INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the original text directly from the copy submitted. Thus, some dissertation copies are in typewriter face, while others may be from a computer printer.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyrighted material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is available as one exposure on a standard 35 mm slide or as a 17″ × 23″ black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. 35 mm slides or 6″ × 9″ black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I
Accessing the World's Information since 1938
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

Order Number 8820894

On tensor products formulations of additive fast Fourier transform algorithms and their implementations

Rodriguez, Domingo Antonio, Ph.D.

City University of New York, 1988

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

ON TENSOR PRODUCTS FORMULATIONS OF ADDITIVE FAST FOURIER TRANSFORM ALGORITHMS AND THEIR IMPLEMENTATIONS

by

DOMINGO RODRIGUEZ

A dissertation submitted to the Graduate Faculty in Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1988

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Jan 22, 1988
date

Richard Johnson
Chairman of Examining Committee

Jan. 22, 1988
date

Jacques E. Benveniste
Executive Officer

Louis Auslander

Michael Conner

George Eichmann

Allen Gorin

Robert Johnson
Supervisory Committee

Abstract

ON TENSORS PRODUCTS FORMULATIONS OF ADDITIVE FAST FOURIER TRANSFORM ALGORITHMS AND THEIR IMPLEMENTATIONS

by

DOMINGO RODRIGUEZ

Advisor: Professor Richard Tolimieri

One of the objectives of this work is to present a mathematical language or structure in which to analyze in a unified format similarities and differences among the commonly known fast Fourier transform (FFT) algorithms. This language is the language of tensor products, a branch of finite dimensional multilinear algebra. We concentrate on algorithms which take advantage of the additive structure of the indexing sets of input and output data during an algorithmic computation. One of the advantages of using tensor products language to describe FFT algorithms is that this mathematical language may be used as an analytic tool for the study of algorithmic structures for machine hardware and software implementations as well as the identification of new algorithms. For instance, an inherent part in the study of computer implementation of FFT algorithms is the analysis of the data communications aspects of the algorithms which manifest themselves during implementation procedures. These data communication aspects can be best studied, in turn, through the analysis of permutation matrices which appear in our tensor products formulations of the FFT algorithms.

Another objective of this work is to present a mathematical characterization of linear shift invariant, finite impulse response (LSI-FIR) filters, and describe how to use the tensor products language as tool to aid in the implementation of these filters using FFT algorithms.

TABLE OF CONTENTS

Introduction:

One of the objectives of this work is to present a mathematical language or structure in which to analyze in a unified format similarities and differences among the commonly known fast Fourier transform (FFT) algorithms. This language is the language of tensor products, a branch of finite dimensional multilinear algebra. We will concentrate on algorithms which take advantage of the additive structure of the indexing sets of input and output data during an algorithmic computation. Why we say that these algorithms rest on the additive structure will become evident when we describe them in other sections later on. This approach did not start with this present work, and below we give a brief account of prior work performed by other authors on the subject of tensor products formulation of FFT algorithms. This present work itself started with the lectures and notes imparted by professor Richard Tolimieri [1], [2], [3], [4], on the subjects of digital signal processing and algorithm design for scientific computation.

Of the additive algorithms described in this work, the Cooley-Tukey algorithm was the first reported to compute the discrete Fourier transform (DFT) of a vector signal much more rapidly than any other available algorithm known at the time. James W. Cooley and John W. Tukey stated in their original paper [5] that, by a process of iterations on a given 1-dimensional array of n data values, the number of arithmetic operations required to compute the DFT could be reduced significantly from the n^2 operations required for straightforward calculation. The Cooley-Tukey FFT algorithm essentially allows for the transformation of a 1-dimensional array of n data values into an m-dimensional array of data values, in every case when n is a composite integer of the form $n = n_1 n_2 ... n_m$. A serial computation of this m-dimensional array of data values produces a set of m recursive equations; each equation describing a modified Fourier sum.

After the reporting of the Cooley-Tukey algorithm by J.W. Cooley and J.W. Tukey in 1965, a great many implementations of this algorithm were performed on sequential machines, improving the computational performance of the algorithms by resorting, sometimes, to clever computer programming techniques, proper handling of the input/output data indexing sets, and by studying the properties of the discrete Fourier transform matrix itself. In 1968, M. Pease [6] proposed to

further improve the computational performance of the Cooley-Tukey algorithm by developing special purpose computers which would take advantage of the fact that some operations in the algorithm could be performed in a parallel fashion, and in this way improve the computational speed. Pease utilized the language of tensor products to express the Cooley-Tukey algorithm; and he used some properties of tensor products to formulate variants of the algorithm which were suitable for implementation on a special purpose computer. After the work reported by Pease, other authors, such as Corinthios [7] and Temperton [8], have used the language of tensor products to describe their work on Cooley-Tukey type algorithms. M.J. Corinthios utilized the technique introduced by Pease to design and implement algorithms obtained by factoring the order of the discrete Fourier transform (DFT) matrix to an arbitrary radix, as opposed to Pease's radix two (2) formulation. C. Temperton has provided tensor products formulations for a number of variants of the Cooley-Tukey FFT algorithm. He has also introduced an algorithm for computing the DFT in an ordered input, ordered output format when the order of the DFT matrix has been factored to a mixed radix.

This work adds to the work of previous authors in the sense that it treats the tensor products formulation of algorithms in a unified manner, identifying the necessary mathematic tools for the analysis of these algorithms, and providing the algorithm user with guidelines which will aid in identifying useful computer implementations. For instance, in a tensor products formulation of a Cooley-Tukey type FFT algorithm, there exist three types of mathematical entities, not including the permutation operations, which can always be identified. These entities are: diagonal operations, which are usually termed phase or twiddle factors, and expressions of the form $(I_r \otimes A)$ or $(B \otimes I_s)$, where A and B are matrices with special properties which we will describe later on in another section. Of the expressions $(I_r \otimes A)$ and $(B \otimes I_s)$, it will be shown that the latter is more suited for vector computer implementation. The former, $(I_r \otimes A)$, it will be shown, adapts itself better to parallel computer implementation. Manipulating these mathematical entities, through the use of tensor products properties, gives us some of the necessary information to effect efficient implementation tasks on given machine architectures.

One of the advantages of using tensor products language to describe FFT al-

gorithms is that this mathematical language may be used as an analytic tool for the study of algorithmic structures for machine hardware and software implementations as well as the identification of new algorithms. For instance, an inherent part in the study of computer implementation of FFT algorithms is the analysis of the data communications aspects of the algorithms which manifest themselves during implementation procedures. These data communication aspects can be best studied, in turn, through the analysis of permutation matrices which appear in our tensor products formulations of the FFT algorithms.

We present, in tensor products form, the description of FFT algorithms with the following objective in mind. To provide the user of these algorithms with guidelines which will enable him to effectively study their implementation on either special purpose or general purpose computers. By "effectively study their implementation," we mean to be able to produce algorithms which best conform to the inherent constraints identified on any given machine hardware architecture.

Fast algorithms for digital signal processing, including fast Fourier transform (FFT) algorithms, are also implemented on special purpose hardware using digital devices. One of the digital hardware devices most commonly used for the implementation of FFT algorithms is the finite impulse response (FIR) filter [9]. Another objective of this work is to present a mathematical characterization of linear shift invariant, finite impulse response (LSI-FIR) filters, and describe how to use the tensor products language as tool to aid in the implementation of these LSI-FIR filters using FFT algorithms. A reason for the mathematical characterization of LSI-FIR filters is that it will help describe and analyze digital signal processing (DSP) applications where these systems are used, and possibly obtain further simplifications of desired implementations.

Chapter I

TENSOR PRODUCTS:

1.1. Tensor Products Properties:

In this section we present some of the basic properties of tensor products which are encountered in the formulations of the algorithms that we will be describing in future sections of this work. Tensor products algebra becomes an important tool for presenting mathematical formulations of digital signal processing algorithms so that these algorithms may be studied and analyzed in a unified format. We first describe the tensor products as bilinear maps and present some of their properties. We then define the tensor product of matrices and describe some of the properties. These properties are very useful in manipulating the Discrete Fourier Transform matrix in order to obtain different formulations of algorithms.

Finite sets, with internal additive property, are used as indexing sets in algorithmic computations of the discrete Fourier transform (DFT). Complex functions may be defined on these sets, and linear vector spaces related to these functions may be identified. The tensor product of two complex functions is defined in this type of scenario. First, we give the following definitions:

Let

$$Z/m = \{0, 1, \ldots, m-1\}, \tag{1}$$

and let the complex field be denoted by C. The set of all functions

$$\begin{aligned} f &: Z/m \longrightarrow C \\ j &\longmapsto f(j) = f_j \end{aligned} \tag{2}$$

forms a linear vector space which we denote by $L(Z/m)$. Define $\underline{f}$ as the m-tuple

$$\underline{f} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{m-1} \end{bmatrix}, \quad f_j = f(j), \quad j \in Z/M \tag{3}$$

The set of all m-tuples $\underline{f}$ forms the linear complex vector space C^m. The set $L(Z/m)$ is isomorphic to C^m.

The set of all functions

$$h : Z/m \times Z/n \longrightarrow C, \tag{4}$$

defined on the cartesian product set $Z/m \times Z/n$, forms a linear vector space which we denote by $L(Z/m \times Z/n)$. By identifying every function $h \in L(Z/m \times Z/n)$ with an $m \times n$ matrix H:

$$H = [h(j,k)]_{\substack{j \in Z/m \\ k \in Z/n}}, \tag{5}$$

it can be shown that the space $L(Z/m \times Z/n)$ is isomorphic to the space $M_{m \times n}(C)$ of $m \times n$ matrices over C.

Given two linear spaces $L(Z/m), L(Z/n)$, we can define an application $\otimes$ from the cartesian product space $L(Z/m) \times L(Z/n)$ to $L(Z/m \times Z/n)$ as follows

$$\begin{gathered} \otimes : L(Z/m) \times L(Z/n) \longrightarrow L(Z/m \times Z/n) \\ (f,g) \longmapsto h = f \otimes g \end{gathered} \tag{6}$$

where h, evaluated at $(j,k) \in Z/m \times Z/n$, is given by

$$h(j,k) = (f \otimes g)(j,k) = f(j) \cdot g(k), \quad j \in Z/m, \quad k \in Z/n. \tag{7}$$

the product on the right being performed in C.

The set

$$\{\delta_{[j]} : j \in Z/m\}, \quad \delta_{[j]} \in L(Z/m), \tag{8}$$

where

$$\delta_{[j]}(t) = \begin{cases} 1, & t = j; \\ 0, & t \neq j \end{cases} \quad j,t \in Z/m, \tag{9}$$

forms a basis for the space $L(Z/m)$, which we call the standard basis for $L(Z/m)$.

Similarly, the set

$$\{\delta_{[k]} : k \in Z/n\}, \quad \delta_{[k]} \in L(Z/n), \tag{10}$$

where

$$\delta_{[k]}(u) = \begin{cases} 1, & u = k; \\ 0, & u \neq k \end{cases} \quad k,u \in Z/n, \tag{11}$$

forms a basis for the space $L(Z/n)$, which we call the standard basis for $L(Z/n)$.

Define $\delta_{[j,k]} \in L(Z/m \times Z/n)$ by

$$\delta_{[j,k]}(t,u) = \begin{cases} 1, & t = j, u = k; \\ 0, & t \neq j, u \neq k \end{cases} \quad j,t \in Z/m, \ k,u \in Z/n \tag{12}$$

The set

$$\{\delta_{[j,k]} : j \in Z/m, \ k \in Z/n\} \tag{13}$$

forms a basis which we call the standard basis for $L(Z/m \times Z/n)$.

For functions $f \in L(Z/m)$, $g \in L(Z/n)$, $h \in L(Z/m \times Z/n)$, we can write the following respective expressions uniquely:

$$\begin{aligned} f &= \sum_{j=0}^{m-1} f(j)\delta_{[j]}\,, \\ g &= \sum_{k=0}^{n-1} g(k)\delta_{[k]}\,, \\ h &= \sum_{j=0}^{m-1}\sum_{k=0}^{n-1} h(j,k)\delta_{[j,k]} \end{aligned} \tag{14}$$

Consider the function $v = f \otimes g$, $v \in L(Z/m) \otimes L(Z/n)$.We can write

$$v = f \otimes g = \Big(\sum_{j=0}^{m-1} f(j)\delta_{[j]}\Big) \otimes \Big(\sum_{k=0}^{n-1} g(k)\delta_{[k]}\Big) \tag{15}$$

Using the linearity property of $L(Z/m) \otimes L(Z/n)$ results in

$$v = f \otimes g = \sum_{j=0}^{m-1}\sum_{k=0}^{n-1} f(j)g(k)\delta_{[j]} \otimes \delta_{[k]} \tag{16}$$

We notice that

$$(\delta_{[j]} \otimes \delta_{[k]})(t,u) = \delta_{[j]}(t) \cdot \delta_{[k]}(u) \equiv \delta_{[j,k]}(t,u), \quad j,t \in Z/m,\ k,u \in Z/n \tag{17}$$

Thus, we write

$$v = f \otimes g = \sum_{j=0}^{m-1}\sum_{k=0}^{n-1} f(j)g(k)\delta_{[j]} \otimes \delta_{[k]} = \sum_{j=0}^{m-1}\sum_{k=0}^{n-1} v(j,k)\delta_{[j,k]} \tag{18}$$

We would like to introduce at this point some of the basic properties of tensor products matrices which are useful for the type of analysis of FFT algorithms described in this work. We also relate some tensor products expressions to specific computer operations performed on selected machine architectures. These identifications will aid in our analysis of the structure of the FFT algorithms, with the objective of determining feasible implementations on given computer architectures; and to possibly produce new variants, from the computer implementation's point of view, of known algorithms. We proceed to give the following definitions.

