usually becomes nonlinear in any other frequency domain.

The effects of rounding off the coefficients of G_t could be studied in terms of their effects on G_f, as a change in G_t, called ΔG_t, causes a change in G_f, called ΔG_f, since

$$\Delta G_f = FT^{-1} \Delta G_t \, TF^{-1} . \tag{4.6}$$

Some numerical experiments in this regard show that off-diagonal terms appear in G_f when G_t is rounded off, especially if the rounding is very coarse. This is a nonlinear effect.

In the area of roundoff noise properties of the cascade and the LSA architectures, the following noise models are likely to be useful, at least to a first approximation. Since the cascade implements the DIF FFT butterfly, the butterfly diagram itself

can constitute a noise model for the cascade. Quantizers can be placed at the nodes which are multiplier outputs in the butterfly diagram. This model suggests that the output noise power of the cascade is proportional to $\log_2 N$, the number of stages in the butterfly diagram (also the number of cascade stages). The LSA can be thought of as a transversal filter, with quantizers at the outputs of the multipliers in each IPSP. The multipliers are like the taps in a transversal filter. There are O(N) taps and so the noise power at the output of the LSA can be expected to be proportional to N. Thus, it seems that the LSA has a poorer noise performance than the cascade for a given N and M (word length).

4.3 TWO-DIMENSIONAL DFT SPECTRUM FILTERING

The spectral filtering scheme of this thesis appears useful only in the case of one-dimensional DFT spectrum filtering. Can the method be used to filter two-dimensional signals? The method could be useful in digital image processing applications.

It has already been stated in Chapter II (section 2.2) that Kahveci and Hall [28] briefly considered the two-dimensional problem. In [28] a two-dimensional signal is represented as a matrix rather than as a vector. However, either the columns or the rows of an N×N matrix may be stacked in order to give a vector of N^2 components. Thus, the two-dimensional N×N signal is equivalent to an $N^2 \times 1$ one-dimensional signal (assuming column vectors) and so the two- dimensional DFT spectrum filtering problem can be reduced to an equivalent one-dimensional DFT spectrum filtering problem. However, the vector equivalent of the N×N matrix is N times bigger than an N-component vector and so N different G_f matrices (N×N) are needed to filter the $N^2 \times 1$ vector. This implies that there will be N G_t matrices in the T transform domain as well. Clearly, the two-dimensional problem is tougher than the one-dimensional problem, especially if the G_t matrices cannot be efficiently factorized.

4.4 THE USE OF OTHER TRANSFORMS

This thesis considered only certain special cases for T, one of which was quite trivial. How can the results of Chapter II in this thesis be generalized in a meaningful way to other T? The representation theory of groups may be useful in this problem.

In Apple and Wintz [55] and Cairns [56] the problem of fast transforms on finite Abelian groups is examined. Such considerations lead to the FFT and FWT. It turns out that the character table of an order N (not necessarily a positive integer power of two now) cyclic group (cyclic groups are Abelian [35]) can be regarded as the DFT matrix. The dyadic group (Rosenbloom [57]) has a character table which can be regarded as the DWT matrix. Karpovsky [58] considered transforms on non-Abelian groups. Non-Abelian groups have irreducible unitary representation tables with matrix entries, while the irreducible representation tables of Abelian groups always contain only scalar entries. This leads to the fact that transforms on non-Abelian groups give matrix valued spectral components, while transforms on Abelian groups give rise only to scalar valued spectral components. Transforms on non-Abelian groups are little known and little used. The only studies of these transforms known to the author of this thesis are by Karpovsky [58] and Karpovsky and Trachtenberg [59]. In [55],[56], and [58] it is demonstrated that if a group G can be factored into a direct product of smaller groups then any transform defined on G will have a fast computational algorithm very like the FFT and the FWT.

It may be that group theory could be used to construct fairly arbitrary T transforms and the structure of the resulting G_t matrices could be studied under the general framework of group theory. This might lead to interesting results, especially in the case of transforms on non-Abelian groups.

Chapter V

CONCLUSIONS

This thesis has studied a class of nonrecursive filters that are used to filter DFT spectra: the DFT spectrum filters. Such filtering operations amount to a form of periodic convolution. As well, the VLSI implementation of such filters with the $\theta(N)$ time complexity hardware algorithms, known as the radix 2, pipeline and the linear systolic array, were considered. The principal conclusions of this thesis are:

1. If the Fourier gain matrix (G_f) is that of a linear filter then the T transform domain filter (G_t) will be real and block-diagonal in general, provided that T is a real matrix.

2. The structure of G_t for T = W (DWT matrix) and T = H (DHT matrix) is such that DFT spectrum filtering using T and G_t is computationally more efficient on a sequential processor than DFT spectrum filtering using F (DFT matrix) and G_f for $N \leq 64$, where N is the number of components in the real signal vector.

3. If the elements on the main diagonal of G_f, called g_i, satisfy $g_i \in \mathbf{R}$ for all i and $g_i = g_{N-i}$ for i = 1,2,...,N/2-1, then DFT spectrum filtering with T = W and $G_t = G_w$ is computationally more efficient on a sequential processor than DFT spectrum filtering with F and G_f for $N \leq 128$.

4. Linear systolic arrays are generally not suitable for the implementation of transforms such as the DFT, DWT and DHT, since the radix 2, pipeline can be used to implement them with $O(\log_2 N)$ processing stages instead of the O(N) stages needed by the linear systolic array. The radix 2, pipeline is also faster by a constant factor than the linear systolic array.

5. Because G_t has no known computationally efficient factorization (other than the trivial one of Chapter IV) that can be implemented with a radix 2, pipeline structure (or something very similar), it must be implemented with a linear systolic array.

6. A special purpose hardware implementation of T as a radix 2, pipeline structure and G_t as a linear systolic array is not as area and time efficient as a cascade implementation of F and a complex multiplier implementation of G_f. Therefore, DFT spectrum filters using T and G_t are not as readily implemented with VLSI methods as are DFT spectrum filters using F and G_f, except in certain special cases.

7. DFT spectrum filtering with a linear systolic array implementation of G_t may be worthwhile if the entire filter can fit onto a single silicon chip, or if the structure of G_t is particularly simple regardless of how big N is.

Appendix A

TYPICAL ELEMENT OF THE A-MATRIX

Here it is shown that (2.10) is true. Let

$$f_{rn} = \exp\left(-j\frac{2\pi nr}{N}\right), r,n = 0,1,...,N-1, \tag{A1}$$

be a typical element of F and let

$$\bar{f}_{rn} = \frac{1}{N}\exp\left(j\frac{2\pi rn}{N}\right), \tag{A2}$$

for the same range of r and n values, be the typical element of F^{-1}. Thus, $F^{-1} = F^{*}/N$ and $F = F^{T}, F^{-1} = (F^{-1})^{T}$ as well. Using (A1) and (A2) and the definition of matrix multiplication yields

$$A = F^{-1}G_f F = \left[\sum_{i=0}^{N-1} \bar{f}_{ni} g_i f_{im}\right]. \tag{A3}$$

Substituting (A1) and (A2) into (A3) clearly yields (2.10).

Appendix B

PROPERTIES OF MATRIX C

We first show that $\tilde{A}_N^{(k)}$ is Hermitian. Clearly then, B will be Hermitian as well since it is in quadrants two and four of $\tilde{A}_N^{(k)}$. A typical element of $\tilde{A}_N^{(k)}$ is

$$\tilde{a}_{nm}^{(k)} = \exp\left(j\frac{2\pi k}{N}(n-m)\right) \tag{B1}$$

from which it is easy to see that $\tilde{a}_{nm}^{(k)}=\tilde{a}_{mn}^{*(k)}$ which implies that $\tilde{A}_N^{(k)}=(\tilde{A}_N^{(k)*})^T$, or $\tilde{A}_N^{(k)}$ is Hermitian.

It is now necessary to show that $C=(C^*)^T$ and that $C=exp(j\pi k)B=(-1)^k B$ in (2.13). Begin by changing the indexing scheme used to represent typical elements in each of the four quadrants of $\tilde{A}_N^{(k)}$ as follows:

$$Quadrant-1) \quad \tilde{a}_{nm}^{(k)} = \exp\left(j\frac{2\pi k}{N}(n-\frac{N}{2}-r)\right)$$

$$r = 0,1,...,\frac{N}{2}-1,\ n = 0,1,...,\frac{N}{2}-1 \text{ and } m = \frac{N}{2}+r \tag{B2}$$

$$Quadrant-2) \quad \tilde{a}_{nm}^{(k)} = \exp\left(j\frac{2\pi k}{N}(n-m)\right)$$

$$n = 0,1,...,\frac{N}{2}-1,\ m = 0,1,...,\frac{N}{2}-1 \tag{B3}$$

$$Quadrant-3) \quad \tilde{a}_{nm}^{(k)} = \exp\left(j\frac{2\pi k}{N}(\frac{N}{2}+r-m)\right)$$

$$r = 0,1,...,\frac{N}{2}-1,\ m = 0,1,...,\frac{N}{2}-1, \text{ and } n = \frac{N}{2}+r \tag{B4}$$

$$Quadrant-4) \quad \tilde{a}_{nm}^{(k)} = \exp\left(j\frac{2\pi k}{N}(r-p)\right)$$

$$r = 0,1,...,\frac{N}{2}-1,\ p = 0,1,...,\frac{N}{2}-1,\ n = \frac{N}{2}+r\text{, and } m = \frac{N}{2}+p \tag{B5}$$

We can write (B4) as

$$\tilde{a}_{nm}^{(k)} = exp(j\pi k)\exp\left(j\frac{2\pi k}{N}(r-m)\right) \tag{B6}$$

which allows one to write that C = exp(jπk)B since (B6) is merely (B3) multiplied by exp(jπk). (B4) represents a typical element of C. (B2) represents a typical element of $(C^*)^T$. It is possible to write $(C^*)^T = \exp(-j\pi k)(B^*)^T = \exp(-j\pi k)B = \exp(j\pi k)B = C$ because $\tilde{A}_N^{(k)}$ is Hermitian- Toeplitz.

Appendix C

THE METHOD OF TADOKORO AND HIGUCHI

Tadokoro and Higuchi [14,15] have considered the samples of a signal to be $f(\theta_i)$ and

$$f(\theta_i) = a_0 + \sum_{k=1}^{\frac{N}{2}-1} [a_k \cos k\theta_i + b_k \sin k\theta_i] + b_{N/2} \sin (N/2)\theta_i \ , \qquad \text{(C1)}$$

where $N = 2^M$ and $\theta_i = \frac{2\pi i}{N} - \frac{\pi}{N}$, $i = 1,2,...,N$. This is equation (8) in [14]. Rewrite (C1) as

$$f(m) = a_0 + \sum_{n=1}^{\frac{N}{2}-1} [a_n \cos n\theta_m + b_n \sin n\theta_m] + b_{N/2} \sin (N/2)\theta_m \ , \qquad \text{(C2)}$$

where now $\theta_m = \frac{2\pi m}{N} + \frac{\pi}{N}$ and $m = 0,1,...,N-1$ (using i = m+1).