Let matrices A, B, C be described as follows:

$$A = \big[a_{(t',r')}\big]_{\substack{t' \in \mathbb{Z}/t \\ r' \in \mathbb{Z}/r}} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,r-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,r-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{t-1,0} & a_{t-1,1} & \cdots & a_{t-1,r-1} \end{bmatrix} \tag{19}$$

$$B = [b_{(u',s')}]_{\substack{u'\in Z/u \\ s'\in Z/s}} = \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,s-1} \\ b_{1,1} & b_{1,1} & \cdots & b_{1,s-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{u-1,0} & a_{u-1,1} & \cdots & a_{u-1,s-1} \end{bmatrix} \tag{20}$$

$$C = [c_{(m',n')}]_{\substack{m'\in Z/m \\ n'\in Z/n}} = \begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,n-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m-1,0} & c_{m-1,1} & \cdots & c_{m-1,n-1} \end{bmatrix} \tag{21}$$

The tensor product $A \otimes B$ of the matrices A and B is the matrix C defined by

$$C = A \otimes B = [a_{(t',r')}B]_{\substack{t'\in Z/t \\ r'\in Z/r}} = \begin{bmatrix} a_{(0,0)}B & a_{(0,1)}B & \cdots & a_{(0,r-1)}B \\ a_{(1,0)}B & a_{(1,1)}B & \cdots & a_{(1,r-1)}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{(t-1,0)}B & a_{(t-1,1)}B & \cdots & a_{(t-1,r-1)}B \end{bmatrix} \tag{22}$$

Since C is an $m \times n$ matrix, the integer values m, n are given by

$$m = t \cdot u, \quad n = r \cdot s, \tag{23}$$

and we write C as

$$C = A \otimes B = [a_{(t',r')}] \otimes [b_{(u',s')}] = [c_{(j,k)}]_{\substack{0\le j<m \\ 0\le k<n}} \tag{24}$$

We can see from the expression above that any (j,k)-th entry of the matrix C depends on four (4) indices:

$$t' \in Z/t, \quad r' \in Z/r, \quad u' \in Z/u, \quad s' \in Z/s \tag{25}$$

Thus, we rewrite C as

$$C = [c_{(j,k)}]_{\substack{j\in Z/m \\ k\in Z/n}} = [c_{t',u';r',s'}]_{\substack{t'\in Z/t \\ u'\in Z/u \\ r'\in Z/r \\ s'\in Z/s}} \tag{26}$$

or

$$C = [c_{m',n'}] = [c_{(t',u';r',s')}] = [a_{(t',r')} \cdot b_{(u',s')}] =$$

$$= \begin{bmatrix} c_{(0,0;0,0)} & c_{(0,0;0,1)} & \cdots & c_{(0,0;1,0)} & \cdots & c_{(0,0;r-1,s-1)} \\ c_{(0,1;0,0)} & c_{(0,1;0,1)} & \cdots & c_{(0,1;1,0)} & \cdots & c_{(0,1;r-1,s-1)} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ c_{(0,u-1;0,0)} & c_{(0,u-1;0,1)} & \cdots & c_{(0,u-1;1,0)} & \cdots & c_{(0,u-1;r-1,s-1)} \\ c_{(1,0;0,0)} & c_{(1,0;0,1)} & \cdots & c_{(1,0;1,0)} & \cdots & c_{(1,0;r-1,s-1)} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ c_{(t-1,u-1;0,0)} & c_{(t-1,u-1;0,1)} & \cdots & c_{(t-1,u-1;1,0)} & \cdots & c_{(t-1,u-1;r-1,s-1)} \end{bmatrix} \tag{27}$$

A close look at the matrix C reveals the following ordering imposed on this matrix: Two indices, namely, r', s', remain constant when reading down any given column

of C. The two other indices, namely, t', u', remain constant when reading along any given row of C. The two indices which do not remain constant on any given row or column, are ordered lexicographically. Consider the index pair (t', u') as being an element of the cartesian set $Z/t \times Z/u$. If we order this set lexicographically, we can establish a one-to-one mapping between the ordered elements of this set and the first two indices, namely, t', u', of the entries of the C matrix as we read, in order, down any column of this matrix. In the same manner, a one-to-one mapping may be established between the lexicographically ordered elements of the cartesian set $Z/r \times Z/s$ and the last two indices, namely, r', s', of the entries of the C matrix, again, as we read, in order, along any row of the matrix C.

We can also relate the elements of the set $Z/t \times Z/u$, ordered lexicographically, to the elements of the set Z/m ordered in the natural order, i.e., $Z/m = \{0, 1, \ldots, m-1\}$. A possible mapping is described by the following table:

$$
\begin{aligned}
Z/t \times Z/u &\longrightarrow Z/m \\
(0,0) &\longmapsto 0 \\
(0,1) &\longmapsto 1 \\
(0,2) &\longmapsto 2 \\
&\vdots \\
(0,u-1) &\longmapsto u-1 \\
(1,0) &\longmapsto u \\
(1,1) &\longmapsto u+1 \\
(1,2) &\longmapsto u+2 \\
&\vdots \\
(1,u-1) &\longmapsto 2u-1 \\
&\vdots \\
(t-1,0) &\longmapsto (t-1)u \\
(t-1,1) &\longmapsto (t-1)u+1 \\
(t-1,2) &\longmapsto (t-1)u+2 \\
&\vdots \\
(t-1,u-1) &\longmapsto tu-1
\end{aligned}
\tag{28}
$$

We can also describe the above table by the following mapping:

$$\begin{aligned} &\alpha: Z/t \times Z/u \longrightarrow Z/m \\ &(j_1, j_2) \longmapsto \alpha(j_1, j_2) = (j) = j_1 + tj_2 \end{aligned} \tag{29}$$

In the same manner, we can establish a mapping between the set $Z/r \times Z/s$, ordered lexicographically, and the set Z/n, ordered in the natural order. We describe this mapping in the following way:

$$\begin{aligned} &\beta: Z/r \times Z/s \longrightarrow Z/n \\ &(k_1, k_2) \longmapsto \beta(k_1, k_2) = (k) = k_1 + rk_2 \end{aligned} \tag{30}$$

Using the above mappings, we can now rewrite the tensor product of A and B in the following way:

$$C = A \otimes B = [c_{(j,k)}] = [c(j_1 + tj_2; k_1 + rk_2)] \underset{\substack{k_1 \in Z/r \\ k_2 \in Z/s}}{\substack{j_1 \in Z/t \\ j_2 \in Z/u}} \tag{31}$$

We define the tensor products of two vectors in a similar way. Thus, if we let $\underline{x}$ and $\underline{y}$ be any two vectors of dimensions r and s respectively, the tensor products operation $\underline{x} \otimes \underline{y}$ is defined as

$$\underline{x} \otimes \underline{y} = \begin{bmatrix} x_0\underline{y} \\ x_1\underline{y} \\ x_2\underline{y} \\ \vdots \\ x_{r-1}\underline{y} \end{bmatrix} = \begin{bmatrix} x_0y_0 \\ \vdots \\ x_0y_{(s-1)} \\ \vdots \\ x_{(r-1)}y_0 \\ \vdots \\ x_{(r-1)}y_{(s-1)} \end{bmatrix} \tag{32}$$

Using arbitrary n-th order square matrices, including the identity matrix I_n, we state the following identities which can be verified through direct computation:

$$I_n \otimes I_m = I_{nm} \tag{33}$$

$$(A_0 \otimes A_1 \otimes \ldots \otimes A_{m-1})^{-1} = A_0^{-1} \otimes A_1^{-1} \otimes \ldots \otimes A_{m-1}^{-1} \tag{34}$$

$$(A_0 \otimes A_1 \otimes \ldots \otimes A_{m-1})^{T} = A_0^{T} \otimes A_1^{T} \otimes \ldots \otimes A_{m-1}^{T} \tag{35}$$

$$A_0 \otimes (B_0 + B_1 + \ldots + B_{m-1}) = (A_0 \otimes B_0) + (A_0 \otimes B_1) + \ldots + (A_0 \otimes B_{m-1}) \tag{36}$$

$$(A_0 \otimes A_1 \otimes \ldots \otimes A_{m-1})(B_0 \otimes B_1 \otimes \ldots \otimes B_{m-1}) = (A_0B_0) \otimes (A_1B_1) \otimes \ldots \otimes (A_{m-1}B_{m-1}) \tag{37}$$

$$(A_0 A_1 A_2 \dots A_{m-1}) \otimes I_n = (A_0 \otimes I_n)(A_1 \otimes I_n)(A_2 \otimes I_n)\dots(A_{m-1} \otimes I_n) \quad (38)$$

There some tensor products expressions which may be readily identified with specific operations performed on a given computer. Two of the most important ones are the expressions $(I_r \otimes A)$ and $(B \otimes I_s)$, where A and B are square matrices of arbitrary order. These expressions may be implemented as parallel and vector processing operations, respectively, on machines possessing the required hardware structure. For example, if we let F_s denote the discrete Fourier transform (DFT) matrix of order s, then the expression

$$(F_s \otimes I_r) \quad (39)$$

can be implemented on a machine with vector processor architecture, with at least s vector registers, whose vector length is at least r. We call this expression the s-point Fourier factor on vectors of length r.

The expression

$$(I_r \otimes F_s) \quad (40)$$

may be implemented on a parallel machine with at least r processing units. We term this expression the s point Fourier factor on r parallel units. We would like to point out that when expressing additive FFT algorithms in tensor products form, expressions of the form $(I_r \otimes A)$ or $(B \otimes I_s)$ are prevalent in the formulations. The matrices A, B which appear in these expressions, respectively, are always square matrices corresponding to either Discrete Fourier transform (DFT) matrices or lower order Fourier factors as the ones described above (Eqs. (39), (40)). If either A or B is further composed of of lower Fourier factors, the associated expression can be further expanded until the last non-identity matrix in the in the Fourier factor expression is a Fourier matrix. Thus, for instance, A may be of the form $(I_p \otimes F_q)$ or $(F_p \otimes I_q)$, p, q any integers. In this case, the expression $(I_r \otimes A)$ would have the final form $(I_r \otimes I_p \otimes F_q)$ or $(I_r \otimes F_p \otimes I_q)$, respectively.

In the course of this work, as we try to formulate mathematical expressions describing the various FFT algorithms, we will classify the operations encountered in these algorithms as belonging to one of the type of expressions described above so that we can emphasize the modular nature that the tensor products formulation

brings about. It is important to point out that these operations are the ones, which, when properly matched to the architecture of a given machine during an algorithm implementation, account for the increase in performance when compared with the standard sequential scalar processing. We will also analyze in detail the parallel or vector processing nature of these expressions and study their interrelationships through permutation matrices.

We use, in the following examples, some of the properties of tensor of products described above. The matrices used in these examples, unless otherwise stated, are arbitrary matrices; their order indicated most of the time by subindices. At other times, we use subindices to indicate a sequence of arbitrary matrices of the same order, the order, in these cases, being stated explicitly beforehand. First, let us make the following definition

Let U_m denote the vector defined by

$$U_m = [\,u_0 \quad u_1 \quad \ldots \quad u_{m-1}\,]^T, \quad u_0 = u_1 = \ldots = u_{m-1} = 1 \tag{41}$$

Example 1:

$$(F_n \otimes F_m) = (F_n \otimes F_m)I_{mn}$$

$$(F_n \otimes F_m)I_{mn} = (F_n \otimes F_m)(I_n \otimes I_m)$$

$$(F_n \otimes F_m)(I_n \otimes I_m) = F_nI_n \otimes I_mF_m$$

$$F_nI_n \otimes F_mI_m) = F_nI_n \otimes I_mF_m$$

$$F_nI_n \otimes I_mF_m = (F_n \otimes I_m)(I_n \otimes F_m)$$

Thus, we have

$$(F_n \otimes F_m) = (F_n \otimes I_m)(I_n \otimes F_m)$$

Example 2:

$$(F_n \otimes I_m)(I_n \otimes F_m) = F_nI_n \otimes I_mF_m$$

$$F_nI_n \otimes F_mI_m = I_nF_n \otimes F_mI_m$$

$$I_n F_n \otimes F_m I_m = (I_n \otimes F_m)(F_n \otimes I_m)$$

Thus, we have

$$(F_n \otimes I_m)(I_n \otimes F_m) = (I_n \otimes F_m)(F_n \otimes I_m)$$

Example 3:

$$(F_n \otimes U_m) = (F_n \otimes I_m)(I_n \otimes U_m)$$

For F_n a symmetric matrix, we have

$$(F_n \otimes U_m)^T = F_m \otimes U_m^T$$

Thus, we have

$$F_m \otimes U_m^T = (I_m \otimes U_m^T)(F_m \otimes I_m)$$

Example 4:

Let $C_{m\times 1}$ and $D_{n\times 1}$ be any two matrices of dimensions $m \times 1$ and $n \times 1$, respectively. By direct computation we can show that

$$C_{m\times 1}^T \otimes D_{n\times 1} = D_{n\times 1} C_{m\times 1}^T$$

Taking the transpose, we have

$$C_{m\times 1} \otimes D_{n\times 1}^T = C_{m\times 1} D_{n\times 1}^T$$

In the following examples all the square matrices are considered to be symmetric.

Example 5:

$$F_s = F_s \otimes U_1$$

$$(I_s \otimes U_m)(F_s \otimes U_1) = I_s F_s \otimes U_m U_1 = F_s U_m$$

And, we have

$$(F_s \otimes U_m) = (I_s \otimes U_m) F_s$$

The factor $(I_s \otimes U_m)$ is termed a parallel assignment factor, which, in this case, aids in reducing the number of arithmetic operations involved in the computation $(F_s \otimes U_m)$. Thus, if we let $m = 3$ and $s = 2$, we see that

$$(F_2 \otimes U_3)\underline{x} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} x_0 + x_1 \\ x_0 + x_1 \\ x_0 + x_1 \\ x_0 - x_1 \\ x_0 - x_1 \\ x_0 - x_1 \end{bmatrix}$$

$$(I_2 \otimes U_3)F_2\underline{x} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 + x_1 \\ x_0 - x_1 \end{bmatrix}$$

Example 6:

$$F_s = U_1 \otimes F_s$$

$$(U_m \otimes I_s)(U_1 \otimes F_s) = U_m U_1 \otimes I_s F_s = U_m \otimes F_s$$

Thus, we have

$$U_m \otimes F_s = (U_m \otimes I_s)F_s$$

We term the factor $(U_m \otimes I_s)$ a vector assignment factor.

Example 7:

Taking the transpose of the expression $(F_s \otimes U_m)$ given in Ex. 5 results in

$$F_s \otimes U_m^T = F_s(I_s \otimes U_m^T)$$

The expression $(I_s \otimes U_m^T)$ is termed a parallel pre-addition factor in the sense that the computation $(F_s \otimes U_m^T)\underline{x}$, where $\underline{x}$ is a data vector, can be performed by first performing the computation $(I_s \otimes U_m^T)\underline{x}$ and then allowing F_s to act on the resulting vector.