Let

$$f_2(m) = \sum_{n=0}^{N-1} \hat{c}_n W^{nm} \ , m = 0,1,...,N-1 \ , \qquad \text{(C3)}$$

where W = exp(j2π/N). The $\hat{c}_n$ are spectral components (the true DFT coefficients) and they satisfy $\hat{c}_0, \hat{c}_{N/2} \in \mathbf{R}$ and $\hat{c}_i = \hat{c}^*_{N-i}$ for i=1,2,...,N/2-1. Note that $b_{N/2} \sin (N/2)\theta_m = b_{N/2} \sin(\pi m + \frac{\pi}{2}) = (-1)^m b_{N/2}$. Using (C3) and the fact that $\hat{c}_i = \hat{c}^*_{N-i}$ gives

$$\hat{f}(m) = \hat{c}_0 + \sum_{n=1}^{\frac{N}{2}-1} \left[\hat{c}_n e^{j\frac{2\pi nm}{N}} \right] + \sum_{n=\frac{N}{2}+1}^{N-1} \left[\hat{c}_n e^{j\frac{2\pi nm}{N}} \right] + \hat{c}_{\frac{N}{2}} e^{j\pi m}$$

$$= \hat{c}_0 + \sum_{n=1}^{\frac{N}{2}-1}\left[\hat{c}_n e^{j\frac{2\pi nm}{N}} + \hat{c}_n^* e^{-j\frac{2\pi nm}{N}}\right] + (-1)^m \hat{c}_{\frac{N}{2}} \ . \qquad \text{(C4)}$$

Now let

$$\hat{c}_n = W^{\frac{n}{2}} \frac{a_n - jb_n}{2} \ so \ \hat{c}_n^* = W^{-\frac{n}{2}} \frac{a_n + jb_n}{2} \ . \qquad \text{(C5)}$$

Substitute (C5) into (C4) in order to get

$$\hat{f}(m) = \hat{c}_0 + \sum_{n=1}^{\frac{N}{2}-1}\left[a_n \cos\left(\frac{2\pi nm}{N} + \frac{n\pi}{N}\right) + b_n \sin\left(\frac{2\pi nm}{N} + \frac{n\pi}{N}\right)\right] + (-1)^m \hat{c}_{\frac{N}{2}} \ , \qquad \text{(C6)}$$

but this is identical to (C2) if

$$\hat{c}_0 = a_0 \ , \qquad \text{(C7a)}$$

$$\hat{c}_{N/2} = b_{N/2} \ , \qquad \text{(C7b)}$$

$$\hat{c}_n = W^{\frac{n}{2}} \frac{a_n - jb_n}{2} \ , n = 1,2,..,\frac{N}{2}-1 \ , \qquad \text{(C7c)}$$

$$\hat{c}_{N-n} = \hat{c}_n^* \ . \qquad \text{(C7d)}$$

Thus, (C1) is the inverse DFT "in disguise" and to get the true DFT coefficients one would use (C7).

Appendix D

TYPICAL ELEMENT OF $G_w^{(k)}$

Here it is shown that (2.27) is true. Using (2.24) and (2.12),

$$A^{(k)}W = \left[\sum_{p=0}^{N-1} a_{np}^{(k)} w_{pm}\right]$$

$$= \left[\frac{1}{N} g_k \sum_{p=0}^{N-1} (-1)^{\sum_{l=0}^{q-1} b_l(p) b_l(m)} \exp\left(j\frac{2\pi k}{N}(n-p)\right)\right]. \qquad \text{(D1)}$$

Let $A^{(k)}W = [\alpha_{nm}^{(k)}]$ so

$$\frac{1}{N} W A^{(k)} W = \left[\frac{1}{N} \sum_{r=0}^{N-1} w_{nr}\, \alpha_{rm}^{(k)}\right]$$

$$= \left[\frac{1}{N^2} g_k \sum_{r=0}^{N-1} \sum_{p=0}^{N-1} (-1)^{\sum_{l=0}^{q-1} [b_l(n) b_l(r) + b_l(p) b_l(m)]} \exp\left(j\frac{2\pi k}{N}(r-p)\right)\right], \qquad \text{(D2)}$$

but this is $\gamma_{nm}^{(k)}$ in (2.27). Note that in each of the first lines of (D1) and (D2), the definition of matrix multiplication has been used.

Appendix E

EXAMPLES OF PARTIAL GAIN MATRICES

The following matrices are examples of partial gain matrices for $G_w^{(k)}$, $G_h^{(k)}$, and $G_{t_r}^{(k)}$ for N = 8. It is assumed that $g_i = 1$ for all i. We can define $G_w^{(k)} = g_k \tilde{G}_w^{(k)}$ and so the matrices displayed below are $\tilde{G}_w^{(k)}$ matrices.

THE PARTIAL WALSH GAIN MATRICES ARE:

PARTIAL WALSH GAIN MATRIX GW(0)
REAL PART

1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL WALSH GAIN MATRIX GW(1)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.427	0.000	0.000	-0.177
0.000	0.000	0.000	0.000	0.000	0.073	0.177	0.000
0.000	0.000	0.000	0.000	0.000	0.177	0.427	0.000
0.000	0.000	0.000	0.000	-0.177	0.000	0.000	0.073

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.177	0.427	0.000
0.000	0.000	0.000	0.000	-0.177	0.000	0.000	0.073
0.000	0.000	0.000	0.000	-0.427	0.000	0.000	0.177
0.000	0.000	0.000	0.000	0.000	-0.073	-0.177	0.000

PARTIAL WALSH GAIN MATRIX GW(2)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000
0.000	0.000	-0.500	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL WALSH GAIN MATRIX GW(3)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.073	0.000	0.000	0.177
0.000	0.000	0.000	0.000	0.000	0.427	-0.177	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	0.073	0.000
0.000	0.000	0.000	0.000	0.177	0.000	0.000	0.427

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.177	-0.073	0.000
0.000	0.000	0.000	0.000	-0.177	0.000	0.000	-0.427
0.000	0.000	0.000	0.000	0.073	0.000	0.000	0.177
0.000	0.000	0.000	0.000	0.000	0.427	-0.177	0.000

PARTIAL WALSH GAIN MATRIX GW(4)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL WALSH GAIN MATRIX GW(5)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.073	0.000	0.000	0.177
0.000	0.000	0.000	0.000	0.000	0.427	-0.177	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	0.073	0.000
0.000	0.000	0.000	0.000	0.177	0.000	0.000	0.427

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	0.073	0.000
0.000	0.000	0.000	0.000	0.177	0.000	0.000	0.427
0.000	0.000	0.000	0.000	-0.073	0.000	0.000	-0.177
0.000	0.000	0.000	0.000	0.000	-0.427	0.177	0.000

PARTIAL WALSH GAIN MATRIX GW(6)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	-0.500	0.000	0.000	0.000	0.000
0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL WALSH GAIN MATRIX GW(7)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.427	0.000	0.000	-0.177
0.000	0.000	0.000	0.000	0.000	0.073	0.177	0.000
0.000	0.000	0.000	0.000	0.000	0.177	0.427	0.000
0.000	0.000	0.000	0.000	-0.177	0.000	0.000	0.073

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	-0.427	0.000
0.000	0.000	0.000	0.000	0.177	0.000	0.000	-0.073
0.000	0.000	0.000	0.000	0.427	0.000	0.000	-0.177
0.000	0.000	0.000	0.000	0.000	0.073	0.177	0.000

THE PARTIAL HAAR GAIN MATRICES ARE:

PARTIAL HAAR GAIN MATRIX GH(0)

REAL PART

1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL HAAR GAIN MATRIX GH(1)

REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.250	0.177	0.000	-0.177
0.000	0.000	0.000	0.000	0.177	0.250	0.177	0.000
0.000	0.000	0.000	0.000	0.000	0.177	0.250	0.177
0.000	0.000	0.000	0.000	-0.177	0.000	0.177	0.250

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	-0.250	-0.177
0.000	0.000	0.000	0.000	0.177	0.000	-0.177	-0.250
0.000	0.000	0.000	0.000	0.250	0.177	0.000	-0.177
0.000	0.000	0.000	0.000	0.177	0.250	0.177	0.000

PARTIAL HAAR GAIN MATRIX GH(2)

REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	-0.500	0.000	0.000	0.000	0.000
0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL HAAR GAIN MATRIX GH(3)

REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.250	-0.177	0.000	0.177
0.000	0.000	0.000	0.000	-0.177	0.250	-0.177	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	0.250	-0.177
0.000	0.000	0.000	0.000	0.177	0.000	-0.177	0.250

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	0.250	-0.177
0.000	0.000	0.000	0.000	0.177	0.000	-0.177	0.250
0.000	0.000	0.000	0.000	-0.250	0.177	0.000	-0.177
0.000	0.000	0.000	0.000	0.177	-0.250	0.177	0.000

PARTIAL HAAR GAIN MATRIX GH(4)

REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL HAAR GAIN MATRIX GH(5)
REAL PART

```
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.250-0.177 0.000 0.177
0.000 0.000 0.000 0.000-0.177 0.250-0.177 0.000
0.000 0.000 0.000 0.000 0.000-0.177 0.250-0.177
0.000 0.000 0.000 0.000 0.177 0.000-0.177 0.250
```

IMAGINARY PART

```
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.177-0.250 0.177
0.000 0.000 0.000 0.000-0.177 0.000 0.177-0.250
0.000 0.000 0.000 0.000 0.250-0.177 0.000 0.177
0.000 0.000 0.000 0.000-0.177 0.250-0.177 0.000
```

PARTIAL HAAR GAIN MATRIX GH(6)

REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000
0.000	0.000	-0.500	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL HAAR GAIN MATRIX GH(7)

REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.250	0.177	0.000	-0.177
0.000	0.000	0.000	0.000	0.177	0.250	0.177	0.000
0.000	0.000	0.000	0.000	0.000	0.177	0.250	0.177
0.000	0.000	0.000	0.000	-0.177	0.000	0.177	0.250

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.177	0.250	0.177
0.000	0.000	0.000	0.000	-0.177	0.000	0.177	0.250
0.000	0.000	0.000	0.000	-0.250	-0.177	0.000	0.177
0.000	0.000	0.000	0.000	-0.177	-0.250	-0.177	0.000

THE PARTIAL TRIDIAGONAL GAIN MATRICES ARE:

PARTIAL TRIDIAGONAL GAIN MATRIX GT(0)
REAL PART

0.250	0.250	0.250	0.250	0.000	0.000	0.000	0.000
0.250	0.250	0.250	0.250	0.000	0.000	0.000	0.000
0.250	0.250	0.250	0.250	0.000	0.000	0.000	0.000
0.250	0.250	0.250	0.250	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL TRIDIAGONAL GAIN MATRIX GT(1)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.250	0.177	0.000	-0.177
0.000	0.000	0.000	0.000	0.177	0.250	0.177	0.000
0.000	0.000	0.000	0.000	0.000	0.177	0.250	0.177
0.000	0.000	0.000	0.000	-0.177	0.000	0.177	0.250

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	-0.250	-0.177
0.000	0.000	0.000	0.000	0.177	0.000	-0.177	-0.250
0.000	0.000	0.000	0.000	0.250	0.177	0.000	-0.177
0.000	0.000	0.000	0.000	0.177	0.250	0.177	0.000