Example 8:

If we take the transpose of the expression $(U_m \otimes F_s)$ given in Ex. 6, it results in

$$U_m^T \otimes F_s = F_s(U_m^T \otimes I_s)$$

The factor $(U_m^T \otimes I_s)$ is termed a vector pre-addition factor. Letting $m = 4$ and $s = 2$ allows us to obtain the following result for the computation $(U_m^T \otimes F_s)\underline{x}$.

$$(U_4^T \otimes F_2)\underline{x} = [F_2 \quad F_2 \quad F_2 \quad F_2] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \\ x_0 - x_1 + x_2 - x_3 + x_4 - x_5 + x_6 - x_7 \end{bmatrix}$$

$$F_2(U_4^T \otimes I_2) = F_2 \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

$$= F_2 \begin{bmatrix} x_0 + x_2 + x_4 + x_6 \\ x_1 + x_3 + x_5 + x_7 \end{bmatrix} = \begin{bmatrix} x_0 + x_2 + x_4 + x_6 + x_1 + x_3 + x_5 + x_7 \\ x_0 + x_2 + x_4 + x_6 - x_1 - x_3 - x_5 - x_7 \end{bmatrix}$$

Chapter II

PERMUTATION MATRICES

2.1. Properties of Permutation Matrices:

In this section we spend some time describing the permutation matrices that appear in the various formulations of the FFT algorithms, and the role they play in manipulating tensor products formulations of algorithms in order to adapt them to suitable machine hardware architectures. Permutation matrices play a crucial role when trying to obtain variants of FFT algorithms to fit given computer architectures. The actual data flow required to carry out Cooley-Tukey type algorithms can be described by permutation matrices. A type of permutation matrices which play an important role in the manipulation of tensor products expressions for the formulation of FFT algorithms is called "stride" permutation matrices. They are termed "stride" permutations because they can essentially be performed by striding or sampling through the data with a constant distance or length. These permutations are easy to visualize when they are described in terms of two dimensional arrays. We proceed to elaborate on this description.

If we associate an array X with the n-dimensional data vector $\underline{x}$, by writing, in order, down the columns of X the elements of $\underline{x}$, then the "stride by s" permutation matrix $P(n,s)$, or $P_{n,s}$, will be completely defined by the equation

$$\underline{x}_1 = P(n,s)\underline{x}, \quad n = r \cdot s \tag{1}$$

where $\underline{x}_1$ is the data vector associated with the array X_1, and obtained from this array by reading, in order, down the columns of the array. The array X_1 itself is obtained from the array X by performing a matrix transposition. Thus, if we have $n = r \cdot s$, the array X is written as

$$X = \begin{bmatrix} x_0 & x_s & \cdots & x_{(r-1)s} \\ x_1 & x_{s+1} & \cdots & x_{(r-1)s+1} \\ \vdots & & & \\ x_{s-1} & x_{2s-1} & \cdots & x_{n-1} \end{bmatrix} \tag{2}$$

The array $X_1 = X^T$ is written as

$$X_1 = \begin{bmatrix} x_0 & x_1 & \cdots & x_{s-1} \\ x_s & x_{s+1} & \cdots & x_{2s-1} \\ \vdots & & & \\ x_{(r-1)s} & x_{(r-1)s+1} & \cdots & x_{n-1} \end{bmatrix} \tag{3}$$

and the vector $\underline{x}_1$ becomes

$$\underline{x}_1 = \begin{bmatrix} x_0 \\ x_s \\ \vdots \\ x_{(r-1)s} \\ x_1 \\ x_{s+1} \\ \vdots \\ x_{n-1} \end{bmatrix} = P(n,s) \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_s \\ x_{s+1} \\ x_{s+2} \\ \vdots \\ x_{n-1} \end{bmatrix} \tag{4}$$

We provide the following example.

Example 1: For $n = 6$, $r = 2$, $s = 3$, we have

$$\underline{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \longrightarrow X = \begin{bmatrix} x_0 & x_3 \\ x_1 & x_4 \\ x_2 & x_5 \end{bmatrix} \longrightarrow X_1 = X^T = \begin{bmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \end{bmatrix} \longrightarrow \underline{x}_1 = P(6,3)\underline{x} = \begin{bmatrix} x_0 \\ x_3 \\ x_1 \\ x_4 \\ x_2 \\ x_5 \end{bmatrix}$$

For the special case of $n = 2 \cdot m$, the X and X_1 arrays, of dimensions $m \times 2$ and $2 \times m$, respectively, are given by the following expressions:

$$X = \begin{bmatrix} x_0 & x_m \\ x_1 & x_{m+1} \\ \vdots & \vdots \\ x_{m-1} & x_{n-1} \end{bmatrix}, \qquad X_1 = \begin{bmatrix} x_0 & x_1 & \cdots & x_{m-1} \\ x_m & x_{m+1} & \cdots & x_{n-1} \end{bmatrix} \tag{5}$$

The $n \times n$ permutation matrix $P(n,m)$, or $P_{n,m}$, is then given by the rule:

$$\underline{x}_1 = P_{n,m}\underline{x}. \tag{6}$$

The matrix $P_{n,m}$ is usually termed the perfect shuffle permutation matrix whenever $n = 2 \cdot m$. We provide the following examples of perfect shuffle permutation matrices:

Example 2: Take $n = 4$. Then

$$P(4,2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$P(4,2)\underline{x} = \begin{bmatrix} x_0 \\ x_2 \\ x_1 \\ x_3 \end{bmatrix}$$

Example 3: Take $n = 8$. Then

$$P(8,4) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$P(8,4)\underline{x} = \begin{bmatrix} x_0 \\ x_4 \\ x_1 \\ x_5 \\ x_2 \\ x_6 \\ x_3 \\ x_7 \end{bmatrix}$$

Example 4: Take $n = 6$. Then

$$P(6,3) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

and

$$P(6,3)\underline{x} = \begin{bmatrix} x_0 \\ x_3 \\ x_1 \\ x_4 \\ x_2 \\ x_5 \end{bmatrix}$$

The group action of $P(n,2)$ on $\underline{x}$ is found by placing down, in order, the even indexed data points followed by, in order, the odd data points. In particular,

$$P(n,n/2)P(n,2) = I_n. \tag{7}$$

More generally,

$$P(n,r)P(n,s) = I_n, \quad n = r \cdot s. \tag{8}$$

Example 5: Take $n = 4 \cdot 2$. Then

$$P(8,2)\underline{x} = \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \\ x_1 \\ x_3 \\ x_5 \\ x_7 \end{bmatrix}$$

and

$$P(8,2)^2 \underline{x} = \begin{bmatrix} x_0 \\ x_4 \\ x_1 \\ x_5 \\ x_2 \\ x_6 \\ x_3 \\ x_7 \end{bmatrix}$$

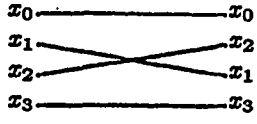
from which we see that

$$P(8,2)^2 = P(8,4).$$

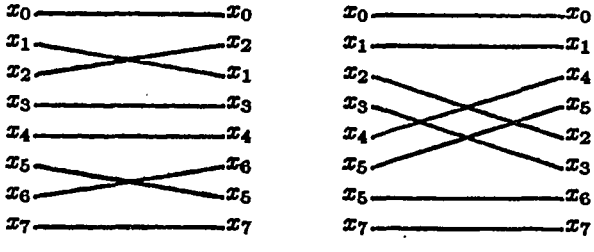
$$P(8,2)^3 = I_8.$$

It is sometimes useful to represent permutations by diagrams which give a picture of data flow.

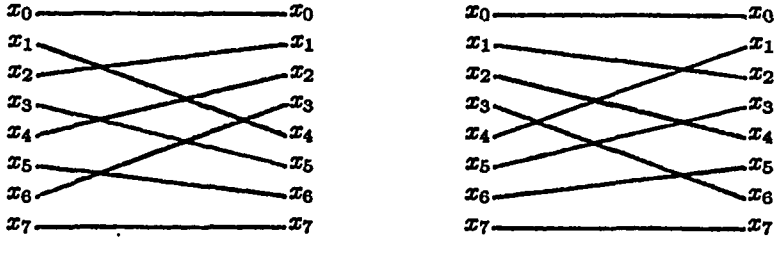
Example 6: The permutation $P(4,2)$ can be represented by



Example 7: The permutation $I_2 \otimes P(4,2)$ and $P(4,2) \otimes I_2$ can be represented by



Example 8: The permutations $P(8,2)$ and $P(8,4)$ can be represented by



In general, an $n \times n$ permutation matrix can be given by a permutation of Z/n. Let Π be a permutation of the indexing set Z/n:

$$\Pi: Z/n \longrightarrow Z/n$$
$$(j) \mapsto \Pi(j) \tag{9}$$

Then, the permutation Π can be described by the table

$$\begin{array}{cc} (j) & \Pi(j) \\ 0 & \Pi(0) \\ 1 & \Pi(1) \\ \vdots & \vdots \\ n-1 & \Pi(n-1) \end{array} \tag{10}$$

Or simply by

$$\Pi = (\Pi(0), \Pi(1), \ldots, \Pi(n-1)) \tag{11}$$

Identify with every permutation Π a permutation matrix denoted by P_Π or $P(\Pi)$ and defined by

$$P_\Pi(\delta_{[j]}) = \delta_{[\Pi^{-1}(j)]}, \quad j \in Z/n \tag{12}$$

where $\delta_{[j]} \in L(Z/n)$ is a standard basis vector, $L(Z/n)$ is the linear vector space formed by all n-point complex sequences, and Π^{-1} is the inverse of the permutation Π.

The matrix P_Π can be written as

$$P_\Pi = [\delta_{[\Pi^{-1}(0)]}\, \delta_{[\Pi^{-1}(1)]} \cdots \delta_{[\Pi^{-1}(n-1)]}] \tag{13}$$

Or

$$P_\Pi = [p_{j,k}]_{0 \le j,k < n} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-1} \\ \vdots & & & \\ p_{n-1,0} & p_{n-1,1} & \cdots & p_{n-1,n-1} \end{bmatrix} \tag{14}$$

where

$$p_{j,k} = \begin{cases} 1, & k = \Pi(j); \quad j,k \in Z/n \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

Thus, in general, the j-th row of P_Π will have a 1 in the k-th column whenever $k = \Pi(j)$ and a 0 otherwise. We write

$$P_\Pi = [p_{j,k}] = [p_{j,\Pi(j)}] \tag{16}$$

We associate with the permutation Π^{-1} (the inverse of Π) a permutation matrix $P_{\Pi^{-1}}$ defined by

$$P_{\Pi^{-1}} = [p_{j,\Pi^{-1}(j)}] = [p_{\Pi(j),j}], \qquad P_\Pi = [p_{j,\Pi(j)}] = [p_{\Pi^{-1}(k),k}] \tag{17}$$

Using (16), (17) and the fact that a permutation matrix is unitary, we arrive at the following result

$$P_\Pi^{-1} = P_\Pi^T = P_{\Pi^{-1}} \tag{18}$$

Define the matrix H_R as the matrix obtained by postmultiplying the matrix H by the permutation matrix P_Π associated with the permutation Π:

$$H_R = P_\Pi H, \quad H = [h_{j,k}]_{0 \le j,k < n} \tag{19}$$

Direct computation shows that H_R is given by

$$H_R = [h_{\Pi(j),k}]_{0 \le j,k < n} \tag{20}$$

Thus, H_R is the matrix obtained by permuting the rows of H.

Define the matrix H_C as the matrix obtained by premultiplying the matrix H by the permutation matrix P_Π associated with the permutation matrix Π:

$$H_C = HP_\Pi, \quad H = [h_{j,k}]_{0 \le j,k < n} \tag{21}$$

Direct computation shows that H_C is given by

$$H_C = [h_{j,\Pi^{-1}(k)}] \tag{22}$$

Thus, H_C is the matrix obtained by permuting the columns of H.

We notice that if we let

$$\underline{h}(k) = \begin{bmatrix} h_{0,k} \\ h_{1,k} \\ h_{2,k} \\ \vdots \\ h_{n-1,k} \end{bmatrix}, \quad k \in Z/n, \tag{23}$$

be the k-th column of the matrix H, then the k-th column of the matrix H_R is given by

$$\underline{h}_R(k) = P_\Pi \underline{h}(k) = \begin{bmatrix} h_{\Pi(0),k} \\ h_{\Pi(1),k} \\ h_{\Pi(2),k} \\ \vdots \\ h_{\Pi(n-1),k} \end{bmatrix} \tag{24}$$

Let E_n be the matrix formed by the vectors of the basis set $\{e_j \delta_{[j]},\ j \in Z/n\}$:

$$E_n = [e_0\delta_{[0]} \quad e_1\delta_{[1]} \quad e_2\delta_{[2]} \quad \cdots \quad e_{n-1}\delta_{[n-1]}] \tag{25}$$

The matrix E_n is the diagonal matrix

$$E_n = diag\, \underline{e} = \begin{bmatrix} e_0 & & & & \\ & e_1 & & & \\ & & e_2 & & \\ & & & \ddots & \\ & & & & e_{n-1} \end{bmatrix} \tag{26}$$

where $\underline{e}$ is the n-dimensional vector

$$\underline{e} = [e_0 \quad e_1 \quad e_2 \quad \cdots \quad e_{n-1}]^T \tag{27}$$

The product $P_\Pi E_n$ of the matrix E_n and the permutation matrix P_Π is given by

$$\begin{aligned} P_\Pi E_n &= [e_{\Pi(j),k}] \\ &= [e_0\delta_{[\Pi^{-1}(0)]} \quad e_1\delta_{[\Pi^{-1}(1)]} \quad \cdots \quad e_{n-1}\delta_{[\Pi^{-1}(n-1)]}] \end{aligned} \tag{28}$$

Direct computation shows that the last expression in the identity above (Eq. (28)) can be rewritten as

$$diag\begin{bmatrix} e_{\Pi(0)} \\ e_{\Pi(1)} \\ \vdots \\ e_{\Pi(n-1)} \end{bmatrix} \cdot [\delta_{[\Pi^{-1}(0)]} \quad \delta_{[\Pi^{-1}(1)]} \quad \cdots \quad \delta_{[\Pi^{-1}(n-1)]}] \tag{29}$$

or

$$P_\Pi E_n = diag\begin{bmatrix} e_{\Pi(0)} \\ e_{\Pi(1)} \\ \vdots \\ e_{\Pi(n-1)} \end{bmatrix} \cdot P_\Pi \tag{30}$$

This last equation allows us to make the following observation:

<u>Observation 1:</u>

$$P_\Pi E_n P_\Pi^{-1} = diag[P_\Pi \cdot \underline{e}] \tag{31}$$

We can arrive at the same observation in a slightly different way by using some of the results given above. We start by using Eq. (22) to obtain the following result

$$E_n P_\Pi^{-1} = E_n P_{\Pi^{-1}} = [e_{j,\Pi(k)}] \tag{32}$$

If the above expression (Eq. (32)) is postmultiplied by the permutation matrix P_Π, we obtain

$$P_\Pi E_n P_\Pi^{-1} = P_\Pi E_n P_{\Pi^{-1}} = [e_{\Pi(j),\Pi(k)}] \tag{33}$$

Since E_n is a diagonal matrix, we have

$$P_\Pi E_n P_\Pi^{-1} = [e_{\Pi(j),\Pi(j)}] = diag[P_\Pi \cdot \underline{e}] \tag{34}$$

Suppose that n is the composite $n = r \cdot s$, Take any $j \in Z/n$. This element can be written uniquely as

$$j = c + dr, \quad 0 \leq c < r, \quad 0 \leq d < s \tag{35}$$

Consider any $k \in Z/n$. This element can be written uniquely as

$$k = a + bs, \quad 0 \leq a < s, \quad 0 \leq b < r \tag{36}$$

Corresponding to this factorization of n ($n = r \cdot s$), we define the permutation Π of Z/n by setting

$$\Pi(b + ar) = (a + bs), \quad 0 \leq a < s, \quad 0 \leq b < r \tag{37}$$

The matrix representing the permutation Π is given by

$$P_\Pi = [p_{j,k}] = [p_{j,\Pi(j)}] \tag{38}$$

which satisfies the condition

$$P_\Pi \underline{x} = \underline{y} \tag{39}$$

where

$$y_j = x_{\Pi(j)}, \quad 0 \leq j < n \tag{40}$$

Thus, the action of P_Π on this n-dimensional vector $\underline{x}$ results in

$$P_\Pi \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} x_{\Pi(0)} \\ x_{\Pi(1)} \\ x_{\Pi(2)} \\ \vdots \\ x_{\Pi(n-1)} \end{bmatrix} = \begin{bmatrix} x_0 \\ x_s \\ x_{2s} \\ \vdots \\ x_{(r-1)s+s-1} \end{bmatrix} \tag{41}$$

This results states that, in general, if $n = r \cdot s$ and Π is the permutation of Z/n given by (37), then

$$P_\Pi = P_{n,s} \tag{42}$$

where $P_{n,s}$ is the stride by s permutation matrix defined Eq. (4).