PARTIAL TRIDIAGONAL GAIN MATRIX GT(2)
REAL PART

0.250	0.000	-0.250	0.000	0.000	0.000	0.000	0.000
0.000	0.250	0.000	-0.250	0.000	0.000	0.000	0.000
-0.250	0.000	0.250	0.000	0.000	0.000	0.000	0.000
0.000	-0.250	0.000	0.250	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	-0.250	0.000	0.250	0.000	0.000	0.000	0.000
0.250	0.000	-0.250	0.000	0.000	0.000	0.000	0.000
0.000	0.250	0.000	-0.250	0.000	0.000	0.000	0.000
-0.250	0.000	0.250	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL TRIDIAGONAL GAIN MATRIX GT(3)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.250	-0.177	0.000	0.177
0.000	0.000	0.000	0.000	-0.177	0.250	-0.177	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	0.250	-0.177
0.000	0.000	0.000	0.000	0.177	0.000	-0.177	0.250

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	0.250	-0.177
0.000	0.000	0.000	0.000	0.177	0.000	-0.177	0.250
0.000	0.000	0.000	0.000	-0.250	0.177	0.000	-0.177
0.000	0.000	0.000	0.000	0.177	-0.250	0.177	0.000

PARTIAL TRIDIAGONAL GAIN MATRIX GT(4)
REAL PART

0.250	-0.250	0.250	-0.250	0.000	0.000	0.000	0.000
-0.250	0.250	-0.250	0.250	0.000	0.000	0.000	0.000
0.250	-0.250	0.250	-0.250	0.000	0.000	0.000	0.000
-0.250	0.250	-0.250	0.250	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL TRIDIAGONAL GAIN MATRIX GT(5)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.250	-0.177	0.000	0.177
0.000	0.000	0.000	0.000	-0.177	0.250	-0.177	0.000
0.000	0.000	0.000	0.000	0.000	-0.177	0.250	-0.177
0.000	0.000	0.000	0.000	0.177	0.000	-0.177	0.250

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.177	-0.250	0.177
0.000	0.000	0.000	0.000	-0.177	0.000	0.177	-0.250
0.000	0.000	0.000	0.000	0.250	-0.177	0.000	0.177
0.000	0.000	0.000	0.000	-0.177	0.250	-0.177	0.000

PARTIAL TRIDIAGONAL GAIN MATRIX GT(6)
REAL PART

0.250	0.000	-0.250	0.000	0.000	0.000	0.000	0.000
0.000	0.250	0.000	-0.250	0.000	0.000	0.000	0.000
-0.250	0.000	0.250	0.000	0.000	0.000	0.000	0.000
0.000	-0.250	0.000	0.250	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

IMAGINARY PART

0.000	0.250	0.000	-0.250	0.000	0.000	0.000	0.000
-0.250	0.000	0.250	0.000	0.000	0.000	0.000	0.000
0.000	-0.250	0.000	0.250	0.000	0.000	0.000	0.000
0.250	0.000	-0.250	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

PARTIAL TRIDIAGONAL GAIN MATRIX GT(7)
REAL PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.250	0.177	0.000	-0.177
0.000	0.000	0.000	0.000	0.177	0.250	0.177	0.000
0.000	0.000	0.000	0.000	0.000	0.177	0.250	0.177
0.000	0.000	0.000	0.000	-0.177	0.000	0.177	0.250

IMAGINARY PART

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.177	0.250	0.177
0.000	0.000	0.000	0.000	-0.177	0.000	0.177	0.250
0.000	0.000	0.000	0.000	-0.250	-0.177	0.000	0.177
0.000	0.000	0.000	0.000	-0.177	-0.250	-0.177	0.000

Appendix F

EXAMPLES OF GAIN MATRICES

The following three matrices are examples of gain matrices for G_w, G_h, and G_{t_r} for N = 16. The prototype filter is a first order low pass Butterworth filter corresponding to a sampling rate of 100 Hz and a digital cutoff frequency of 62.8 rads/sec (10 Hz). The g_i values for the G_f matrix are specified in the following table. Note that only the g_i for i = 0,1,...,N/2-1 are shown since $g_i = g_{N-i}^*$ holds for i = 0,1,...,N/2-1.

i	g_i	i	g_i
0	1.000	5	0.045-0.207j
1	0.727-0.445j	6	0.018-0.132j
2	0.381-0.486j	7	0.004-0.064j
3	0.191-0.393j	8	0.000
4	0.095-0.294j		

THE GW MATRIX IS:

1.0	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.10	.29	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	-.29	.10	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.33	.22	.40	-.13	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	-.22	.07	.13	-.04	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	-.40	.13	.33	.22	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	-.13	.04	-.22	.07	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.00	.00	.00	.00	.62	.12	.22	-.07	.33	-.11	-.20	.06
.00	.00	.00	.00	.00	.00	.00	.00	-.12	.04	.07	-.02	.11	-.04	-.06	.02
.00	.00	.00	.00	.00	.00	.00	.00	-.22	.07	.23	.25	.20	-.06	-.12	.04
.00	.00	.00	.00	.00	.00	.00	.00	-.07	.02	-.25	.08	.06	-.02	-.04	.01
.00	.00	.00	.00	.00	.00	.00	.00	-.33	.11	.20	-.06	.62	.12	.22	-.07
.00	.00	.00	.00	.00	.00	.00	.00	-.11	.04	.06	-.02	-.12	.04	.07	-.02
.00	.00	.00	.00	.00	.00	.00	.00	-.20	.06	.12	-.04	-.22	.07	.23	.25
.00	.00	.00	.00	.00	.00	.00	.00	-.06	.02	.04	-.01	-.07	.02	-.25	.08

THE GH MATRIX IS:

1.0	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.10	-.29	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.29	.10	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.20	-.09	-.18	-.35	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.35	.20	-.09	-.18	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.18	.35	.20	-.09	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.09	.18	.35	.20	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.00	.00	.00	.00	.24	-.01	-.01	-.02	-.05	-.10	-.19	-.37
.00	.00	.00	.00	.00	.00	.00	.00	.37	.24	-.01	-.01	-.02	-.05	-.10	-.19
.00	.00	.00	.00	.00	.00	.00	.00	.19	.37	.24	-.01	-.01	-.02	-.05	-.10
.00	.00	.00	.00	.00	.00	.00	.00	.10	.19	.37	.24	-.01	-.01	-.02	-.05
.00	.00	.00	.00	.00	.00	.00	.00	.05	.10	.19	.37	.24	-.01	-.01	-.02
.00	.00	.00	.00	.00	.00	.00	.00	.02	.05	.10	.19	.37	.24	-.01	-.01
.00	.00	.00	.00	.00	.00	.00	.00	.01	.02	.05	.10	.19	.37	.24	-.01
.00	.00	.00	.00	.00	.00	.00	.00	.01	.01	.02	.05	.10	.19	.37	.24

THE GT MATRIX IS:

.25	.01	.01	.03	.05	.10	.19	.37	.00	.00	.00	.00	.00	.00	.00	.00
.37	.25	.01	.01	.03	.05	.10	.19	.00	.00	.00	.00	.00	.00	.00	.00
.19	.37	.25	.01	.01	.03	.05	.10	.00	.00	.00	.00	.00	.00	.00	.00
.10	.19	.37	.25	.01	.01	.03	.05	.00	.00	.00	.00	.00	.00	.00	.00
.05	.10	.19	.37	.25	.01	.01	.03	.00	.00	.00	.00	.00	.00	.00	.00
.03	.05	.10	.19	.37	.25	.01	.01	.00	.00	.00	.00	.00	.00	.00	.00
.01	.03	.05	.10	.19	.37	.25	.01	.00	.00	.00	.00	.00	.00	.00	.00
.01	.01	.03	.05	.10	.19	.37	.25	.00	.00	.00	.00	.00	.00	.00	.00
.00	.00	.00	.00	.00	.00	.00	.00	.24	-.01	-.01	-.02	-.05	-.10	-.19	-.37
.00	.00	.00	.00	.00	.00	.00	.00	.37	.24	-.01	-.01	-.02	-.05	-.10	-.19
.00	.00	.00	.00	.00	.00	.00	.00	.19	.37	.24	-.01	-.01	-.02	-.05	-.10
.00	.00	.00	.00	.00	.00	.00	.00	.10	.19	.37	.24	-.01	-.01	-.02	-.05
.00	.00	.00	.00	.00	.00	.00	.00	.05	.10	.19	.37	.24	-.01	-.01	-.02
.00	.00	.00	.00	.00	.00	.00	.00	.02	.05	.10	.19	.37	.24	-.01	-.01
.00	.00	.00	.00	.00	.00	.00	.00	.01	.02	.05	.10	.19	.37	.24	-.01
.00	.00	.00	.00	.00	.00	.00	.00	.01	.01	.02	.05	.10	.19	.37	.24