We have previously associated an n-th order permutation matrix P_Π or $P(\Pi)$ with a permutation Π of order n. Direct computation shows that the mapping

$$\Pi \longrightarrow P_\Pi \tag{43}$$

is a group-isomorphism from the group of permutations of Z/n under composition onto the group of $n \times n$ permutation matrices under matrix product. In fact

$$P(\Pi_2 \cdot \Pi_1) = P(\Pi_1)P(\Pi_2), \tag{44}$$

$$P(\Pi^{-1}) = P(\Pi)^{-1}. \tag{45}$$

Consider the set of $n \times n$ permutation matrices

$$\{P(n,s): s|n\}. \tag{46}$$

We will describe the permutation matrices in this set in terms of the unit group $U(n-1)$ of $Z/n-1$. The unit group $U(n-1)$ is given by

$$U(n-1) = \{0 \leq t < n-1: (t, n-1) = 1\}. \tag{47}$$

If $t \in U(n-1)$, then multiplication by t mod $(n-1)$ is a bijection of the set

$$\{0, 1, \ldots, n-2\}. \tag{48}$$

Define the permutation Π_t of Z/n by the two rules

$$\Pi_t(j) \equiv jt \bmod (n-1), \quad 0 \leq j < n-1, \tag{49}$$

$$\Pi_t(n-1) = (n-1) \tag{50}$$

Observe that if $t|n$ the $(t, n-1) = 1$ and we can define Π_t.

<u>Example 9:</u> Take $n = 12, s = 3$ and $t = 4$. Then

$$\Pi_4 = (0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11)$$

We see that

$$\Pi_4(a+3b) = b+4a, \quad 0 \leq a < 3, \ 0 \leq b < 4,$$

and

$$P(\Pi_4) = P(12, 4).$$

<u>Observation 2:</u> If $n = st$ then

$$P(\Pi_t) = P(n, t) \tag{51}$$

<u>Proof:</u> We will show that

$$\Pi_t(a+bs) = b+at, \quad 0 \leq a < s, \ 0 \leq b < t. \tag{52}$$

First if $0 \leq a < s$ then $0 \leq at < n-1$ and

$$\Pi_t(a) = at. \tag{53}$$

Since $n = st \equiv 1 \bmod (n-1)$ we have

$$\Pi_t(a+s) = 1+at, \quad 0 \le a < s. \tag{54}$$

Continuing in this way, we complete the proof of the theorem.

Consider the set of $n \times n$ permutation matrices

$$\{P(\Pi_t): t \in U(n-1)\} \tag{55}$$

This set is a group isomorphic to $U(n-1)$. In fact

$$P(\Pi_s)P(\Pi_t) = P(\Pi_u), \quad u \equiv st \bmod (n-1) \tag{56}$$

$$P(\Pi_s)^{-1} = P(\Pi_{s^{-1}}), \quad s^{-1} \text{ taken mod } (n-1) \tag{57}$$

Observation 3: If $n = 2^m$, the set

$$\{P(2^m, 2^j): 0 \le j < m\} \tag{58}$$

is a cyclic group generated by $P(2^m, 2)$. In fact,

$$P(2^m, 2^j)P(2^m, 2^k) = P(2^m, 2^{j+k}) \tag{59}$$

where $j+k$ is taken mod m.

Proof: Consider integers $0 \le j, k < m$. If $j+k < m$ then

$$2^{j+k} \equiv 2^j 2^k \bmod (2^m - 1) \tag{60}$$

which along with (51) and (56) shows that

$$P(2^m, 2^j)P(2^m, 2^k) = P(2^m, 2^{j+k}). \tag{61}$$

Suppose that $m \le j+k$. Set $l = j+k-m$. we have $0 \le l < m$ and $l \equiv j+k \bmod m$. From

$$2^l(2^m - 1) < 2^{j+k} < (2^l + 1)(2^m - 1) \tag{62}$$

It follows that

$$2^l \equiv 2^j 2^k \bmod (2^m - 1) \tag{63}$$

Applying observation 2 completes the proof of the theorem.

More generally, we have the following result which we offer without proof

Observation 4: If p is a prime then the set

$$\{P(p^r, s): s|p^r\} \tag{64}$$

is a cyclic group of order r generated by $P(p^r, p)$.

We now proceed to prove an important special case of the commutation theorem which describes the action of the permutation matrix $p_{n,s}$, $s|n$, on tensor products of matrices. Take $n = r \cdot s$. Let $\underline{x}$ and $\underline{y}$ be vectors of dimension r and s, respectively. Then

$$\underline{x} \otimes \underline{y} = \begin{bmatrix} x_0\underline{y} \\ x_1\underline{y} \\ \vdots \\ x_{r-1}\underline{y} \end{bmatrix} \tag{65}$$

The vector $\underline{x} \otimes \underline{y}$ is equivalent to the 2-dimensional array (see (1) – (6) above)

$$[x_0\underline{y}\, x_1\underline{y} \ldots x_{r-1}\underline{y}] \tag{66}$$

The operation $P_{n,s}(\underline{x} \otimes \underline{y})$ is equivalent to

$$[x_0\underline{y}\, x_1\underline{y} \ldots x_{r-1}\underline{y}]^T = [y_0\underline{x}\, y_1\underline{x} \ldots y_{s-1}\underline{x}] \tag{67}$$

The above identity allows us to write the following expression which can be verified by direct computation

$$P_{n,s}(\underline{x} \otimes \underline{y}) = \underline{y} \otimes \underline{x} \tag{68}$$

We now state the following theorem

Theorem 1: If A is an $r \times r$ matrix and B is an $s \times s$ matrix then

$$P_{n,s}(A \otimes B)P_{n,s}^{-1} = B \otimes A \tag{69}$$

Proof: Set $\underline{z} = \underline{x} \otimes \underline{y}$ where $\underline{x}$ is an r-dimensional vector and $\underline{y}$ is an s-dimensional vector. Then, by definition,

$$(A \otimes B)(\underline{x} \otimes \underline{y}) = A\underline{x} \otimes B\underline{y} \tag{70}$$

Applying (68),

$$P_{n,s}((A \otimes B)\underline{z}) = B\underline{y} \otimes A\underline{x} \tag{71}$$

Arguing in the same way

$$(B \otimes A)P_{n,s}\underline{z} = B\underline{y} \otimes A\underline{x}, \tag{72}$$

proving the theorem.

Corollary:

$$P_{n,s}(I_r \otimes F(s))P_{n,s}^{-1} = F(s) \otimes I_r \tag{73}$$

Chapter III

TENSOR PRODUCTS FORMULATION OF KAHANER'S ALGORITHM:

3.1. Basic Description:

In his paper [10], D. K. Kahaner describes a procedure for factoring the Fourier matrix F_N when $N = p^\gamma$, p and γ any integers. Kahaner's factorization method produces, up to matrix factor expansion, what is commonly known as the Cooley-Tukey (C-T) decimation in frequency algorithm. In this section we describe Kahaner's algorithm in detail, and then present it a tensor products formulation. This will aid in the understanding of the tensor products language used to analyze other FFT algorithms later on.

3.2. Kahaner's Mathematical Formulation:

Kahaner starts by defining the discrete Fourier transform of N equally spaced data points x_k, $k = 0, \ldots, N-1$:

$$F_r = \frac{1}{N}\sum_{k=0}^{N-1} x_k e^{-\frac{2\pi i}{N}rk} = \frac{1}{N}\sum_{k=0}^{N-1} x_k a^{rk}, \quad 0 \le r < N, \quad a = e^{-2\pi i/N} \tag{1}$$

In matrix form,

$$\overline{F} = \frac{1}{N}A\overline{X}, \quad \overline{F}^T = [F_0 \quad F_1 \quad \ldots \quad F_{N-1}] \tag{2}$$

where A is the matrix:

$$A = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & a & a^2 & \ldots & a^{N-1} \\ 1 & a^2 & a^4 & \ldots & a^{2(N-1)} \\ \vdots & & & & \\ 1 & a^{N-1} & a^{2(N-1)} & \ldots & a^{(N-1)(N-1)} \end{bmatrix} \tag{3}$$

Kahaner proceeds to write a general expression for F_r, $r = 0, 1, \ldots, N-1$:

$$F_r = \frac{1}{N}\sum_{k=0}^{N-1} x_k a^{rk} = \frac{1}{N}\sum_{k=0}^{M-1}\left\{\sum_{t=0}^{p-1} x_{k+tM} a^{r(k+tM)}\right\}, \quad M = p^{\gamma-1}, \quad r = 0, 1, \ldots, N-1 \tag{4}$$

Writing $r = pm + l$, the following expression is obtained,

$$F_r = F_{pm+l} = \frac{1}{N}\sum_{k=0}^{M-1}\left\{\sum_{t=0}^{p-1} x_{k+tM}\theta^{lt}a^{lk}a^{pmk}\right\}, \quad \theta = e^{-2\pi i/p} \tag{5}$$

For each fixed l, the following vector is formed:

$$[F_l \quad F_{p+l} \quad F_{2p+l} \quad \ldots \quad F_{(M-1)p+l}]^T, \quad l = 0, 1, \ldots, p-1 \tag{6}$$

After some algebraic manipulations, this vector is written as,

$$\frac{1}{N}\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & a^p & a^{2p} & \dots & a^{(M-1)p} \\ \vdots & & & & \\ 1 & a^{(M-1)p} & a^{2(M-1)p} & \dots & a^{(M-1)(M-1)p} \end{bmatrix} \begin{bmatrix} \sum_{t=0}^{p-1} x_{tM}\theta^{lt} \\ \sum_{t=0}^{p-1} x_{1+tM}\theta^{lt}a^{l} \\ \vdots \\ \sum_{t=0}^{p-1} x_{(M-1)+tM}\theta^{lt}a^{l(M-1)} \end{bmatrix} \tag{7}$$

or

$$\begin{bmatrix} F_l \\ F_{p+l} \\ \vdots \\ F_{(M-1)p+l} \end{bmatrix} \equiv \frac{1}{N} B \overline{X}_l^1, \quad l = 0, \dots, p-1 \tag{8}$$

By writing these p vectors $(l = 0, 1, \dots, p-1)$ in a column, the following result is obtained:

$$\overline{F}^{(1)} = \begin{bmatrix} \begin{bmatrix} F_0 \\ F_p \\ \vdots \\ F_{(M-1)p+0} \end{bmatrix} \\ \begin{bmatrix} F_1 \\ F_{p+1} \\ \vdots \\ F_{(M-1)p+1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} F_{p-1} \\ F_{2p-1} \\ \vdots \\ F_{pM-p+p-1} \end{bmatrix} \end{bmatrix} = \frac{1}{N} \begin{bmatrix} B & & & \\ & B & & \\ & & \ddots & \\ & & & B \end{bmatrix} \begin{bmatrix} \overline{X}_0^{(1)} \\ \overline{X}_1^{(1)} \\ \vdots \\ \overline{X}_{p-1}^{(1)} \end{bmatrix} \tag{9}$$

where,

$$\begin{bmatrix} \overline{X}_0^{(1)} \\ \overline{X}_1^{(1)} \\ \vdots \\ \overline{X}_{p-1}^{(1)} \end{bmatrix} = \overline{X}^{(1)} = \Delta^{(0)} \overline{X} \tag{10}$$

and,

$$\Delta^{(0)} = \begin{bmatrix} I & I & \dots & I \\ D & \theta D & \dots & \theta^{p-1} D \\ D^2 & (\theta D)^2 & \dots & (\theta^{p-1} D)^2 \\ \vdots & & & \\ D^{p-1} & (\theta D)^{p-1} & \dots & (\theta^{p-1} D)^{p-1} \end{bmatrix}, \quad D = \operatorname{diag}\left[1, a, a^2, \dots, a^{M-1}\right] \tag{11}$$

The vector $\overline{F}^{(1)}$ differs from the vector $\overline{F}$ by the permutation matrix π_0:

$$\overline{F} = \pi_0 \overline{F}^{(1)} = \frac{\pi_0}{N} \begin{bmatrix} B & & & \\ & B & & \\ & & \ddots & \\ & & & B \end{bmatrix} \Delta^{(0)} \overline{X} \tag{12}$$

Since the matrix B has the general form of the matrix A, this result is generalized:

$$\overline{F} = \frac{\pi_0}{N}\hat{\pi}_1\hat{\pi}_2\ldots\hat{\pi}_{\gamma-1}\begin{bmatrix}\Delta^{(\gamma-1)} & & \\ & \ddots & \\ & & \Delta^{(\gamma-1)}\end{bmatrix}\ldots\begin{bmatrix}\Delta^{(1)} & & \\ & \ddots & \\ & & \Delta^{(1)}\end{bmatrix}\left[\Delta^{(0)}\right]\overline{X} \tag{13}$$

where

$$\hat{\pi}_i = \begin{bmatrix}\pi_i & & & \\ & \pi_i & & \\ & & \ddots & \\ & & & \pi_i\end{bmatrix}, \quad i = 1, 2, \ldots, \gamma - 1 \tag{14}$$

$$\Delta^{(j)} = \begin{bmatrix} I & I & \cdots & I \\ D^{(j)} & \theta D^{(j)} & \cdots & \theta^{p-1}D^{(j)} \\ \vdots & & & \\ (D^{(j)})^{p-1} & (\theta D^{(j)})^{p-1} & \cdots & (\theta^{p-1}D^{(j)})^{p-1}\end{bmatrix} \tag{15}$$

$$D^{(j)} = \begin{bmatrix}1 & & & \\ & a^{p^j} & & \\ & & \ddots & \\ & & & a^{(p^{\gamma-j-1}-1)p^j}\end{bmatrix}, \quad j = 0, 1, \ldots, \gamma - 1, \quad D^{\gamma-1} \equiv [1] \tag{16}$$

3.3. Tensor Products Formulation:

We proceed to describe this matrix factorization method in tensor products form. We start by introducing the following definitions:

The matrix $P_{n,s}$, of order $n = r \cdot s$ is called the stride by s permutation matrix, and is defined by

$$P_{n,s} \cdot d = (d_0, d_s, d_{2s}, \ldots, d_1, d_{s+1}, \ldots, d_{(r-1)s+s-1})^T \tag{16}$$

for

$$d = (d_0, d_1, \ldots, d_{s-1}, d_s, \ldots, d_{(r-1)s+s-1})^T \tag{17}$$

The diagonal matrix $D_{n,s}$, of order s is defined by

$$D_{n,s} = \text{diag}\left[1, w_n, w_n^2, \ldots, w_n^{s-1}\right], \quad w_n = e^{-2\pi i/n} \tag{18}$$

The twiddle factor (phase factor) matrix $T_{n,s(r)}$, of order n, is defined as the direct sum of s diagonal matrices $D_{r,n/s}$ of order n/s:

$$T_{n,s(r)} = \sum_{0 \le j < s} \oplus D^j_{r,n/s} \tag{19}$$

If $n = r \cdot s$, then

$$T_{n,s(n)} = T_{n,s} = \sum_{0 \le j < s} \oplus D^j_{n,n/s} = \sum_{0 \le j < s} \oplus D^j_{n,r} \tag{20}$$

To arrive at a general form for the Fourier matrix F_N, $N = p^\gamma$, expressed in tensor products, we start with an expression for the Fourier matrix F_p and use this expression, and the Cooley-Tukey decimation in frequency algorithm expressed in tensor products, to obtain higher order Fourier matrices expressed in tensor products form:

$$F_N = F_{p^\gamma} = F_p = P_{p,1}\Delta^{(\gamma-1)} = T_{p,p}\Delta^{(\gamma-1)} = I_p\Delta^{(\gamma-1)}, \quad \gamma = 1 \tag{21}$$

where

$$\Delta^{(\gamma-1)} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w & w^2 & \dots & w^{p-1} \\ 1 & w^2 & w^4 & \dots & w^{2(p-1)} \\ \vdots & & & & \\ 1 & w^{p-1} & w^{2(p-1)} & \dots & w^{(p-1)(p-1)} \end{bmatrix}, \quad w = e^{-2\pi i/p} \tag{22}$$

The Cooley-Tukey decimation in frequency algorithm allows us to write F_{p^2} in the following form:

$$F_{p^2} = P_{p^2,p}(I_p \otimes F_p)T_{p^2,p}(F_p \otimes I_p) \tag{23}$$

Using

$$\Delta^{(\gamma-2)} = T_{p^2,p}(F_p \otimes I_p) \tag{24}$$

we get,

$$F_{p^2} = P_{p^2,p}(I_p \otimes F_p)\Delta^{(\gamma-2)} \tag{25}$$

where,

$$T_{p^2,p} = \sum_{0 \le j < p} \oplus D^j_{p,p}, \quad D_{p,p} = [1, w_p, w_p^2, \dots, w_p^{p-1}] \tag{26}$$

Using the expression for F_p given above, we obtain,

$$F_{p^2} = P_{p^2,p}(I_p \otimes P_{p,1})(I_p \otimes \Delta^{(\gamma-1)})\Delta^{(\gamma-2)} \tag{27}$$

For the matrix F_{p^3}, we write down again the expression for the Cooley-Tukey decimation in frequency algorithm:

$$F_{p^3} = P_{p^3,p^2}(I_p \otimes F_{p^2})T_{p^3,p}(F_p \otimes I_{p^2}) \tag{28}$$

where,

$$T_{p^3,p} = \sum_{0 \le j < p} \oplus D^j_{p^2,p}, \quad D_{p^2,p} = \left[1, w_{p^2}, w_{p^2}^2, \dots, w_{p^2}^{p-1}\right] \tag{29}$$

Using,

$$\Delta^{(\gamma-3)} = T_{p^3,p}(F_p \otimes I_{p^2}), \tag{30}$$

and the expression given above for F_{p^2}, we get,

$$F_{p^3} = P_{p^3,p^2}(I_p \otimes P_{p^2,p})(I_{p^2} \otimes P_{p,1}) \cdot (I_{p^2} \otimes \Delta^{(\gamma-1)})(I_p \otimes \Delta^{(\gamma-2)})\Delta^{(\gamma-3)} \tag{31}$$

Continuing in the same manner:

$$F_{p^4} = P_{p^4,p^3}(I_p \otimes F_{p^3})T_{p^4,p}(F_p \otimes I_{p^3}) \tag{32}$$

Using

$$\Delta^{(\gamma-4)} = T_{p^4,p}(F_p \otimes I_{p^3}) \tag{33}$$

we get,

$$F_{p^4} = P_{p^4,p^3}(I_p \otimes P_{p^3,p^2})(I_{p^2} \otimes P_{p^2,p})(I_{p^3} \otimes P_{p,1}) \cdot (I_{p^3} \otimes \Delta^{(\gamma-1)})(I_{p^2} \otimes \Delta^{(\gamma-2)})(I_p \otimes \Delta^{(\gamma-3)})\Delta^{(\gamma-4)} \tag{34}$$

In general, for a Fourier matrix $F_{p^{\gamma-k}}$, $0 \leq k < \gamma$, **we write:**

$$F_{p^{\gamma-k}} = P_{p^{\gamma-k},p^{\gamma-k-1}}(I_p \otimes F_{p^{\gamma-k-1}})T_{p^{\gamma-k},p}(F_p \otimes I_{p^{\gamma-k-1}}) \tag{35}$$

We, again, set

$$\Delta^{(\gamma-(\gamma-k))} = \Delta^{(k)} = T_{p^{\gamma-k},p}(F_p \otimes I_{p^{\gamma-k-1}}) \tag{36}$$

where

$$T_{p^{\gamma-k},p} = \sum_{0 \leq j < p} \oplus D^j_{p^{\gamma-k},p^{\gamma-k-1}}, \quad 0 \leq k < \gamma ; \tag{37}$$

and the identity

$$w_{p^{\gamma-k-1}} = e^{-2\pi i p(p^k/p^\gamma)} = (e^{-2\pi i p^k/p^\gamma})^p = w^p_{p^{\gamma-k}} \tag{38}$$

may be used to write down the elements of $D_{p^{\gamma-k},p^{\gamma-k-1}}$, $0 \leq k < \gamma$

The general expression for $F_{p^{\gamma-k}}$ **thus becomes:**

$$F_{p^{\gamma-k}} = P_{p^{\gamma-k},p^{\gamma-k-1}}(I_p \otimes P_{p^{\gamma-k-1},p^{\gamma-k-2}})(I_{p^2} \otimes P_{p^{\gamma-k-2},p^{\gamma-k-3}}) \ldots \ldots (I_{p^{\gamma-k-1}} \otimes P_{p,1})(I_{p^{\gamma-k-1}} \otimes \Delta^{(\gamma-1)})(I_{p^{\gamma-k-2}} \otimes \Delta^{(\gamma-2)}) \ldots (I_p \otimes \Delta^{(k+1)})\Delta^{(k)} \tag{39}$$

Chapter IV

TENSOR PRODUCTS FORMULATIONS OF FFT ALGORITHMS:

4.1. Basic Definitions:

We start with this section to describe in tensor products form those FFT algorithms which are most commonly known, and which exhibit the additive property. We will use the tensor products properties and, especially, the properties of the permutation matrices described above to analyze the interrelationship among these algorithms; and we will describe how to use these properties in order to obtain implementations which fit given architectures. First, we would like to introduce the following definitions.

The diagonal matrix $D_{n,s}$, of order s is defined by

$$D_{n,s} = \text{diag}\left[1, w_n, w_n^2, \ldots, w_n^{s-1}\right], \qquad w_n = e^{-2\pi i/n} \tag{1}$$

The twiddle factor (phase factor) matrix $T_{n,s(r)}$, of order n, is defined as the direct sum of s diagonal matrices $D_{r,n/s}$ of order n/s:

$$T_{n,s(r)} = \sum_{j=0}^{s-1} \oplus D_{r,n/s}^{j} \tag{2}$$

If $n = r.s$, then

$$T_{n,s(n)} = T_{n,s} = \sum_{j=0}^{s-1} \oplus D_{n,n/s}^{j} = \sum_{j=0}^{s-1} \oplus D_{n,r}^{j} \tag{3}$$

The discrete Fourier transform (DFT) of an n-point sequence x is defined as

$$y(k) = \sum_{0 \le j < n} x(j) w_n^{jk}, \quad 0 \le k < n; \quad w_n = e^{-2\pi i/n} \tag{4}$$

where $y(k)$ represents the kth-term of the DFT sequence y. Written in matrix form, we have

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & w_n & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ \vdots & & & & & \\ 1 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} \tag{5}$$

We denote the above matrix by F_n or $F(n)$, and call it the DFT matrix of order n.

4.2. Cooley-Tukey Algorithm:

The general case Cooley-Tukey FFT algorithm is presented in this section in tensor products form for a composite of the form $n = r \cdot s$. In this case a sequence x, which serves as the input data to be transformed, is represented as a 2-dimensional array which we denote by X. The transformed output sequence y is also expressed as a 2-dimensional array which we denote by Y. These representations are the key to the reduction in the number of operations required to compute the DFT of a given sequence. Through these identifications, the original DFT sum is turned into a set of 2 recursive equations, each equation describing a modified Fourier sum. We first state the identifications explicitly and then proceed to formulate the Cooley-Tukey algorithm, expressing it in tensor products form.

Using correspondences of the form:

$$\begin{aligned} &\alpha : Z/s \times Z/r \longrightarrow Z/n \\ &(j_2, j_1) \mapsto \alpha(j_2, j_1) = (j) = j_2 + sj_1 \end{aligned} \tag{1}$$

$$\begin{aligned} &\beta : Z/r \times Z/s \longrightarrow Z/n \\ &(k_1, k_2) \mapsto \beta(k_1, k_2) = (k) = k_1 + rk_2 \,, \end{aligned} \tag{2}$$

we obtain 2-dimensional array representations of the sequences x and y:

$$X(j_2, j_1) = x(j_2 + sj_1), \quad 0 \le j_1 < r, \quad 0 \le j_2 < s. \tag{3}$$

$$Y(k_1, k_2) = y(k_1 + rk_2), \quad 0 \le k_1 < r, \quad 0 \le k_2 < s. \tag{4}$$

Let $X_1 = X^T$ and let $\underline{x}_1$ be the vector formed by reading, in order, down the columns of X_1. Then, by definition (4) given in chapter II, we have

$$\underline{x}_1 = P_{n,s}\underline{x}, \quad n = r \cdot s. \tag{5}$$

where $\underline{x}$ is the vector representation of the input sequence x. We can write

$$X_1(j_1, j_2) = x(j_2 + sj_1), \quad 0 \le j_1 < r, \quad 0 \le j_2 < s. \tag{6}$$

The expression for the for the DFT of the sequence x:

$$y_k = \sum_{j=0}^{n-1} w_n^{jk} x_j \,, \quad 0 \le k < n, \quad w_n = e^{-2\pi i/n}. \tag{7}$$

can be rewritten as

$$Y(k_1,k_2)=\sum_{j_2=0}^{s-1}\sum_{j_1=0}^{r-1} w_n^{(j_2+sj_1)(k_1+rk_2)}X_1(j_1,j_2) \tag{8}$$

Now,

$$(j_2+sj_1)(k_1+rk_2)\equiv j_2k_1+j_1k_1s+j_2k_2r \mod n. \tag{9}$$

Noticing that

$$w_n^s = w_{r\cdot s}^s = w_r, \quad w_n^r = w_{r\cdot s}^r = w_s, \quad w_n^n = 1, \tag{10}$$

we rewrite (8) as

$$Y(k_1,k_2) = \sum_{j_2=0}^{s-1}\Big(\sum_{j_1=0}^{r-1} X_1(j_1,j_2)w_r^{j_1k_1}\Big)w_n^{j_2k_1}w_s^{j_2k_2} \tag{11}$$

If we look closely at the inner Fourier sum

$$Y_1(k_1,j_2) = \sum_{j_1=0}^{r-1} X_1(j_1,j_2)w_r^{j_1k_1}, \tag{12}$$

we notice that it computes, for each evaluation $0 \le j_2 < s$, the r-point discrete Fourier transform of the j_2th-column of X_1 and it places the result in the j_2th-column of Y_1. If we let $\underline{y}_1$ be the vector formed by reading, in order, down the columns of Y_1, then we can write the following expression

$$\begin{bmatrix} y_1(0,0)\\ y_1(1,0)\\ \vdots\\ y_1(r-1,0)\\ \vdots\\ y_1(0,s-1)\\ \vdots\\ y_1(r-1,s-1)\end{bmatrix} = \begin{bmatrix} 1 & \dots & 1 & & & & \\ \vdots & & & & & & \\ 1 & \dots & w_r^{(r-1)(r-1)} & & & & \\ & & & \ddots & & & \\ & & & & 1 & \dots & 1\\ & & & & \vdots & & \\ & & & & 1 & \dots & w_r^{(r-1)(r-1)}\end{bmatrix}\cdot \underline{x}_1 \tag{13}$$

which, using tensor products notation, can be written as

$$\underline{y}_1 = (I_s\otimes F_r)\underline{x}_1 \tag{14}$$

Using (5), we rewrite this expression as

$$\underline{y}_1 = (I_s\otimes F_r)P_{n,s}\underline{x} \tag{15}$$

The next stage of the computation:

$$Y_2(k_1,j_2) = Y_1(k_1,j_2)w_n^{j_2k_1} \tag{16}$$

can be given by the diagonal matrix multiplication

$$\underline{y}_2 = T_{n,s}\underline{y}_1 \tag{17}$$

where $T_{n,s}$ was previously defined by (3) above.