Appendix G

PARTIAL GAIN MATRIX PROGRAMS

The following three PASCAL programs GWGEN, GHGEN and TRGEN generate the partial gain matrices $G_w^{(k)}$, $G_h^{(k)}$, and $G_{t_r}^{(k)}$, respectively. The programs assume that $g_i = 1$ for all i values.

```
PROGRAM GWGEN(IMAGEIN,IMAGEOUT);
  (* THIS PROGRAM GENERATES ONE-DIMENSIONAL WALSH    *)
  (* GAIN MATRICES.  IT IS USED TO FIND THE GAIN     *)
  (* MATRIX IN THE WALSH DOMAIN, EQUIVALENT TO THAT  *)
  (* IN THE FOURIER DOMAIN.  THIS IS DONE IN THIS    *)
  (* PROGRAM BY AN IMPLEMENTATION OF THE MATRIX      *)
  (* EQUATION,                                       *)
  (*                    1     -1                     *)
  (*             GW =  - W F   GF F W                *)
  (*                   N                             *)
  (* WHERE,  GW = WALSH GAIN MATRIX (NON-DIAGONAL),  *)
  (*         GF = FOURIER GAIN MATRIX (DIAGONAL),    *)
  (*         W  = WALSH MATRIX OF THE TRANSFORM, &   *)
  (*         F  = FOURIER MATRIX OF THE TRANSFORM    *)
  (* GW MAY BE COMPLEX IF GF IS PHYSICALLY NOT       *)
  (* REALIZABLE.  NOTE,                              *)
  (*    GF = DIAG(G(0),G(1),...,G(I),...,G(N-1))     *)
  (* SO G(I) IS THE ITH DIAGONAL ELEMENT OF GF.  WE  *)
  (* IMPLEMENT THE ABOVE FORMULA FOR GW BY A TWO-D    *)
  (* FAST WALSH TRANSFORM OF COMPLEX EXPONENTIAL     *)
  (* FUNCTIONS SCALED BY G(I)/(N**2).  THUS,         *)
  (*                                                 *)
  (*  GW(I)=TWOD[(G(I)/N**2)*EXP(2*J*I*PI*(R-P)/N)]  *)
  (*                                                 *)
  (* WHERE, I = 0,1,2,...,N-1,                       *)
  (*        J = SQRT(-1),                            *)
  (*        PI= 3.14152654, &                        *)
  (*        R,P = COMPLEX SPATIAL DOMAIN INDICES.    *)
  (* THE TOTAL WALSH GAIN MATRIX, GW, IS             *)
```

```
    (*                  N-1                                    *)
    (*            GW =  SUM GW(I)                              *)
    (*                  I=0                                    *)

 CONST
    SIZE=64;
    PI=3.141592654;
 TYPE
    DECODED=ARRAY[1..SIZE,1..SIZE] OF INTEGER;
    SPECTRUM=ARRAY[1..SIZE,1..SIZE] OF REAL;
    COMPLEX=ARRAY[1..2,1..1024] OF REAL;
    COMSPEC=ARRAY[1..2,1..SIZE,1..SIZE] OF REAL;
 VAR
    FLAGY,FLEG,IX,JX,NN,IL:INTEGER;
    IMAGEIN,IMAGEOUT:TEXT;
    DECIMG:DECODED;
    FWT:SPECTRUM;FF:COMPLEX;
    GWI:COMSPEC;


 PROCEDURE HFWT(VAR F:COMPLEX;LN:INTEGER);
    (* THIS PROCEDURE FINDS THE FAST WALSH TRANSFORM ,OR *)
    (* THE INVERSE, AND PRODUCES THE HADAMARD OR         *)
    (* NATURAL ORDERING OF SPECTRAL COMPONENTS.  THIS    *)
    (* PROGRAM IS ADAPTED FROM PAGE 95 OF THE BOOK:      *)
    (*        DIGITAL IMAGE PROCESSING                   *)
    (*        BY R. C. GONZALES AND P. WINTZ             *)
    (*        PUBLISHED BY ADDISON-WESLEY, 1977          *)
    (*TYPE DECLARATION,                                  *)
    (*   TYPE                                           *)
    (*     COMPLEX=ARRAY[1..2,1..1024] OF REAL;         *)
    (*THE USER'S MAIN ROUTINE PASSES THE BASE-2 LOG OF   *)
    (*THE SAMPLE SEQUENCE LENGTH AND A 2X1024 REAL ARRAY *)
    (*WHICH IS THE SET OF COMPLEX NUMBER SAMPLES.  ROW-1 *)
    (*IS THE X-COORDINATE AND ROW-2 IS THE Y-COORDINATE. *)
    (*COLUMN-1 IS THE X-Y VALUE OF THE 1ST SEQUENCE      *)
    (*SAMPLE.  HFWT RETURNS A 2X1024 ARRAY OF FWT        *)
    (*VALUES (COMPLEX), IN HADAMARD ORDER (NATURAL).     *)
 CONST
    PI=3.141592654;
 VAR
    I,J,L,N,LE,LE1,IP:INTEGER;
```

```
    T1,T2:REAL;
  BEGIN (* HFWT *)
  N:=POWER(2,LN);
  FOR L:=1 TO LN DO
    BEGIN
    LE:=POWER(2,L);
    LE1:=LE DIV 2;
    FOR J:=1 TO LE1 DO
      BEGIN
      I:=J;
      WHILE I<=N DO
        BEGIN
        IP:=I+LE1;
        T1:=F[1,IP];T2:=F[2,IP];
        F[1,IP]:=F[1,I]-T1;F[2,IP]:=F[2,I]-T2;
        F[1,I]:=F[1,I]+T1;F[2,I]:=F[2,I]+T2;
        I:=I+LE;
        END;
      END;
    END;  (* END OF IN-PLACE COMPUTATION OF FWT PROCESS *)
END;  (* HFWT *)


PROCEDURE PFWT(VAR F:COMPLEX;LN:INTEGER);
    (* THIS PROCEDURE FINDS THE FAST WALSH TRANSFORM ,OR *)
    (* THE INVERSE, AND PRODUCES THE PALEY OR            *)
    (* BINARY  ORDERING OF SPECTRAL COMPONENTS.  THIS    *)
    (* PROGRAM IS ADAPTED FROM PAGE 95 OF THE BOOK:      *)
    (*        DIGITAL IMAGE PROCESSING                   *)
    (*        BY R. C. GONZALES AND P. WINTZ             *)
    (*        PUBLISHED BY ADDISON-WESLEY, 1977          *)
    (*TYPE DECLARATION,                                  *)
    (*   TYPE                                           *)
    (*     COMPLEX=ARRAY[1..2,1..1024] OF REAL;         *)
    (*THE USER'S MAIN ROUTINE PASSES THE BASE-2 LOG OF   *)
    (*THE SAMPLE SEQUENCE LENGTH AND A 2X1024 REAL ARRAY *)
    (*WHICH IS THE SET OF COMPLEX NUMBER SAMPLES.  ROW-1 *)
    (*IS THE X-COORDINATE AND ROW-2 IS THE Y-COORDINATE. *)
    (*COLUMN-1 IS THE X-Y VALUE OF THE 1ST SEQUENCE      *)
    (*SAMPLE.  HFWT RETURNS A 2X1024 ARRAY OF FWT        *)
    (*VALUES (COMPLEX), IN PALEY ORDER (BINARY).         *)
VAR
```

```
    I,J,K,L,NV2,NM1,N,LE,LE1,FLG,IP:INTEGER;
    T1,T2:REAL;
  BEGIN (* PFWT *)
  N:=POWER(2,LN);
  NV2:=N DIV 2;
  NM1:=N-1;
  J:=1;
  FOR I:=1 TO NM1 DO
    BEGIN
    IF I>=J THEN
      BEGIN
      K:=NV2;
      END
    ELSE
      BEGIN
      T1:=F[1,J];T2:=F[2,J];
      F[1,J]:=F[1,I];F[2,J]:=F[2,I];
      F[1,I]:=T1;F[2,I]:=T2;
      K:=NV2;
      END;
    FLG:=1;
    WHILE FLG=1 DO
      BEGIN
      IF K<J THEN
        BEGIN
        J:=J-K;
        K:=K DIV 2;
        END
      ELSE
        BEGIN
        FLG:=0;
        END;
      END;
    J:=J+K;
    END;   (* END OF THE BIT REVERSAL PROCESS *)
  FOR L:=1 TO LN DO
    BEGIN
    LE:=POWER(2,L);
    LE1:=LE DIV 2;
    FOR J:=1 TO LE1 DO
      BEGIN
      I:=J;
```

```
    WHILE I<=N DO
      BEGIN
      IP:=I+LE1;
      T1:=F[1,IP];T2:=F[2,IP];
      F[1,IP]:=F[1,I]-T1;F[2,IP]:=F[2,I]-T2;
      F[1,I]:=F[1,I]+T1;F[2,I]:=F[2,I]+T2;
      I:=I+LE;
      END;
    END;
  END;
END;  (* PFWT *)


PROCEDURE TWOD(VAR FWT1:SPECTRUM;VAR COMFWT1:COMSPEC;
               DECIMG2:DECODED;NX4,NY4,FLAGY2,FLEG1:INTEGER);
  (* THIS PROCEDURE PRODUCES A TWO-D WALSH OR INVERSE WALSH    *)
  (* (BOTH ARE THE SAME PROCEDURE).  THE FOLLOWING             *)
  (* TYPE DECLARATIONS,                                        *)
  (*     TYPE                                                  *)
  (*       DECODED=ARRAY[1..SIZE,1..SIZE] OF INTEGER;          *)
  (*       SPECTRUM=ARRAY[1..SIZE,1..SIZE] OF REAL;            *)
  (*       COMSPEC=ARRAY[1..2,1..SIZE,1..SIZE] OF REAL;        *)
  (* MUST APPEAR UNDER THE MAIN DECLARATIONS.  A VARIABLE,     *)
  (* CALLED FF, OF TYPE COMPLEX, MUST ALSO APPEAR UNDER THE    *)
  (* MAIN DECLARATIONS.  WHENEVER THE PROCEDURE TWOD IS        *)
  (* CALLED THE FWT IS FOUND, EITHER OF THE CONTENTS OF FWT1   *)
  (* OR OF COMFWT1, DEPENDING UPON THE VALUE OF A FLAG, CALLED*)
  (* FLEG1 LOCALLY.  THE CALLING ROUTINE WILL GET THE REAL     *)
  (* PART OF COMFWT1 VIA FWT1 AND COMFWT1 HAS THE COMPLEX      *)
  (* WALSH SPECTRUM.  USE OF THIS PROCEDURE CAN BE DESCRIBED   *)
  (* AS FOLLOWS:                                               *)
  (*                                                           *)
  (*          FINDING WALSH TRANSFORMS OF COMPLEX INPUT        *)
  (*                                                           *)
  (* (A) FLEG1=1,FLAGY2=1 -                                    *)
  (*       1) TWOD WALSH ON THE CONTENTS OF COMFWT1[.,.,.,.]   *)
  (*          USING THE HADAMARD ORDER (HFWT PROCEDURE USED)   *)
  (*          WILL RESULT WHEN THESE FLAGS ARE USED.           *)
  (*       2) FWT1[.,.,.] WILL CONTAIN THE REAL PART OF        *)
  (*          COMFWT1[.,.,.,.] AFTER TWOD HAS RUN.             *)
  (*       3) COMFWT1[.,.,.,.] WILL HAVE THE TWO-D WALSH OUTPUT *)
  (*          (THE ORIGINAL DATA IN COMFWT1[.,.,.,.] WILL BE   *)
```

```
    (*            DISPLACED OF COURSE).                                *)
    (* (B) FLEG1=1,FLAGY2=0 -                                          *)