We complete the computation by

$$Y(k_1,k_2) = \sum_{j_2=0}^{s-1} Y_2(k_1,j_2)w_s^{j_2k_2}. \tag{18}$$

If we denote by $\underline{y}_2$ the vector obtained by writing down, in order, the columns of Y_2, we can write the following matrix-vector product

$$\begin{bmatrix} y(0,0) \\ \vdots \\ y(r-1,0) \\ y(0,1) \\ \vdots \\ y(r-1,1) \\ y(0,2) \\ \vdots \\ y(r-1,2) \\ \vdots \\ y(0,s-1) \\ \vdots \\ y(r-1,s-1) \end{bmatrix} = \begin{bmatrix} I_r & I_r & I_r & \dots & I_r \\ I_r & w_sI_r & w_s^2I_r & \dots & w_s^{s-1}I_r \\ I_r & w_s^2I_r & w_s^4I_r & \dots & w_s^{s-2}I_r \\ \vdots & & & & \\ I_r & w_s^{s-1}I_r & w_s^{s-2}I_r & \dots & w_sI_r \end{bmatrix} \cdot \begin{bmatrix} y_2(0,0) \\ \vdots \\ y_2(r-1,0) \\ y_2(0,1) \\ \vdots \\ y_2(r-1,1) \\ y_2(0,2) \\ \vdots \\ y_2(r-1,2) \\ \vdots \\ y_2(0,s-1) \\ \vdots \\ y_2(r-1,s-1) \end{bmatrix}, \tag{19}$$

which, written in tensor products form, results in the expression

$$\underline{y} = (F(s)\otimes I_r)\underline{y}_2 \tag{20}$$

Combining the above results we arrive at the tensor products formulation of the Cooley-Tukey FFT algorithm for computing a given n-point sequence x:

$$\underline{y} = (F(s)\otimes I_r)T_{n,s}(I_s\otimes F_r)P_{n,s}\underline{x} \tag{21}$$

which produces the following factorization for the Fourier matrix $F(n)$:

$$F(n) = (F(s)\otimes I_r)T_{n,s}(I_s\otimes F_r)P_{n,s} \tag{22}$$

The above factorization is commonly known as the decimation in time (DIT) Cooley-Tukey 2-factor FFT algorithm. Performing a matrix transposition on both sides of the equation results in the decimation in frequency (DIF) FFT algorithm:

$$F(n) = P_{n,s}^{-1}(I_s\otimes F_r)T_{n,s}(F(s)\otimes I_r) \tag{23}$$

4.3. General 2^k FFT Algorithm:

Historically, radix-2 FFT algorithms were the first to be designed and used extensively in scientific applications [11] due to the ease of hardware and software implementations. We present in this section the general Cooley-Tukey decimation in time (DIT) and decimation in frequency (DIF) algorithms for the case of $n = 2^k$. These algorithms lead to other well known algorithms which we will describe later on.

The bit-reversal permutation matrix Q is uniquely defined by the condition

$$Q(\underline{x}_1 \otimes \ldots \otimes \underline{x}_k) = (\underline{x}_k \otimes \ldots \otimes \underline{x}_1), \tag{1}$$

where $\underline{x}_j$ is a 2-dimensional vector. The matrix Q actually corresponds to the indexing set permutation Π given by bit-reversal. Explicitly, each integer $0 \leq j < n$ can be uniquely written as

$$j = a_0 + 2a_1 + \ldots + 2^{k-1}a_{k-1}, \quad 0 \leq a_j < 2, \tag{2}$$

and we call the ordered k-tuple

$$(a_0, a_1, \ldots, a_{k-1}), \tag{3}$$

the binary bit representation of j. Define the permutation

$$\Pi(a_0, a_1, \ldots, a_{k-1}) = (a_{k-1}, \ldots, a_1, a_0). \tag{4}$$

The corresponding permutation matrix satisfies (1). For the case when $n = 2^k$ the following identity can be established

$$Q(n) = (P(4) \otimes I_{n/4}) \ldots (P(n/2) \otimes I_2) P(n), \tag{5}$$

where

$$P(n/2^{j-1}) = (n/2^{j-1}, n/2^j), \quad j = 1, 2, \ldots, k \tag{6}$$

Setting $N_j = 2^{k-j+1}$, we can write

$$T_j = I_{n/N_j} \otimes T(N_j), \quad j = 1, 2, \ldots, k \tag{7}$$

where

$$T(N_j) = T_{N_j/2}(N_j) = T_{N_j,2} = \begin{bmatrix} I_{N_j/2} & \\ & D_{N_j,N_j/2} \end{bmatrix}, \quad j = 1,2,\ldots,k \tag{8}$$

By substituting inductively (6) and (7) on (22) of §4.2, we arrive at the general formula for the DIT Cooley-Tukey FFT Algorithm:

$$F(n) = (F(2) \otimes I_{n/2})T_1 \ldots T_{k-1}(I_{n/4} \otimes F(2) \otimes I_2)(I_{n/2} \otimes F(2))Q \tag{9}$$

An alternate Fourier matrix factorization known as the Gentleman-Sande [12] radix-2 FFT algorithm, is obtained by simply taking the matrix transpose of (9) resulting in

$$F(n) = Q(I_{n/2} \otimes F(2))T_{k-1}(I_{n/4} \otimes F(2) \otimes I_2) \ldots T_1(F(2) \otimes I_{n/2}). \tag{10}$$

We can organize the computation of the Gentleman-Sande radix-2 FFT algorithm into the sequence of k operations

$$X_j = T_j(I_{n/N_j} \otimes F(2) \otimes I_{N_{j+1}}), \quad j = 1,2,\ldots,k \tag{11}$$

followed by the output permutation Q. In each stage, $n/2$ 2-point discrete Fourier transforms are computed; but, the data flow varies from stage to stage. Viewed as a vector operation, one effect of varying the data flow is to change the vector length during the computation. In the j-th stage, the operation

$$I_{n/N_j} \otimes F(2) \otimes I_{N_{j+1}}, \tag{12}$$

is n/N_j independent 2-point discrete Fourier transforms on vectors of length N_{j+1}. In particular, during the computation, vector length varies from $n/2$ to 1.

4.4. Pease FFT Algorithm:

In [6], M. C. Pease designed a variation of the Cooley-Tukey FFT algorithm he asserts is "better to parallel processing in a special purpose machine." One of the main features of the Pease FFT algorithm is that it has constant data flow in all stages of the computation. To accomplish this, the commutation theorem of tensor matrices described in the in the section on permutation matrices is used. We now derive the general case Pease FFT.

Set

$$P = (n, 2), \quad n = 2^k. \tag{1}$$

Direct computation shows that

$$P^j = P(n, 2^j); \tag{2}$$

and, in particular, $p^k = I_n$. **We apply the commutation theorem in the form**

$$I_{n/N_j} \otimes F(2) \otimes I_{N_{j+1}} = P^j (I_{n/2} \otimes F(2)) P^{-j} \tag{3}$$

Set

$$C = I_{n/2} \otimes F(2) \tag{4}$$

We can view the operation C **as** $n/2$ **concurrent 2-point discrete Fourier transforms. The commutation theorem in form expressed in (3) can be introduced into the radix-2 Gentleman-Sande FFT to obtain the following factorization**

$$F(n) = QCT_{k-1}P^{k-1}CP^{-(k-1)} \ldots P^2 CP^{-2} T_1 PCP^{-1} \tag{5}$$

where Q **and** T_j **have the same meaning as described in §4.3. Introducing brackets, we can write**

$$F(n) = Q(CT_{k-1}P^{k-1}) \ldots (CP^2 T_1 P)(CP^{-}1) \tag{6}$$

Using $P^{-1} = P^{k-1}$, **we have the set of formulas,**

$$\begin{aligned} CP^{-2}T_1P &= CT_1'P^{-1} \\ CP^{-3}T_2P^2 &= CT_2'P^{-1} \\ &\vdots \\ CT_{k-1}P^{k-1} &= CT_{k-1}'P^{-1} \end{aligned} \tag{7}$$

where T'_j is the diagonal matrix

$$T'_j = P^{-(j+1)} T_j P^{j+1}, \tag{8}$$

This discussion leads to the following theorem due to Pease.

Theorem 1: If $n = 2^k$ and $N_j = 2^{k-j+1}$, then

$$F(n) = Q(CT'_{k-1}P^{-1}) \dots (CT'_1 P^{-1})(CP^{-1}) \tag{9}$$

where Q is the bit-reversal permutation matrix and

$$\begin{aligned} C &= I_{n/2} \otimes F(2), \\ T'_j &= P^{-(j+1)}(I_{n/N_j} \otimes T(N_j))P^{j+1}, \\ P &= P(n,2). \end{aligned} \tag{10}$$

If we set $T_0 = T_n$ and

$$Y_j = CT'_j P^{-1}, \quad 0 \le j < k, \tag{11}$$

then the Fourier matrix factorization given by (6) determines a k stage computation given by the sequence

$$Y_0, Y_1, \dots, Y_{k-1}, \tag{12}$$

followed by bit-reversal Q. In the j-th stage, we begin permuting the data by the perfect shuffle matrix P^{-1}. We then perform the twiddle factor operation T'_j and complete the computation by the operation C which can be carried out by $n/2$ concurrent 2-point discrete Fourier transforms. Each stage has the same data flow with, only the twiddle factor varying.

4.5. Korn-Lambiotte FFT Algorithm:

A vectorized variation of the Pease FFT algorithm is presented by Korn-Lambiotte in [13] for implementation on the STAR 100 computer. We will present two vector variations. In each case, we replace the parallelizable operation C, given by (4) in §4.4, by the vectorizable operation

$$G = F(2) \otimes I_{n/2} \tag{1}$$

In the first variation, we refactor the operation Y_j, given by (11) in §4.4, as a vectorizable operation. By the commutation theorem of tensor matrices,

$$C = I_{n/2} \otimes F(2) = P^{-1}(F(2) \otimes I_{n/2})P = P^{-1}GP. \tag{2}$$

We can rewrite Y_j as

$$Y_j = P^{-1}GPT_j'P^{-1}. \tag{3}$$

Using

$$T'' = PT_j'P^{-1}, \tag{4}$$

where, T_j' is given by (8) in §4.4, results in

$$Y_j = P^{-1}GT_j''. \tag{5}$$

This last expression allows us to state the following theorem

Theorem 1: **IF $n = 2^k$, then**

$$F(n) = Q(P^{-1}GT_{k-1}'')\dots(P^{-1}G), \tag{6}$$

where $N_j = 2^{k-j+1}$ and

$$\begin{aligned} G &= F(2) \otimes I_{n/2}, \\ T_j'' &= P^{-j}(I_{n/N_j} \otimes T(N_j))P^{+j}, \\ P &= P(n,2). \end{aligned} \tag{7}$$

The second variation is obtained by choosing a different bracketing for (5) given in §4.4. Using (9) given in §4.4, we can write

$$F(n) = (P^{-1}G)(P^{-1}D_{k-1}G)\dots(P^{-1}DG), \tag{8}$$

where D_j is the diagonal matrix

$$D_j \equiv P^{-(j-1)}T_jP^{j-1} = T(N_j) \otimes I_{n/N_j}. \tag{9}$$

The commutation theorem was utilized in order to obtain the last equality in (9).

4.6. Stockham Auto-Sort FFT Algorithm:

The effort required to perform an n-point bit-reversal operation, either on output or input data, can be an important segment of the overall task of an FFT computation on a given vector machine. In Cochran et al. [14], an FFT algorithm, attributed to T. G. Stockham, is described which computes the FFT in proper order, without requiring permutation either after or before computational stages. We call such an algorithm an auto-sort algorithm. C. Temperton [8] examines in detail the implementation of the Stockham FFT and mixed-radix generalizations on the CRAY-1 computer.

The main idea underlying the Stockham auto-sort FFT is to distribute the n-point bit-reversal operation throughout the different stages of the computation. At the same time, all the discrete Fourier transform computations are vectorized. To design the Pease FFT, permutation matrices of the form $P(n,m)$, $m|n$, were introduced into the Cooley-Tukey factorization in order to vectorize the discrete Fourier transform factors. The Stockham FFT is derived by vectorizing these factors by bit reversal. Consider the factorization

$$F(n) = Q(n)(I_{n/2} \otimes F(n))T_{k-1}(I_{n/4} \otimes F(2) \otimes I_2) \dots T_1(F(2) \otimes I_{n/2}), \tag{1}$$

where $n = 2^k$. From the formulas

$$Q(n)(I_{n/2} \otimes F(2))Q(n)^{-1} = C \tag{2}$$

$$(Q(n/2) \otimes I_2)(I_{n/4} \otimes F(2) \otimes I_2)(Q(n/2) \otimes I_2)^{-1} = C \tag{3}$$

$$\vdots$$

$$(Q(4) \otimes I_{n/4})(I_2 \otimes F(2) \otimes I_{n/4})(Q(4) \otimes I_{n/4})^{-1} = C \tag{4}$$

where $C = F(2) \otimes I_{n/2}$, we have

$$F(n) = CQ(n)T_{k-1}(Q(n/2) \otimes I_2)^{-1}C(Q(n/2 \otimes I_2) \dots T_1 C. \tag{5}$$

Direct computation shows that

$$R(n/2^j) = (Q(n/2^j) \otimes I_{2^j})(Q(n/2^{j+1}) \otimes I_{2^{j+1}})^{-1} \tag{6}$$

$$R(n/2^j) = (Q(n/2^j)(Q(n/2^{j+1}) \otimes I_2)) \otimes I_{2^j} \tag{7}$$

$$R(n/2^j) = P(n/2^j, 2) \otimes I_{2^j} \tag{8}$$

where we have used the fact that the bit-reversal matrix and its inverse are equal. The above discussion allows us to state the Stockham FFT algorithm.

Theorem 1: If $n = 2^k$, then

$$F(n) = (CT'_{k-1}R(n))(CT'_{k-2}R(n/2)\dots(CT'_1R(4))C \tag{9}$$

where

$$C = F(2) \otimes I_{n/2} \tag{10}$$

$$R(n/2^j) = P(n/2^j, 2) \otimes I_{2^j} \tag{11}$$

$$T'_j = Q(2^{j+1})T_jQ(2^{j+1})^{-1} \tag{12}$$

If we set

$$T'_{k-1} = Q(n)T_{k-1}Q(n)^{-1} \tag{13}$$

$$T'_{k-2} = (Q(n/2) \otimes I_2)T_{k-2}(Q(n/2) \otimes I_2)^{-1} \tag{14}$$

$$\vdots$$

$$T'_1 = (Q(4) \otimes I_{n/4})T_1(Q(4) \otimes I_{n/4})^{-1}, \tag{15}$$

then, $T'_1, T'_2, \dots, T'_{k-1}$ are diagonal matrices; and we can rewrite (5) as

$$F(n) = (CT'_{k-1}R(n))(CT'_{k-2}R(n/2))\dots(CT'_1R(4))C, \tag{16}$$

where

$$R(n) = Q(n)(Q(n/2) \otimes I_2)^{-1}, \tag{17}$$

$$R(n/2) = (Q(n/2) \otimes I_2)(Q(n/4) \otimes I_4)^{-1}, \tag{18}$$

$$\vdots$$

$$R(4) = Q(4) \otimes I_{n/4} \tag{19}$$

4.7. Mixed-Radix Cooley-Tukey FFT Algorithm:

The mixed-radix decimation in time (DIT) and decimation in frequency (DIF) Cooley-Tukey algorithms are simply a generalization of the two-factor Cooley-Tukey FFT presented in §4.2. We proceed to give a general description of these algorithms.