    (*            SAME AS CASE (A) EXCEPT WE USE PFWT (PALEY           *)
    (*            ORDER).                                              *)
    (*                                                                 *)
    (*            FINDING WALSH TRANSFORMS OF INTEGER INPUT            *)
    (*                                                                 *)
    (* (C) FLEG1=0,FLAGY2=1 -                                          *)
    (*         1) TWOD WALSH ON THE CONTENTS OF DECIMG2[.,.]           *)
    (*            USING THE HADAMARD ORDER (HFWT PROCEDURE USED)       *)
    (*            WILL RESULT WHEN THESE FLAGS ARE USED.               *)
    (*         2) FWT1[.,.] WILL CONTAIN THE REAL PART OF              *)
    (*            COMFWT1[.,.,.,.] AFTER TWOD HAS RUN.                 *)
    (*         3) COMFWT1[.,.,.,.] WILL HAVE THE TWO-D WALSH OUTPUT    *)
    (*            (REAL PART NONZERO, IMAGINARY PART ZERO).            *)
    (* (D) FLEG1=0,FLAGY2=0 -                                          *)
    (*            SAME AS CASE (C) EXCEPT WE USE PFWT (BINARY          *)
    (*            ORDER).                                              *)
VAR
  I,J,LNN:INTEGER;
  TEMP:COMSPEC;
BEGIN (* TWOD *)
LNN:=ROUND(LN(NY4*1.0)/LN(2.0));
IF FLEG1=1 THEN (* DO FAST WALSH OF COMFWT CONTENTS *)
  BEGIN
  IF FLAGY2=1 THEN (* USE HADAMARD ORDER (HFWT PROCEDURE) *)
    BEGIN
    FOR I:=1 TO NX4 DO
      BEGIN
      FOR J:=1 TO NY4 DO
        BEGIN
        FF[1,J]:=COMFWT1[1,I,J]; (* REAL PART INTO FF[1,J] *)
        FF[2,J]:=COMFWT1[2,I,J]; (* IMAG. PART INTO FF[2,J] *)
        END;
      HFWT(FF,LNN); (* ONE-D FWT OF ROW I *)
      FOR J:=1 TO NY4 DO
        BEGIN
        TEMP[1,I,J]:=FF[1,J];
        TEMP[2,I,J]:=FF[2,J];
        END;
      END; (* END OF ROW TRANSFORMATIONS *)
    FOR J:=1 TO NY4 DO
```

```
      BEGIN
      FOR I:=1 TO NX4 DO
        BEGIN
        FF[1,I]:=TEMP[1,I,J];
        FF[2,I]:=TEMP[2,I,J];
        END;
      HFWT(FF,LNN); (* ONE-D FWT OF COLUMN J *)
      FOR I:=1 TO NX4 DO
        BEGIN
        COMFWT1[1,I,J]:=FF[1,I];
        COMFWT1[2,I,J]:=FF[2,I];
        END;
      END; (* END OF FINDING FWT OF COLUMNS *)
    FOR I:=1 TO NX4 DO
      BEGIN
      FOR J:=1 TO NY4 DO
        BEGIN
        COMFWT1[1,I,J]:=COMFWT1[1,I,J]/NX4;
        COMFWT1[2,I,J]:=COMFWT1[2,I,J]/NX4;
        FWT1[I,J]:=COMFWT1[1,I,J];
        END;
      END;
    END
  ELSE  (* USE PALEY ORDER (PFWT PROCEDURE) *)
    BEGIN
    FOR I:=1 TO NX4 DO
      BEGIN
      FOR J:=1 TO NY4 DO
        BEGIN
        FF[1,J]:=COMFWT1[1,I,J]; (* REAL PART INTO FF[1,J] *)
        FF[2,J]:=COMFWT1[2,I,J]; (* IMAG. PART INTO FF[2,J] *)
        END;
      PFWT(FF,LNN); (* ONE-D FWT OF ROW I *)
      FOR J:=1 TO NY4 DO
        BEGIN
        TEMP[1,I,J]:=FF[1,J];
        TEMP[2,I,J]:=FF[2,J];
        END;
      END; (* END OF ROW TRANSFORMATIONS *)
    FOR J:=1 TO NY4 DO
      BEGIN
      FOR I:=1 TO NX4 DO
```

```
          BEGIN
          FF[1,I]:=TEMP[1,I,J];
          FF[2,I]:=TEMP[2,I,J];
          END;
        PFWT(FF,LNN); (* ONE-D FWT OF COLUMN J *)
        FOR I:=1 TO NX4 DO
          BEGIN
          COMFWT1[1,I,J]:=FF[1,I];
          COMFWT1[2,I,J]:=FF[2,I];
          END;
        END; (* END OF FINDING FWT OF COLUMNS *)
      FOR I:=1 TO NX4 DO
        BEGIN
        FOR J:=1 TO NY4 DO
          BEGIN
          COMFWT1[1,I,J]:=COMFWT1[1,I,J]/NX4;
          COMFWT1[2,I,J]:=COMFWT1[2,I,J]/NX4;
          FWT1[I,J]:=COMFWT1[1,I,J];
          END;
        END;
      END;
    END
  ELSE   (* DO FAST WALSH OF DECIMG CONTENTS *)
      BEGIN
      IF FLAGY2=1 THEN  (* USE HADAMARD ORDER (HFWT PROCEDURE) *)
        BEGIN
        FOR I:=1 TO NX4 DO
          BEGIN
          FOR J:=1 TO NY4 DO
            BEGIN
            FF[1,J]:=DECIMG2[I,J];
            FF[2,J]:=0.0;
            END;
          HFWT(FF,LNN); (* ONE-D FWT OF ROW I *)
          FOR J:=1 TO NY4 DO
            BEGIN
            TEMP[1,I,J]:=FF[1,J];
            TEMP[2,I,J]:=0.0;
            END;
          END;  (* END OF ROW TRANSFORMATIONS *)
        FOR J:=1 TO NY4 DO
          BEGIN
```

```
      FOR I:=1 TO NX4 DO
        BEGIN
        FF[1,I]:=TEMP[1,I,J];
        FF[2,I]:=TEMP[2,I,J];
        END;
      HFWT(FF,LNN);  (* ONE-D FWT OF COLUMN J *)
      FOR I:=1 TO NX4 DO
        BEGIN
        COMFWT1[1,I,J]:=FF[1,I];
        COMFWT1[2,I,J]:=FF[2,I];
        END;
      END; (* END OF FINDING FWT OF COLUMNS *)
    FOR I:=1 TO NX4 DO
      BEGIN
      FOR J:=1 TO NY4 DO
        BEGIN
        COMFWT1[1,I,J]:=COMFWT1[1,I,J]/NX4;
        COMFWT1[2,I,J]:=COMFWT1[2,I,J]/NX4;
        FWT1[I,J]:=COMFWT1[1,I,J];
        END;
      END;
    END
  ELSE  (* USE PALEY ORDER (PFWT PROCEDURE) *)
    BEGIN
    FOR I:=1 TO NX4 DO
      BEGIN
      FOR J:=1 TO NY4 DO
        BEGIN
        FF[1,J]:=DECIMG2[I,J];
        FF[2,J]:=0.0;
        END;
      PFWT(FF,LNN); (* ONE-D FWT OF ROW I *)
      FOR J:=1 TO NY4 DO
        BEGIN
        TEMP[1,I,J]:=FF[1,J];
        TEMP[2,I,J]:=0.0;
        END;
      END;  (* END OF ROW TRANSFORMATIONS *)
    FOR J:=1 TO NY4 DO
      BEGIN
      FOR I:=1 TO NX4 DO
        BEGIN
```

```
            FF[1,I]:=TEMP[1,I,J];
            FF[2,I]:=TEMP[2,I,J];
            END;
          PFWT(FF,LNN);  (* ONE-D FWT OF COLUMN J *)
          FOR I:=1 TO NX4 DO
            BEGIN
            COMFWT1[1,I,J]:=FF[1,I];
            COMFWT1[2,I,J]:=FF[2,I];
            END;
          END; (* END OF FINDING FWT OF COLUMNS *)
        FOR I:=1 TO NX4 DO
          BEGIN
          FOR J:=1 TO NY4 DO
            BEGIN
            COMFWT1[1,I,J]:=COMFWT1[1,I,J]/NX4;
            COMFWT1[2,I,J]:=COMFWT1[2,I,J]/NX4;
            FWT1[I,J]:=COMFWT1[1,I,J];
            END;
          END;
        END;
      END;
END;  (* TWOD *)


PROCEDURE EXPFGEN(VAR EXPF1:COMSPEC;INDEX,N:INTEGER);
  (* THIS PROCEDURE COMPUTES THE MATRIX OF COMPLEX    *)
  (* EXPONENTIALS FOR A PARTICULAR VALUE OF INDEX     *)
  (* (=0,1,2,...,N-1) & OF A PARTICULAR ORDER (N =    *)
  (* POWER-OF-2).                                     *)
VAR
  SCA:REAL;
  R,P:INTEGER;
BEGIN (* EXPFGEN *)
SCA:=1.0/N;
FOR R:=1 TO N DO
  BEGIN
  FOR P:=R TO N DO
    BEGIN
    EXPF1[1,R,P]:=SCA*COS(2.0*PI*INDEX*(R-P)/N);
    EXPF1[1,P,R]:=EXPF1[1,R,P];
    EXPF1[2,R,P]:=SCA*SIN(2.0*PI*INDEX*(R-P)/N);
    EXPF1[2,P,R]:=-EXPF1[2,R,P];
```

```
    END;
  END;
END;   (* EXPFGEN *)


PROCEDURE GWIGEN(VAR GWI1:COMSPEC; INDEX1,NN1,FLEGA,FLAGYA:INTEGER);
  (* THIS PROCEDURE USES TWOD AND EXPFGEN TO PRODUCE THE *)
  (* PARTIAL WALSH GAIN MATRIX GW(I) (= GLOBAL VARIABLE  *)
  (* GWI), FOR A PARTICULAR INDEX1, I.E., WE GET         *)
  (* GW(INDEX1), & A PARTICULAR ORDER NN1 (NN1=POWER-OF  *)
  (* -2).                                                *)
BEGIN (* GWIGEN *)
EXPFGEN(GWI1,INDEX1,NN1); (* GET EXPF-MATRIX OF COMPLEX
                             EXPONENTIALS *)
TWOD(FWT,GWI1,DECIMG,NN1,NN1,FLAGYA,FLEGA);
                          (* GET TWO-D WALSH OF COMPLEX
                             EXPONENTIALS *)
END;   (* GWIGEN *)


          (* MAINLINE PROGRAM *)


BEGIN (* GWGEN *)
RESET(IMAGEIN,'GWINPUT');
REWRITE(IMAGEOUT,'GWOUT');
READ(IMAGEIN,NN,FLEG,FLAGY);
FOR IL:=0 TO NN-1 DO
  BEGIN
  GWIGEN(GWI,IL,NN,FLEG,FLAGY);   (* GENERATE GW(I) *)
  WRITELN(IMAGEOUT);WRITELN(IMAGEOUT);WRITELN(IMAGEOUT);
  WRITELN(IMAGEOUT,'  PARTIAL WALSH GAIN MATRIX GW(',IL,')');
  WRITELN(IMAGEOUT,'      REAL PART');
  WRITELN(IMAGEOUT);
  FOR IX:=1 TO NN DO
    BEGIN
    FOR JX:=1 TO NN DO
      BEGIN
      WRITE(IMAGEOUT,GWI[1,IX,JX]:6:3);
      END;
    WRITELN(IMAGEOUT);
    END;
```

```
    WRITELN(IMAGEOUT);
    WRITELN(IMAGEOUT,'       IMAGINARY PART');
    WRITELN(IMAGEOUT);
    FOR IX:=1 TO NN DO
      BEGIN
      FOR JX:=1 TO NN DO
        BEGIN
        WRITE(IMAGEOUT,GWI[2,IX,JX]:6:3);
        END;
      WRITELN(IMAGEOUT);
      END;
    END;
END.   (* GWGEN *)
```

```
PROGRAM GHGEN(IMAGEIN,IMAGEOUT);
  (* THIS PROGRAM GENERATES ONE-DIMENSIONAL HAAR     *)
  (* GAIN MATRICES.  IT IS USED TO FIND THE GAIN     *)
  (* MATRIX IN THE HAAR  DOMAIN, EQUIVALENT TO THAT *)
  (* IN THE FOURIER DOMAIN.  THIS IS DONE IN THIS    *)
  (* PROGRAM BY AN IMPLEMENTATION OF THE MATRIX      *)
  (* EQUATION,                                       *)
  (*                    1     -1        T            *)
  (*             GH =  - H F   GF F H  C             *)
  (*                    N                            *)
  (* WHERE,  GH = HAAR  GAIN MATRIX (NON-DIAGONAL),  *)
  (*         GF = FOURIER GAIN MATRIX (DIAGONAL),    *)
  (*         H  = WALSH MATRIX CF THE TRANSFORM, &   *)
  (*         F  = FOURIER MATRIX OF THE TRANSFORM    *)
  (*         C  = DIAGONAL CORRECTION MATRIX         *)
  (*                           T -1                  *)
  (*              = ((1/N) H H )                     *)
  (* GH MAY BE COMPLEX IF GF IS PHYSICALLY NOT       *)
  (* REALIZABLE.  NOTE,                              *)
  (*    GF = DIAG(G(0),G(1),...,G(I),...,G(N-1))     *)
  (* SO G(I) IS THE ITH DIAGONAL ELEMENT OF GF.      *)
  (* THE TOTAL HAAR  GAIN MATRIX, GH, IS             *)
  (*                  N-1                            *)
```

```
  (*            GH = SUM GH(I)                                    *)
  (*                 I=0                                          *)
  (* THE ORTHOGONAL HADAMARD ORDERED HAAR TRANSFORM               *)
  (* , ACCORDING TO                                               *)
  (*     AHMED,NATARAJAN,RAO,"COOLEY-TUKEY-TYPE                   *)
  (*     ALGORITHM FOR THE HAAR TRANSFORM,"                       *)
  (*     ELECTRONICS LETTERS, JUNE 1973, PP.276-278               *)
  (* , IS USED HERE (SEE HFHAART PROCEDURE).                      *)

CONST
  PI=3.141592654;
  SIZE=64;
TYPE
  COMPLEX=ARRAY[1..2,1..1024] OF REAL;
  COMSPEC=ARRAY[1..2,1..SIZE,1..SIZE] OF REAL;
VAR
  IX,JX,NN,IL:INTEGER;
  IMAGEIN,IMAGEOUT:TEXT;
  FF:COMPLEX;
  GHI:COMSPEC;


PROCEDURE HFHAART(VAR F:COMPLEX;LN:INTEGER);
  (* THIS PROCEDURE CALCULATES THE ORTHOGONAL (NOT ORTHONORMAL)   *)
  (* HADAMARD ORDERED DISCRETE HAAR TRANSFORM ACCORDING TO        *)
  (*     "COOLEY-TUKEY-TYPE ALGORITHM FOR THE HAAR TRANSFORM"     *)
  (*      BY N. AHMED,T. NATARAJAN, AND K. R. RAO,                *)
  (*      ELECTRONICS LETTERS, JUNE 1973, PP. 276-278.            *)
  (* THE ORDERING PRODUCED BY THIS PROCEDURE IS RECURSIVELY       *)
  (* GENERATED USING H  = 1 AND                                   *)
  (*                  1                                           *)
  (*                                                              *)
  (*                   |  H     H  |                              *)
  (*        H     =    |   N     N |                              *)
  (*         2N        |           |                              *)
  (*                   |  I    -I  |                              *)
  (*                   |   N     N |                              *)
  (*                                                              *)
  (* WHERE I   IS THE NTH ORDER UNIT MATRIX,                      *)
  (*        N                                                     *)
  (*       H  IS THE NTH ORDER ORTHOGONAL HADAMARD-HAAR MATRIX,   *)
  (*        N                                                     *)
```

```
    (*         H  IS THE 2NTH ORDER ORTHOGONAL HADAMARD-HAAR MATRIX.*)
    (*          2N                                                   *)
  VAR
    I,J,L,N,LE,LE1:INTEGER;
    T1,T2:REAL;
  BEGIN (* HFHAART *)
  N := POWER(2,LN);
  FOR L:=1 TO LN DO
    BEGIN
    LE := POWER(2,LN-L+1);
    LE1 := LE DIV 2;
    FOR J:=1 TO LE1 DO
      BEGIN
      I := J + LE1;
      T1 := F[1,J]; T2 := F[2,J];
      F[1,J] := F[1,J] + F[1,I];
      F[2,J] := F[2,J] + F[2,I];
      F[1,I] := T1 - F[1,I];
      F[2,I] := T2 - F[2,I];
      END;
    END;
  END;   (* HFHAART *)


  PROCEDURE IHFHAART(VAR F:COMPLEX;LN:INTEGER);
    (* PROCEDURE HFHAART DETERMINES H TIMES F WITH H THE ORTHOGONAL *)
    (* HADAMARD ORDERED HAAR AND F THE INPUT VECTOR. THIS PROCEDURE *)
    (*                    T                                         *)
    (* COMPUTES (1/N) H  F, THE INVERSE HAAR TRANSFORM OF THE F     *)
    (* INPUT VECTOR. THIS IS NOT THE TRUE INVERSE, HOWEVER.         *)
  VAR
    I,J,N,L,LE,LE1:INTEGER;
    T1,T2:REAL;
  BEGIN (* IHFHAART *)
  N := POWER(2,LN);
  FOR L:=1 TO LN DO
    BEGIN
    LE := POWER(2,L);
    LE1:= LE DIV 2;
    FOR J:=1 TO LE1 DO
      BEGIN
      I := J + LE1;
```

```
      T1 := F[1,J];  T2 := F[2,J];
      F[1,J] := F[1,J] + F[1,I];
      F[2,J] := F[2,J] + F[2,I];
      F[1,I] := T1 - F[1,I];
      F[2,I] := T2 - F[2,I];
      END;
    END;
  FOR I:=1 TO N DO
    BEGIN
    F[1,I] := F[1,I]/N;
    F[2,I] := F[2,I]/N;
    END;
  END;  (* IHFHAART *)


  PROCEDURE TWOD(VAR COMFHT1:COMSPEC;NX4,NY4:INTEGER);
    (* THIS PROCEDURE PRODUCES A PSEUDO 2D HAAR TRANSFORM (AS    *)
    (* REQUIRED BY FORMULA FOR GH).                              *)
    (* THE A-MATRIX IS                                           *)
    (*                          -1                               *)
    (*                  A = F  GF F                              *)
    (* AND WE DO A PSEUDO 2D HAAR OF A TO GET GH, ACCORDING TO   *)
    (*                            1        T                     *)
    (*                GH  =       - H  A  H  C                  *)
    (*                            N                              *)
  VAR
    I,J,K,OFFSET,LNN:INTEGER;
    TEMP:COMSPEC;
    CMAT:ARRAY[1..SIZE] OF REAL;
  BEGIN (* TWOD *)
  LNN:=ROUND(LN(NY4*1.0)/LN(2.0));

      (* TAKE THE HAAR TRANSFORM, COLUMN BY COLUMN,  *)
         (* OF COMFHT1[.,.,.,.] (A-MATRIX)           *)

  FOR J:=1 TO NY4 DO
    BEGIN
    FOR I:=1 TO NX4 DO
      BEGIN
      FF[1,I] := COMFHT1[1,I,J];
      FF[2,I] := COMFHT1[2,I,J];
      END;
```

```
    HFHAART(FF,LNN);
    FOR I:=1 TO NX4 DO
      BEGIN
      TEMP[1,J,I] := FF[1,I];
      TEMP[2,J,I] := FF[2,I];
      END;
    END;

      (* TAKE THE HAAR TRANSFORM, COLUMN BY COLUMN, *)
         (* OF TEMP[.,.,.] (TRANSPOSE OF PRODUCT H A). *)
         (* "SIMULTANEOUSLY" TAKE THE TRANSPOSE OF THE *)
         (* RESULT AND DIVIDE BY N.                    *)

  FOR J:=1 TO NY4 DO
    BEGIN
    FOR I:=1 TO NX4 DO
      BEGIN
      FF[1,I] := TEMP[1,I,J];
      FF[2,I] := TEMP[2,I,J];
      END;
    HFHAART(FF,LNN);
    FOR I:=1 TO NX4 DO
      BEGIN
      COMFHT1[1,J,I] := FF[1,I]/NX4;
      COMFHT1[2,J,I] := FF[2,I]/NX4;
      END;
    END;

      (* NOW FACTOR IN THE C-MATRIX CORRECTION FACTOR. *)
      (* CMAT[.] IS THE INVERSE C-MATRIX.              *)

OFFSET := 3;
CMAT[1] := 1.0;
CMAT[2] := 1.0;
FOR J:=1 TO (LNN-1) DO
  BEGIN
  K := POWER(2,J);
  FOR I:=OFFSET TO (OFFSET+K-1) DO
    BEGIN
    CMAT[I] := POWER(2,LNN-J)/NX4;
    END;
  OFFSET := OFFSET + K;
```

```
    END;

  FOR I:=1 TO NX4 DO
    BEGIN
    FOR J:=1 TO NY4 DO
      BEGIN
      COMFHT1[1,I,J] := COMFHT1[1,I,J]/CMAT[J];
      COMFHT1[2,I,J] := COMFHT1[2,I,J]/CMAT[J];
      END;
    END;
  END;  (* TWOD *)


  PROCEDURE EXPFGEN(VAR EXPF1:COMSPEC;INDEX,N:INTEGER);
    (* THIS PROCEDURE COMPUTES THE MATRIX OF COMPLEX    *)
    (* EXPONENTIALS FOR A PARTICULAR VALUE OF INDEX     *)
    (* (=0,1,2,...,N-1) & OF A PARTICULAR ORDER (N =    *)
    (* POWER-OF-2).                                     *)
  VAR
    SCA:REAL;
    R,P:INTEGER;
  BEGIN (* EXPFGEN *)
  SCA:=1.0/N;
  FOR R:=1 TO N DO
    BEGIN
    FOR P:=R TO N DO
      BEGIN
      EXPF1[1,R,P]:=SCA*COS(2.0*PI*INDEX*(R-P)/N);
      EXPF1[1,P,R]:=EXPF1[1,R,P];
      EXPF1[2,R,P]:=SCA*SIN(2.0*PI*INDEX*(R-P)/N);
      EXPF1[2,P,R]:=-EXPF1[2,R,P];
      END;
    END;
  END;  (* EXPFGEN *)


PROCEDURE GHIGEN(VAR GHI1:COMSPEC;INDEX1,NN1:INTEGER);
  (* THIS PROCEDURE USES TWOD AND EXPFGEN TO PRODUCE THE *)
  (* PARTIAL HAAR  GAIN MATRIX GH(I) (= GLOBAL VARIABLE  *)
  (* GHI), FOR A PARTICULAR INDEX1, I.E., WE GET         *)
  (* GH(INDEX1), & A PARTICULAR ORDER NN1 (NN1=POWER-OF  *)
  (* -2).                                                *)
```

```
BEGIN (* GHIGEN *)
EXPFGEN(GHI1,INDEX1,NN1); (* GET EXPF-MATRIX OF COMPLEX