Suppose

$$n = n_1 n_2 \dots n_k, \tag{1}$$

$$N_j = n_j \dots n_k. \tag{2}$$

We see that

$$n/N_1 = 1,\ n/N_2 = n_1,\ \dots\ n/N_k = n_1 \dots n_{k-1}. \tag{3}$$

Then,

$$F(n) = Q(I_{n/N_k} \otimes F(n_k)) \dots T_1(F(n_1) \otimes I_{N_2}), \tag{4}$$

where Q is a permutation matrix (described below) and T_j is a diagonal matrix given by the formula

$$T_j = (I_{n/N_j} \otimes T_{N_{j+1}}(N_j)), \quad 1 \le j < k \tag{5}$$

where we take $T_0 = I_n$. The computation can be decomposed into sequence of operations

$$X_1, X_2, \dots, X_k, \tag{6}$$

followed by the output permutation Q. In particular, vector length varies during these computation stages as

$$N_2, N_3, \dots, N_k, 1 \tag{7}$$

The output permutation Q is the mixed-radix analog of the bit-reversal permutation. It is given, by induction, by the following factorization

$$Q = P(n, N_2)(I_{n/N_2} \otimes P(N_2, N_3)) \dots (I_{n/N_{k-1}} \otimes P(N_{k-1}, N_k)) \tag{8}$$

Direct computation shows that

$$Q(\underline{x}_1 \otimes \dots \otimes \underline{x}_k) = \underline{x}_k \otimes \dots \otimes \underline{x}_1 \tag{9}$$

Formula (9) serves to uniquely define Q.

To obtain a decimation in time (DIT) factorization, we simply take the transpose of both sides of (4). The result is

$$F(n) = (F(n_1) \otimes I_{N_2}) T_1 \dots (I_{n/N_k} \otimes F(n_k)) Q^{-1} \tag{10}$$

4.8. Agarwal-Cooley Mixed-Radix FFT Algorithm:

A generalization of the radix-2 Pease FFT to mixed-radix was designed by Agarwal-Cooley [15] for implementation on the IBM 3090 Vector Facility. The goal, as stated, is to produce "a fully vectorized mixed-radix FFT algorithm requiring all load/store [operations] with only small stride." The load/store operations refer to computer instructions devoted to the transfer of data between the vector registers of the vector facility and the storage facility. We proceed to describe the general case.

For

$$n = n_1, \ldots n_k, \tag{1}$$

we begin with the factorization

$$F(n) = (F(n_1) \otimes I_{N_2}) T_1 \ldots (I_{n/N_k} \otimes F(n_k)) Q^{-1}. \tag{2}$$

Set

$$C_j = F(n_j) \otimes I_{n/N_j}, \tag{3}$$

$$P_j = P(n, N_j), \quad N = n_j \ldots n_k. \tag{4}$$

Using the formulas

$$I_{n/N_j} \otimes I_{N_{j+1}} = P_j^{-1} C_j P_j, \tag{5}$$

we can rewrite (2) as

$$F(n) = C_1 T_1 \ldots P_j^{-1} C_j P_j T_j \ldots P_k^{-1} C_k P_k Q^{-1} \tag{6}$$

Set

$$T_j' = P_j T_j P_j^{-1}. \tag{7}$$

Then, T_j' is the diagonal matrix

$$T_j' = T_{N_{j+1}}(N_j) \otimes I_{n/N_j} \tag{8}$$

and we can rewrite (6) as

$$F(n) = (C_1 T_1')(C_2 T_2' P_2 P_3^{-1}) \ldots (C_k P_k) Q^{-1} \tag{9}$$

The j-th factor (from the left) is

$$C_j T_j' P_j P_{j+1}^{-1} \tag{10}$$

Through direct computation we can show that

$$P(n, n_j) = P_j P_{j+1}^{-1} = P(n, N_j) P(n, n/N_{j+1}) \tag{11}$$

which allows us to state the general theorem:

Theorem 1: **If $n = n_1 \dots n_k$, then**

$$F(n) = ((F(n_1) \otimes I_{n/n_1}) T_1')((F(n_2) \otimes I_{n/n_2}) T_2' P(n, n_2)) \dots ((F(n_k) \otimes I_{n/n_k}) P(n, n_k)) Q^{-1}, \tag{12}$$

where T_j' is given by (8).

After the initial data transposition Q^{-1}, we have k computational stages. A typical stage consists of the sequence of operations

1. $P(n, n_j)$
2. T_j'
3. $F(n_j) \otimes I_{n/n_j}$.

4.9. Temperton Mixed-Radix Auto-Sort FFT Algorithm

The generalization of the mixed-radix auto-sort FFT algorithm is due to C. Temperton [8]. We proceed to describe this algorithm below.

Denote by Q_j the data transposition operation corresponding to the ordered factorization

$$n/N_{j+1} = n_1 \dots n_j, \tag{1}$$

and set

$$R_j = Q_j \otimes I_{N_{j+1}}, \quad 2 \leq j \leq k. \tag{2}$$

In particular, $R_k = Q$.Then, the DIF mixed-radix FFT algorithm can be rewritten as

$$F(n) = (C_k R_k R_{k-1}^{-1}) \dots (T_j' C_j R_j R_{j-1}^{-1}) \dots (T_1' C_1) \tag{3}$$

where

$$C_j = F(n_j) \otimes I_{n/n_j} \tag{4}$$

$$T_j' = R_j T_j R_j^{-1}. \tag{5}$$

The matrices T_j' are diagonal matrices and are called the twiddle factors of the factorization of $F(n)$ in Eq. (3) above. Since

$$R_j R_{j-1}^{-1} = (Q_j (Q_{j-1}^{-1} \otimes I_{n_j})) \otimes I_{N_{j+1}} \tag{6}$$

$$R_j R_{j-1}^{-1} = P(n/N_{j+1}, n_j) \otimes I_{N_{j+1}}, \tag{7}$$

we have

$$F(n) = (C_k P(n, n_k)) \dots (T_j' C_j (P(n/N_{j+1}, n_j) \otimes I_{N_{j+1}})) \dots (T_1' C_1). \tag{8}$$

As bracketed, we have a computation requiring k stages to be performed. When in the j-th stage, the data permutation is given by

$$P(n/N_{j+1}, n_j) \otimes I_{N_{j+1}} \tag{9}$$

This expression should be compared with the data permutation operation

$$P(n, n_j) \tag{10}$$

required in the j-th stage of the Agarwal-Cooley FFT algorithm. In the auto-sort FFT we no longer require n-point data transposition on output, or input; but, we pay the price of replacing computation stage data permutations (10) with those given in (9).

Chapter V

COMPUTER IMPLEMENTATION OF FFT ALGORITHMS:

5.1. Factor Decomposition:

One of the objectives of this work is to present an analysis on the computer implementation of additive FFT algorithms. When trying to implement an FFT algorithm on a computer, the given hardware architecture configuration evokes inherent limitations on the feasibility of the implementation. Our objective is to investigate these inherent limitations, and to formulate guidelines which aid in the designing of algorithms which take these limitations into consideration in order to produce optimum results relative to the identified constraints of the given computer hardware architecture.

In our study of computer implementation of FFT algorithms, we use the language of tensor products to present formulations of commonly known algorithms. The purpose of these formulations is to allow us to describe each algorithm as a series of factors or mathematical expressions which we will analyze, each one separately, and will try to implement them in a manner that best adapts to the given computer architecture; i.e., by matching the mathematical expressions to computer operations which use the architectural features of the given machine in order to perform the most efficient computations relative to some known standards.

As an example of the factor decomposition of an algorithm, we present the general two-factor Cooley-Tukey algorithm. The general two-factor tensor products formulation of the Cooley-Tukey decimation in time algorithm may be expressed as follows:

Let $n = r \cdot s$, where r, s are any integers. The factor decomposition of the Fourier matrix F_n, which produces the Cooley-Tukey algorithm, is given by

$$F_n = F(n) = (F_s \otimes I_r) T_{n,s} (I_s \otimes F_r) P_{n,s} \qquad (1)$$

where $T_{n,s}$ is an nth-order diagonal matrix, termed the twiddle or phase factor matrix; and $P_{n,s}$, or $P(n,s)$, is a permutation matrix of order n termed the "stride by s" permutation matrix.

The expressions $(F_s \otimes I_r)$, $(I_s \otimes F_r)$ are termed Fourier factors matrices, and they represent the main bulk of the computational effort. A question which may be

posed to an algorithm user is how to best implement all of the factors stated above on a vector computer?. The answer to this question resides on the manipulation of the above Fourier matrix or FFT algorithm formulation through the properties of tensor properties, with the purpose of obtaining a formulation that matches effectively the architecture of the machine. In the case of this particular Cooley-Tukey formulation, a first improvement may be to use the commutation theorem of tensor products matrices defined by

$$P^{-1}(n,s)(I_s \otimes F_r)P(n,s) = (F_r \otimes I_s) \tag{2}$$

in order to obtain the following formulation

$$F_n = (F_r \otimes I_r)T_{n,s}P(n,s)P^{-1}(n,s)(I_s \otimes F_r)P(n,s) \tag{3}$$

$$F_n = (F_s \otimes I_r)T_{n,s}P(n,s)(F_r \otimes I_s) \tag{4}$$

This formulation would be an improvement over the previous formulation if the architecture allows the performing of the permutation $P(n,s)$ with relative ease, in addition to the requirements needed to be met about the number of vector registers and the length of these registers. Before we elaborate on these requirements, we would like to point out why this formulation would be a better formulation. We will show in the next section that the Fourier factor $(F_r \otimes I_s)$, which written in matrix form is

$$(F_r \otimes I_s) = \begin{bmatrix} I_s & I_s & \dots & I_s \\ I_s & w_r I_s & \dots & w_r^{r-1} I_s \\ \vdots & & & \\ I_s & w_r^{r-1} I_s & \dots & w_r I_s \end{bmatrix}, \tag{5}$$

is better suited for implementation on a vector computer than the Fourier factor $(I_s \otimes F_r)$. In fact, the Fourier factor $(F_r \otimes I_s)$ may be implemented on a vector machine with at least r vector registers of register length at least s. If this requirements are met, then we would have a better implementation.

Expressions of the form $(I_r \otimes A)$, or $(B \otimes I_s)$, are prevalent in the tensor product formulations of additive FFT algorithms. The matrices A, B usually take the form of $(I_p \otimes F_q)$, or $(F_p \otimes I_q)$, where F_p, F_q are Fourier matrices. Thus, it is important to have an understanding of expressions of the form $(I_r \otimes F_p \otimes I_q)$. These expressions, in turn, depending on the underlying computer architecture, can be implemented as either vector or parallel operations. We proceed to explain below how these implementations may be carried out.

5.2. Vector Computer Implementation of Fourier Factors:

The expression $(I_r \otimes F_p \otimes I_q)$ may be implemented on a machine possessing a vector architecture. The basic requirements imposed on this architecture are that it must have at least p vector registers, and that the length of these registers must be at least q. If these requirements are met, then we can start by rewriting the expression

$$(I_r \otimes F_p \otimes I_q) \tag{1}$$

in the form

$$(I_r \otimes (F_p \otimes I_q)), \tag{2}$$

and analyzing the term

$$(F_p \otimes I_q) \tag{3}$$

This term, written in matrix form, becomes

$$F_p \otimes I_q = \begin{bmatrix} I_q & I_q & \cdots & I_q \\ I_q & w_p I_q & \cdots & w_p^{p-1} I_q \\ \vdots & & & \\ I_q & w_p^{p-1} I_q & \cdots & w_p I_p \end{bmatrix} \tag{4}$$

This matrix form reveals the vector nature of the expression $(F_p \otimes I_q)$, if the computation $(F_p \otimes I_q)\underline{x}$ were to be effected on a vector computer. This is due to the fact that the data vector $\underline{x}$ could be loaded, in order, with stride 1, onto p vector registers and then the computation could be performed on these vectors. To be precise, the vector $\underline{x}$, of length pq, could be partitioned into p smaller vectors of length q, namely, $\underline{x}_0, \underline{x}_1, \cdots, \underline{x}_{p-1}$, where

$$\underline{x}_0 = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{q-1} \end{bmatrix}, \quad \underline{x}_1 = \begin{bmatrix} x_q \\ x_{q+1} \\ \vdots \\ x_{2q-1} \end{bmatrix}, \quad \cdots, \quad \underline{x}_{p-1} = \begin{bmatrix} x_{(p-1)q} \\ x_{(p-1)q+1} \\ \vdots \\ x_{pq-1} \end{bmatrix} \tag{5}$$

Thus, the computation $(F_p \otimes I_q)\underline{x}$ could be written as

$$\begin{bmatrix} I_q & I_q & \cdots & I_q \\ I_q & w_p I_q & \cdots & w_p^{p-1} I_q \\ \vdots & & & \\ I_q & w_p^{p-1} I_q & \cdots & w_p I_q \end{bmatrix} \begin{bmatrix} \underline{x}_0 \\ \underline{x}_1 \\ \vdots \\ \underline{x}_{p-1} \end{bmatrix} = \begin{matrix} \underline{x}_0 + \underline{x}_1 + \ldots + \underline{x}_{p-1} \\ \underline{x}_0 + w_p \underline{x}_1 + \ldots + w_p^{p-1} \underline{x}_{p-1} \\ \vdots \\ \underline{x}_0 + w_p^{p-1} \underline{x}_1 + \ldots + w_p \underline{x}_{p-1} \end{matrix} \tag{6}$$

This expression describes the vector computation, where each term in each summation represents a vector register.

The whole expression $(I_r \otimes (F_p \otimes I_q))$ can be written in matrix form in the following way:

$$(I_r \otimes (F_p \otimes I_q)) = \begin{matrix} "0" \\ "1" \\ \vdots \\ "r" \end{matrix} \begin{pmatrix} F_p \otimes I_q & & & \\ & F_p \otimes I_q & & \\ & & \ddots & \\ & & & F_p \otimes I_q \end{pmatrix} \tag{7}$$

Thus, the term I_r may be interpreted as indicating repetition. Summarizing, the expression

$$(I_r \otimes F_p \otimes I_q) \tag{8}$$

may be implemented on a vector computer by repeating the operation $(F_p \otimes I_q)$ (a vector operation) r times, in a parallel fashion, if possible; where the expression $(F_p \otimes I_q)$ itself could be implemented by using p vector processors of vector length q.

5.3. Parallel Implementation of Fourier Factors:

Using the commutation theorem of tensor products matrices, we can take the expression

$$(I_r \otimes F_p \otimes I_q) \tag{1}$$

and rewrite as

$$(I_q \otimes (I_r \otimes F_p)) = P(n,q)((I_r \otimes F_p) \otimes I_q)P^{-1}(n,q) \tag{2}$$

where

$$P(n,q), \quad n = rpq \tag{3}$$

denotes a "stride by q" permutation matrix.