                             EXPONENTIALS *)
TWOD(GHI1,NN1,NN1);
                          (* GET PSEUDO TWO-D HAAR  OF COMPLEX
                             EXPONENTIALS *)
END;  (* GHIGEN *)


        (* MAINLINE PROGRAM *)


BEGIN (* GHGEN *)
RESET(IMAGEIN,'GHINPUT');
REWRITE(IMAGEOUT,'GHOUT');
READ(IMAGEIN,NN);
FOR IL:=0 TO NN-1 DO
  BEGIN
  GHIGEN(GHI,IL,NN);  (* GENERATE GH(I) *)
  WRITELN(IMAGEOUT);WRITELN(IMAGEOUT);WRITELN(IMAGEOUT);
  WRITELN(IMAGEOUT,'  PARTIAL HAAR  GAIN MATRIX GH(',IL,')');
  WRITELN(IMAGEOUT,'      REAL PART');
  WRITELN(IMAGEOUT);
  FOR IX:=1 TO NN DO
    BEGIN
    FOR JX:=1 TO NN DO
      BEGIN
      WRITE(IMAGEOUT,GHI[1,IX,JX]:6:3);
      END;
    WRITELN(IMAGEOUT);
    END;
  WRITELN(IMAGEOUT);
  WRITELN(IMAGEOUT,'      IMAGINARY PART');
  WRITELN(IMAGEOUT);
  FOR IX:=1 TO NN DO
    BEGIN
    FOR JX:=1 TO NN DO
      BEGIN
      WRITE(IMAGEOUT,GHI[2,IX,JX]:6:3);
      END;
    WRITELN(IMAGEOUT);
    END;
```

```
  END;
END.  (* GHGEN *)
```

```
PROGRAM TRGEN(IMAGEIN,IMAGEOUT);
  (* THIS PROGRAM GENERATES ONE-DIMENSIONAL TRIDIAGONAL      *)
  (* GAIN MATRICES.  IT IS USED TO FIND THE GAIN             *)
  (* MATRIX IN THE TRIDIADONAL DOMAIN, EQUIVALENT TO THAT    *)
  (* IN THE FOURIER DOMAIN.  THIS IS DONE IN THIS            *)
  (* PROGRAM BY AN IMPLEMENTATION OF THE MATRIX              *)
  (* EQUATION,                                               *)
  (*                    1     -1                             *)
  (*             GT =   - T F    GF F T                     *)
  (*                    2                                    *)
  (* WHERE,  GT = TRIDIAGONAL MATRIX (NON-DIAGONAL),         *)
  (*         GF = FOURIER GAIN MATRIX (DIAGONAL),            *)
  (*         T  = MATRIX OF THE TRANSFORM, &                 *)
  (*         F  = FOURIER MATRIX OF THE TRANSFORM            *)
  (* GT MAY BE COMPLEX IF GF IS PHYSICALLY NOT               *)
  (* REALIZABLE.  NOTE,                                      *)
  (*    GF = DIAG(G(0),G(1),...,G(I),...,G(N-1))             *)
  (* SO G(I) IS THE ITH DIAGONAL ELEMENT OF GF.              *)
  (* THE TOTAL TRIDIAGONAL GAIN MATRIX, GT, IS               *)
  (*                  N-1                                    *)
  (*            GT = SUM GT(I)                               *)
  (*                  I=0                                    *)

CONST
  PI=3.141592654;
  SIZE=64;
TYPE
  COMPLEX=ARRAY[1..2,1..1024] OF REAL;
  COMSPEC=ARRAY[1..2,1..SIZE,1..SIZE] OF REAL;
VAR
  IX,JX,NN,IL:INTEGER;
  IMAGEIN,IMAGEOUT:TEXT;
  FF:COMPLEX;
  GTI:COMSPEC;
```

```
PROCEDURE TRIDIAG(VAR F:COMPLEX;LN:INTEGER);
  (* THIS PROCEDURE COMPUTES THE TRIDIAGONAL TRANSFORM OF AN     *)
  (* INPUT VECTOR.  THE TRIDIAGONAL TRANSFORM IS                 *)
  (* GENERATED USING T   = 1 AND                                 *)
  (*                  1                                          *)
  (*                                                             *)
  (*                    |   I       I   |                        *)
  (*          T     =   |   N       N   |                        *)
  (*           2N       |               |                        *)
  (*                    |   I      -I   |                        *)
  (*                    |   N       N   |                        *)
  (*                                                             *)
  (* WHERE I   IS THE NTH ORDER UNIT MATRIX,                     *)
  (*        N                                                    *)
  (*       T  IS THE NTH ORDER TRIDIAGONAL TRANSFORM MATRIX,     *)
  (*        N                                                    *)
  (*       T  IS THE 2NTH ORDER TRIDIAGONAL  TRANSFORM  MATRIX.  *)
  (*        2N                                                   *)
VAR
  I,J,N,LE,LE1:INTEGER;
  T1,T2:REAL;
BEGIN (* TRIDIAG *)
N := POWER(2,LN);
LE := POWER(2,LN);
LE1 := LE DIV 2;
FOR J:=1 TO LE1 DO
  BEGIN
  I := J + LE1;
  T1 := F[1,J]; T2 := F[2,J];
  F[1,J] := F[1,J] + F[1,I];
  F[2,J] := F[2,J] + F[2,I];
  F[1,I] := T1 - F[1,I];
  F[2,I] := T2 - F[2,I];
  END;
END;  (* TRIDIAG *)


PROCEDURE TWOD(VAR COMFHT1:COMSPEC;NX4,NY4:INTEGER);
  (* THIS PROCEDURE PRODUCES A PSEUDO 2D TRIDIAGONAL TRANSFORM *)
  (* REQUIRED BY FORMULA FOR GT).                               *)
  (* THE A-MATRIX IS                                            *)
  (*                          -1                                *)
```

```
    (*                          A = F   GF F                                *)
    (* AND WE DO A PSEUDO 2D TRIDIAGONAL TRANSFORM OF A TO GET,             *)
    (*                                 1                                    *)
    (*                      GT    =    -  T  A  T                          *)
    (*                                 2                                    *)
  VAR
    I,J,K,OFFSET,LNN:INTEGER;
    TEMP:COMSPEC;
  BEGIN (* TWOD *)
  LNN:=ROUND(LN(NY4*1.0)/LN(2.0));

      (* TAKE THE TRID. TRANSFORM, COLUMN BY COLUMN,  *)
         (* OF COMFHT1[.,.,.,.] (A-MATRIX)                          *)

  FOR J:=1 TO NY4 DO
    BEGIN
    FOR I:=1 TO NX4 DO
      BEGIN
      FF[1,I] := COMFHT1[1,I,J];
      FF[2,I] := COMFHT1[2,I,J];
      END;
    TRIDIAG(FF,LNN);
    FOR I:=1 TO NX4 DO
      BEGIN
      TEMP[1,J,I] := FF[1,I];
      TEMP[2,J,I] := FF[2,I];
      END;
    END;

      (* TAKE THE TRID. TRANSFORM, COLUMN BY COLUMN,  *)
         (* OF TEMP[.,.,.,.] (TRANSPOSE OF PRODUCT T A).  *)
         (* "SIMULTANEOUSLY" TAKE THE TRANSPOSE OF THE  *)
         (* RESULT AND DIVIDE BY 2.                                 *)

FOR J:=1 TO NY4 DO
  BEGIN
  FOR I:=1 TO NX4 DO
    BEGIN
    FF[1,I] := TEMP[1,I,J];
    FF[2,I] := TEMP[2,I,J];
    END;
  TRIDIAG(FF,LNN);
```

```
    FOR I:=1 TO NX4 DO
      BEGIN
      COMFHT1[1,J,I] := FF[1,I]/2;
      COMFHT1[2,J,I] := FF[2,I]/2;
      END;
    END;
  END;  (* TWOD *)


  PROCEDURE EXPFGEN(VAR EXPF1:COMSPEC;INDEX,N:INTEGER);
    (* THIS PROCEDURE COMPUTES THE MATRIX OF COMPLEX    *)
    (* EXPONENTIALS FOR A PARTICULAR VALUE OF INDEX     *)
    (* (=0,1,2,...,N-1) & OF A PARTICULAR ORDER (N =    *)
    (* POWER-OF-2).                                     *)
  VAR
    SCA:REAL;
    R,P:INTEGER;
  BEGIN (* EXPFGEN *)
  SCA:=1.0/N;
  FOR R:=1 TO N DO
    BEGIN
    FOR P:=R TO N DO
      BEGIN
      EXPF1[1,R,P]:=SCA*COS(2.0*PI*INDEX*(R-P)/N);
      EXPF1[1,P,R]:=EXPF1[1,R,P];
      EXPF1[2,R,P]:=SCA*SIN(2.0*PI*INDEX*(R-P)/N);
      EXPF1[2,P,R]:=-EXPF1[2,R,P];
      END;
    END;
  END;  (* EXPFGEN *)


  PROCEDURE TRIGEN(VAR GTI1:COMSPEC;INDEX1,NN1:INTEGER);
    (* THIS PROCEDURE USES TWOD AND EXPFGEN TO PRODUCE THE *)
    (* PARTIAL TRID. GAIN MATRIX GT(I) (= GLOBAL VARIABLE  *)
    (* GTI), FOR A PARTICULAR INDEX1, I.E., WE GET         *)
    (* GT(INDEX1), & A PARTICULAR ORDER NN1 (NN1=POWER-OF  *)
    (* -2).                                                *)
  BEGIN (* TRIGEN *)
  EXPFGEN(GTI1,INDEX1,NN1); (* GET EXPF-MATRIX OF COMPLEX
                               EXPONENTIALS *)
  TWOD(GTI1,NN1,NN1);
```

```
                                (* GET PSEUDO TWO-D TRID. OF COMPLEX
                                   EXPONENTIALS *)
  END;   (* TRIGEN *)


            (* MAINLINE PROGRAM *)


  BEGIN (* TRGEN *)
  RESET(IMAGEIN,'TRINPUT');
  REWRITE(IMAGEOUT,'TROUT');
  READ(IMAGEIN,NN);
  FOR IL:=0 TO NN-1 DO
    BEGIN
    TRIGEN(GTI,IL,NN);   (* GENERATE GT(I) *)
    WRITELN(IMAGEOUT);WRITELN(IMAGEOUT);WRITELN(IMAGEOUT);
    WRITELN(IMAGEOUT,'  PARTIAL TRIDIAGONAL GAIN MATRIX GT(',IL,')');
    WRITELN(IMAGEOUT,'      REAL PART');
    WRITELN(IMAGEOUT);
    FOR IX:=1 TO NN DO
      BEGIN
      FOR JX:=1 TO NN DO
        BEGIN
        WRITE(IMAGEOUT,GTI[1,IX,JX]:6:3);
        END;
      WRITELN(IMAGEOUT);
      END;
    WRITELN(IMAGEOUT);
    WRITELN(IMAGEOUT,'      IMAGINARY PART');
    WRITELN(IMAGEOUT);
    FOR IX:=1 TO NN DO
      BEGIN
      FOR JX:=1 TO NN DO
        BEGIN
        WRITE(IMAGEOUT,GTI[2,IX,JX]:6:3);
        END;
      WRITELN(IMAGEOUT);
      END;
  END;
END.   (* TRGEN *)
```

Appendix H

DEFINITIONS OF BOUNDS

From Ullman [40], f(N) is O(g(N)) (or f(N) = O(g(N))) if there are positive constants c and N_0 such that for all $N \geq N_0$, $f(N) \leq cg(N)$. Also from Ullman [40], f(N) is $\Omega(g(N))$ (or $f(N) = \Omega(g(N))$) if there are positive constants c and N_0 such that for all $N \geq N_0$, $f(N) \geq cg(N)$. From Thompson [43], f(N) is $\theta(g(N))$ (or $f(N) = \theta(g(N))$) if there exist positive constants c_1 and c_2 for which $c_1 g(N) \leq f(N) \leq c_2 g(N)$ for all sufficiently large N.

REFERENCES

[1] L. R. Rabiner,"The Acoustics,Speech and Signal Processing Society - A Historical Perspective," IEEE ASSP Magazine ,vol. 1, pp.4-10,Jan. 1984.

[2] A. V. Oppenheim and R. W. Schafer, Digital Signal Processing . Englewood Cliffs, New Jersey:Prentice-Hall,1975.

[3] C. T. Chen, One-Dimensional Digital Signal Processing . New York, New York:Marcel-Dekker, 1979.

[4] A. Fettweis,"Digital Filter Structures Related to Classical Filter Networks," Arch. Elek. Ubertragung , vol. 25, pp. 78-89, 1971.

[5] A. H. Gray and J. D. Markel,"Digital Lattice and Ladder Filter Synthesis," IEEE Trans. Aud., Electroacoust., vol. AU-21, pp. 491-500, Dec. 1973.

[6] B. Widrow, J. M. McCool, M. G. Larimore, C. R. Johnson," Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter," Proc. IEEE , vol. 64, pp. 1151-1161, Aug. 1976.

[7] R. C. Gonzalez and P. Wintz, Digital Image Processing . Reading, Massachusetts:Addison-Wesley, 1977.

[8] N. A. Ahmed, H. H. Schreiber, and F. V. Lopresti,"On Notation and Definition of Terms Related to a Class of Complete Orthogonal Functions," IEEE Trans. on Electromagn. Compat., vol. EMC-15, pp. 75-80, May 1973.

[9] J. W. Cooley and J. W. Tukey,"An Algorithm For the Machine Calculation of Complex Fourier Series," Math. Comput., vol. 19, pp. 297-301, 1965.

[10] D. K. Cheng and J. J. Liu,"Time-Shift Theorems for Walsh Transforms and Solution of Difference Equations," IEEE Trans. Electromagn. Compat., vol. EMC-18, pp. 83-87, May 1976.

[11] M. S. Corrington,"Solution of Differential and Integral Equations with Walsh Functions," IEEE Trans. Circ. Theo., vol. CT-20, pp. 470-476, Sept. 1973.

[12] M. Maqusi,"Walsh Series Expansions of Probability Distributions," IEEE Trans. Electromagn. Compat., vol. EMC-23, pp. 197-203, Nov. 1981.

[13] J. Pearl,"Optimal Dyadic Models of Time-Invariant Systems," IEEE Trans. on Comp., vol. C-24, pp. 598-603, June 1975.

[14] Y. Tadokoro and T. Higuchi,"Discrete Fourier Transform Computation via the Walsh Transform," IEEE Trans. Acoust., Speech and Signal Proc., vol. ASSP-26, pp. 236-240, June 1978.

[15] Y. Tadokoro and T. Higuchi,"Conversion Factors From Walsh Coefficients to Fourier Coefficients," IEEE Trans. Acoust., Speech and Signal Proc., vol. ASSP-31, pp. 231-232, Feb. 1983.

[16] C.-F. Chan and C.-H. Hsiao,"Design of Piecewise Constant Gains for Optimal Control via Walsh Functions," IEEE Trans. Automat. Control, vol. AC-20, pp. 596-602, Oct. 1975.

[17] R. Dzwonczyk, M. B. Howie, and J. S. McDonald,"A Comparison Between Walsh and Fourier Analysis of the Electroencephalogram for Tracking the Effects of Anesthesia," IEEE Trans. Biomed. Eng., vol. BME-31, pp. 551-556, Aug. 1984.

[18] G. P. Tolstov, Fourier Series . New York,New York:Dover, 1962.

[19] K. R. Rao, V. Devarajan, V. Vlasenko, and M. A. Narasimhan,"Cal-Sal Walsh-Hadamard Transform," IEEE Trans. Acoust., Speech and Signal Proc., vol. ASSP-26, pp. 605-607, Dec. 1978.

[20] J. L. Shanks,"Computation of the Fast Walsh-Fourier Transform," IEEE Trans. on Comp., vol. C-18, pp. 457-459, May 1969.

[21] J. E. Shore,"On the Applications of Haar Functions," IEEE Trans. on Comm., vol. COM-21, pp. 209-216, March 1973.

[22] V. J. Rejchrt,"Signal Flow Graph and a Fortran Program for Haar- Fourier Transform," IEEE Trans. on Comp., vol. C-21 , pp. 1026-1027, Sept. 1972.

[23] N. Ahmed, T. Natarajan and K. R. Rao,"Cooley-Tukey-Type Algorithm for the Haar Transform," Electronics Letters, vol. 9, pp. 276-278, June 1973.

[24] C. J. Zarowski, M. Yunik and G. O. Martens,"Fourier Spectrum Filtering Methods Useful for VLSI Implementation," submitted to IEEE Trans. on Comp. for review.

[25] P. Yip and K. R. Rao,"A Fast Computational Algorithm for the Discrete Sine Transform," IEEE Trans. on Comm., vol. COM-28, pp. 304-307, Feb. 1980.

[26] N. Ahmed, T. Natarajan and K. R. Rao,"Discrete Cosine Transform," IEEE Trans. on Comp., vol. C-23, pp. 90-93, Jan. 1974.

[27] C. J. Zarowski and M. Yunik,"Spectral Filtering Using the Fast Walsh Transform," accepted for publication in IEEE Trans. Acoust., Speech and Signal Proc.

[28] A. E. Kahveci and E. L. Hall,"Sequency Domain Design of Frequency Filters," IEEE Trans. on Comp., vol. C-23, pp. 976-981, Sept. 1974.

[29] K. H. Siemens and R. Kitai,"Digital Walsh-Fourier Analysis of Periodic Waveforms," IEEE Trans. Instrum. Meas., vol. IM-18, pp. 316-320, Dec. 1969.

[30] N. M. Blachman,"Spectral Analysis With Sinusoids and Walsh Functions," IEEE Trans. Aerosp. Electron. Syst., vol. AES-7, pp. 900-905, Sept. 1971.

[31] S. Winograd,"On Computing the Discrete Fourier Transform," Math. of Comput., vol. 32, pp. 175-199, Jan. 1978.

[32] H. W. Jones, D. N. Hein, and S. C. Knauer,"Karhunen-Loeve Discrete Cosine and Related Transforms Obtained via the Hadamard Transform," Proc. Int. Telemetering Conf. , Los Angeles CA, pp. 87-98, Nov. 1978.

[33] S. Venkatraman, V. R. Kanchan, K. R. Rao and R. Srinivasan,"Discrete Transforms via the Walsh-Hadamard Transform," under review by IEEE Trans. on Acoust., Speech and Signal Proc.

[34] H. S. Kwak, R. Srinivasan and K. R. Rao,"C-Matrix Transform," IEEE Trans. Acoust., Speech and Signal Proc., vol. ASSP-31, pp. 1304-1307,Oct. 1983.

[35] R. B. J. T. Allenby, Rings, Fields and Groups: An Introduction to Abstract Algebra . East Kilbride, Scotland:Edward Arnold, 1983.

[36] I. J. Good,"The Interaction Algorithm and Practical Fourier Analysis," J. Roy. Stat. Soc. (London), vol. B20, pp. 361-371, 1958.

[37] H. C. Andrews and K. L. Caspari,"A Generalized Technique for Spectral Analysis," IEEE Trans. on Comp., vol. C-19, pp. 16-25, Jan. 1970.

[38] L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall, 1974.

[39] C. Mead and L. Conway, Introduction to VLSI Systems. Reading, Massachusetts:Addison-Wesley, 1980.

[40] J. D. Ullman, Computational Aspects of VLSI. Rockville, Maryland:Computer Science Press, 1984.

[41] H. S. Stone,"Parallel Processing With the Perfect Shuffle," IEEE Trans. on Comp., vol. C-20, pp. 153-161, Feb. 1971.

[42] F. P. Preparata and J. Vuillemin,"The Cube-Connected Cycles: A Versatile Network for Parallel Computation," Comm. of ACM , vol. 24, pp. 300-309, May 1981.

[43] C. D. Thompson,"A Complexity Theory for VLSI," Ph. D. thesis, Carnegie-Mellon University, United States of America, August 1980.

[44] C. D. Thompson,"Fourier Transforms in VLSI," IEEE Trans. on Comp., vol. C-32, pp. 1047-1057, Nov. 1983.

[45] H. L. Groginsky and G. A. Works,"A Pipeline Fast Fourier Transform," IEEE Trans. on Comp., vol. C-19, pp. 1015-1019, Nov. 1970.

[46] A. Peled and B. Liu,"A New Hardware Realization of Digital Filters," IEEE Trans. Acoust., Speech and Signal Proc., vol. ASSP-22, pp. 456-461, Dec. 1974.

[47] B. Liu and A. Peled,"A New Hardware Realization of High-Speed Fast Fourier Transforms," IEEE Trans. Acoust., Speech and Signal Proc., vol. ASSP-23, pp. 543-547, Dec. 1975.

[48] A. Despain,"Fourier Transform Computers Using CORDIC Iterations," IEEE Trans. on Comp., vol. C-23, pp. 993-1001, Oct. 1974.

[49] J. E. Volder,"The CORDIC Trigonometric Computing Technique," IRE Trans. on Electron. Comp., vol. EC-8, pp. 330-334, Sept. 1959.

[50] V. C. Hamacher, Z. G. Vranesic, and S. G. Zaky, Computer Organization. New York, New York:McGraw-Hill, 1978.

[51] H. T. Kung and C. E. Leiserson,"Systolic Arrays (for VLSI)," Proc. Symp. Sparse Matrix Computations and Their Applications, pp. 256-282, Nov. 2-3 1978.

[52] G. E. Bridges, W. Pries, R. D. McCleod, M. Yunik, P. G. Gulak, and H. C. Card,"Dual Systolic Architectures for VLSI Digital Signal Processing Systems," submitted to IEEE Trans. on Comp. for review.

[53] S. Winograd,"Some Bilinear Forms Whose Multiplicative Complexity Depends on the Field of Constants," Mathematical Systems Theory, vol. 10, pp. 169-180, 1977.

[54] J. H. McClellan and C. M. Rader, Number Theory in Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall, 1979.

[55] G. G. Apple and P. A. Wintz,"Calculation of Fourier Transforms on Finite Abelian Groups," IEEE Trans. on Info. Theory, vol. IT-16, pp. 233-234, March 1970.

[56] T. W. Cairns,"On the Fast Fourier Transform on Finite Abelian Groups," IEEE Trans. on Comp., vol. C-20, pp. 569-571, May 1971.

[57] J. H. Rosenbloom,"Physical Interpretation of Dyadic Groups," Applications of Walsh Functions Proc., pp. 158-165, 1971.

[58] M. G. Karpovsky,"Fast Fourier Transforms on Finite Non-Abelian Groups," IEEE Trans. on Comp., vol. C-26, pp. 1028-1030, Oct. 1977.

[59] M. G. Karpovsky and Trachtenberg,"Some Optimization Problems for Convolution Systems over Finite Groups," Info. and Control , vol. 34, pp. 227-247, 1977.