We use the properties of tensor products to rewrite $(I_q \otimes (I_r \otimes F_p))$ as

$$(I_{qr} \otimes F_p) \tag{4}$$

This last expression, written in matrix form, becomes

$$(I_{qr} \otimes F_p) = \begin{matrix} \text{"0"} \\ \text{"1"} \\ \text{"2"} \\ \vdots \\ \text{"}qr-1\text{"} \end{matrix} \begin{pmatrix} F_p & & & & \\ & F_p & & & \\ & & F_p & & \\ & & & \ddots & \\ & & & & F_p \end{pmatrix} \tag{5}$$

This matrix representation reveals the parallel nature of the computation $(I_{qr} \otimes F_p)\underline{x}$, where $\underline{x}$ is a data vector of length rpq. This can be seen by thinking of the data vector $\underline{x}$ as a set of qr vectors of length p, namely, $\underline{x}_0, \underline{x}_1, \cdots, \underline{x}_{qr-1}$, where

$$\underline{x}_0 = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}, \quad \underline{x}_1 = \begin{bmatrix} x_p \\ x_{p+1} \\ \vdots \\ x_{2p-1} \end{bmatrix}, \quad \cdots \quad , \quad \underline{x}_{qr-1} = \begin{bmatrix} x_{(qr-1)p} \\ x_{(qr-1)p+1} \\ \vdots \\ x_{rpq-1} \end{bmatrix}; \tag{6}$$

and assigning each vector $\underline{x}_m$, $0 \leq m < qr$, to a given processing unit on a parallel computer. This implies that for a parallel machine to implement the expression $(I_{qr} \otimes Fp)$, in a parallel fashion, it must possess at least qr individual units.

5.4. Computer Implementation of Stride Permutations:

As we have stated in the section on permutation matrices, the permutation operations on a given computer dictate the manner in which the data needs to be presented to the computational stages of a given algorithm implementation. These permutation operations can be thought of as specialized data assignments in that, for the case of the stride permutation matrices, their action on a given data vector results in a reordering of the elements of the data vector according to a simple rule. We present in this section some thoughts as to how these permutations may be implemented on a given computer, and set the stage for future study of the procedures involved in these computer implementations.

5.4.1. Vector Implementation of Stride Permutations:

The stride by s permutation matrix $P(n, s)$ may be implemented using a vector computer in the following manner:

For $n = r \cdot s$,

1) Load s vector registers, of vector length r, with data from memory (the data vector $\underline{x}$), using stride s.

2) Load the data from the s vector registers to memory, with stride 1.

The procedure for loading the vector registers from memory ought to be done in such a manner that, at the end of the process, the content of each vector register would be the same as a column of the array X_1.

5.4.2. Parallel Implementation of Stride Permutations:

For the case of $n = r \cdot s$, the permutation matrix $P(n, s)$ may be performed on a parallel machine in the following way:

1) Load a vector segment, of length r, from "global" memory to each "local" memory on s separate units. This must be done using stride s.

2) Load to the global memory the vector data segment on the s separate processing units using stride 1.

Chapter VI

LSI-FIR SYSTEMS:

6.1. The Importance of LSI-FIR Systems:

This work presents a description of the discrete Fourier transform (DFT) operator, and its importance in the analysis of linear shift invariant, finite impulse response (LSI-FIR) systems. LSI-FIR systems arise frequently in the processing of finite length sequences. Their importance stems from the fact that the response of an LSI-FIR system to an input sequence may be interpreted as a periodic sequence; i.e., since the response sequence is finite, say, of length n, it allows for its representation as a periodic sequence of period n. Through this representation, the processing of finite length signals using LSI-FIR systems may be analyzed through the study of cyclic convolution. The cyclic convolution operation turns the linear space of n-point complex sequences into an algebra. Another important operation defined in this algebra (actually, in an algebra isomorphic to this algebra) is the point-wise multiplication of two n-point sequences, this operation being termed the Hadamard product [16] of two sequences. The cyclic convolution and the Hadamard product operations are related, through the DFT operator, using what is commonly known as the cyclic convolution theorem.

When LSI-FIR systems are used for the processing of finite length sequences, an opportunity arises to use some of the properties of the DFT operator in order to reduce the computational effort involved; and one of the most important properties used is the cyclic convolution theorem, which allows, through the DFT operator, for the replacement of cyclic convolution of sequences by point-wise sequence multiplication. In turn, fast algorithms have been developed for the computation of the discrete Fourier transform (DFT) of a sequence. This has lead to a further reduction in the computational effort required to perform cyclic convolution.

Another important point to make about LSI-FIR systems is that a number of algorithms are known which perform the processing of finite length sequences relatively fast when compared with straightforward methods of computation. The digital hardware implementation of these algorithms may be accomplished through the use of LSI-FIR systems. This makes LSI-FIR systems one of the most important tools in digital signal processing.

6.2. Mathematical Preliminaries:

In this section we introduce some mathematical notions which will help us describe some of the mathematical operators commonly used for the analysis of discrete linear systems with finite impulse response. These systems play a substantive role in the digital processing of discrete signals. Let Z/n denote the set of n nonnegative integers

$$\{0, 1, \ldots, n-1\}. \tag{1}$$

An n-point sequence over the complex field C is the mapping

$$f: Z/n \longrightarrow C \tag{2}$$

The set Z/n is called the indexing set of the sequence of f. The value of the sequence f on $j \in Z/n$ is $f(j)$ and it is usually denoted by f_j.

We denote by the symbol $\underline{f}$ the n-tuple formed with the values f_j, $j = 0, 1, 2, \ldots, n-1$:

$$\underline{f} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{bmatrix} \tag{3}$$

The set of all sequences $f: Z/n \rightarrow C$ forms a linear vector space which we denote by $L(Z/n)$. The set $L(Z/n)$ is isomorphic to the n-dimensional complex vector space C^n.

The set of n n-point sequences

$$\{\delta_{[k]}: k = 0, 1, \ldots, n-1\}, \tag{4}$$

where

$$\delta_{[k]}(j) = \begin{cases} 1, & j = k; \quad j, k \in Z/n \\ 0, & j \neq k \end{cases}, \tag{5}$$

forms a basis for the space $L(Z/n)$ which we call the standard basis.

We now introduce the shift operator S_n over the space $L(Z/n)$. This operator is the central component in the characterization of LSI-FIR systems.

Let the operator S_n over the space $L(Z/n)$ be defined in the following manner:

$$S_n: L(Z/n) \longrightarrow L(Z/n)$$

$$\delta_{[k]} \longmapsto S_n \delta_{[k]} = \delta_{[k+1]} \tag{6}$$

and $\quad S_n \delta_{[n-1]} = \delta_{[n]} = \delta_{[0]} \equiv \delta$

The sequence $\delta_{[0]} = \delta$ is called the unit sample sequence. If we replace the argument $j \in Z/n$ of this sequence by $j-1$, $j \in Z/n$, we obtain the following result

$$\delta_{[0]}(j-1) = \begin{cases} 1, & j-1=0; \quad j \in Z/n \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

This result of (7) is equivalent to the definition of the sequence $\delta_{[1]}$:

$$\delta_{[1]}(j) = \begin{cases} 1, & j=1; \quad j \in Z/n \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Thus, we write

$$\delta_{[1]}(j) = \delta_{[0]}(j-1), \quad j \in Z/n \tag{9}$$

or

$$(S_n\delta_{[0]})(j) = \delta_{[0]}(j-1), \quad j \in Z/n \tag{10}$$

As we stated above, the operator S_n is called the shift operator; and, it can be seen by (10) that shifting the sequence $\delta_{[0]}$ is equivalent to 'delaying' the sequence if its argument is thought to be discrete time instances. In general, we have

$$(S_n\delta_{[m]})(j) = \delta_{[m]}(j-1), \quad m \in Z/n \tag{11}$$

where parentheses are placed around the expression $S_n\delta_{[m]}$ in order to emphasize that it is a sequence obtained by operating by S_n on the sequence $\delta_{[m]}$. We rewrite (11) as

$$\delta_{[k]}(j) = \delta_{[k-1]}(j-1) \quad j,k \in Z/n \tag{12}$$

and notice that

$$\delta_{[k-1]}(j-1) = \delta_{[k-2]}(j-2) \tag{13}$$

We may continue this process and arrive at the following expressions for $\delta_{[k]}(j)$:

$$\delta_{[k]}(j) = \delta_{[0]}(j-k) = \delta(j-k) \quad j,k \in Z/n \tag{14}$$

and

$$\delta_{[k]}(j) = \delta_{[k-j]}(0) \quad j,k \in Z/n \tag{15}$$

The following identities follow from definition (6) of the shift operator

$$\begin{aligned}
\delta_{[1]} &= S_n\delta_{[0]} \\
\delta_{[2]} &= S_n\delta_{[1]} = S_n(S_n\delta_{[0]}) = S_n^2\delta_{[0]} \\
\delta_{[3]} &= S_n\delta_{[2]} = S_n^3\delta_{[0]} \\
&\vdots \\
\delta_{[k]} &= S_n\delta_{[k-1]} = S_n(S_n\ldots(S_n\delta_{[0]})\ldots) = S_n^k\delta_{[0]} \\
&\vdots \\
\delta_{[n]} &= S_n\delta_{[n-1]} = S_n^n\delta_{[0]}
\end{aligned} \tag{16}$$

Since $S_n\delta_{[n-1]} = \delta_{[0]}$ by definition, it follows that

$$S_n^n\delta_{[0]} = \delta_{[0]} = \delta \Longrightarrow S_n^n = I_n \tag{17}$$

where I_n is the identity operator. We can rewrite (14) and (15), respectively, as

$$\delta_{[k]}(j) = (S_n^k\delta_{[0]})(j) = \delta(j-k) \tag{18}$$

and

$$\delta_{[k]}(j) = S_n^{k-j}\delta_{[0]} = \delta(j-k) \tag{19}$$

Given an n-point sequence $f \in L(Z/n)$, we can write this sequence as a linear combination of the set of basis functions $\{\delta_{[k]},\ k \in Z/n\}$ in the following way

$$f = \sum_{0\le k<n} f_k\delta_{[k]} \tag{20}$$

where $f_k = f(k)$. We define the inner product $\langle\ ,\ \rangle$ of two sequences $f, g \in L(Z/n)$ as follows

$$\begin{aligned}
\langle\ ,\ \rangle &: L(Z/n) \times L(Z/n) \longrightarrow C \\
(f,g) &\longmapsto \langle f,g\rangle = \sum_{0\le k<n} f_k g_k^*
\end{aligned} \tag{21}$$

where $L(Z/n) \times L(Z/n)$ defines the cartesian product of the $L(Z/n)$ with itself, and $g_k^* = (g(k))^*$, the operation $(\ )^*$ implying complex conjugation. Using $\langle f, \delta_{[k]}\rangle = f_k$, we rewrite (20) as

$$f = \sum_{0\le k<n} \langle f, \delta_{[k]}\rangle\delta_{[k]} \tag{22}$$

Using the shift operator S_n, we can also write $f \in L(Z/n)$ as

$$f = \sum_{0\le k<n} f_k\delta_{[k]} = \sum_{0\le k<n} f_k S_n^k\delta_{[0]} \tag{23}$$

where

$$f(j) = \Big(\sum_{0\le k<n} f_k\delta_{[k]}\Big)(j) = \sum_{0\le k<n} f_k\delta_{[k]}(j) \tag{24}$$

$$f(j) = \sum_{0\le k<n} f_k(S_n^k\delta_{[0]})(j) = \sum_{0\le k<n} f_k\delta(j-k) = f_j \tag{25}$$

Operating by S_n on f results in

$$\begin{aligned}(S_nf)(j) &= \Big(S_n\sum_{k=0}^{n-1} f_kS_n^k\delta_{[0]}\Big)(j)\\ &= S_n\sum_{0\le k<n} f_kS_n^k\delta[0](j)\\ &= \sum_{0\le k<n} f_kS_n^k(S_n\delta_{[0]})(j)\\ &= \sum_{0\le k<n} f_kS_n^k\delta_{[0]}(j-1) = f(j-1)\end{aligned} \tag{26}$$

By iterating the procedure stated above, the following result follows

$$(S_n^kf)(j) = \sum_{0\le m<n} f_mS_n^m(S^k\delta_{[0]})(j) = \sum_{0\le <n} f_mS_n^m\delta_{[k]}(j) \tag{27}$$

$$\sum_{0\le m<n} f_mS_n^m\delta_{[k]}(j) = \sum_{0\le m<m} f_mS_n^m\delta(j-k) = f(j-k) \tag{28}$$

In general, we will denote by $f_{[k]}$ the function obtained by applying the operator S_n^k to the function f:

$$f_{[k]} = S_n^kf = \sum_{j\in Z/n} f_jS_n^k\delta_{[j]}, \quad f\in L(Z/n) \tag{29}$$

Using the linearity property of the shift operator S_n, the following evaluation is determined

$$(S_n(\alpha f+\beta g))(k) = (\alpha Sf+\beta Sg)(k), \quad f,g\in L(Z/n), \quad \alpha,\beta\in C \tag{30}$$

$$\begin{aligned}(S_n(\alpha f+\beta g))(k) &= (\alpha f_{[1]}+\beta g_{[1]})(k)\\ &= \alpha f_{[1]}(k)+\beta g_{[1]}(k)\\ &= \alpha(S_nf)(k)+\beta(S_ng)(k)\end{aligned} \tag{31}$$

In the next section we will describe a class of operators which commute with the shift operator. These operators are an essential tool in the analysis of discrete (digital) signals.

6.3. Properties LSI-FIR Systems:

In this section we would like to describe a special class of operators which are linear and, most important, shift invariant; i.e., they commute with the shift operator S_n. These operators play a most important role in the field of digital signal processing (DSP). They are usually referred as linear shift invariant (LSI) systems. The reason for this name is now explained. As it was stated in the introduction, in order to extract what may be considered useful information from a digital signal (numeric sequence), this digital signal is usually acted on by a linear process. The result (output) of a linear process is usually another digital signal. The process itself may be described as a finite sequence of operations performed on the input signal. If we adopt the black box concept, then we can describe this scenario as a black box (the linear process) with an input (the input signal) and a response (the output signal). Physical implementation of these black boxes are called systems; whereby, in its simplest form, a system may consist of a hardware device performing a single linear operation, usually shift invariant.

By a process of analogy, the mathematical model of a physical linear process may be described as a finite composition of linear operators which acts on a given signal known as the input signal to the process. A linear process, in its simplest form, may consist of a single linear (usually shift invariant) operator. If the output (response) of an LSI system is finite; i. e., the output digital signal is finite, then we further classify these systems as finite impulse response (FIR) systems. Our objective in this section is to make this classification precise.

LSI-FIR systems are very important in the analysis of digital signals. This stems from the fact that if the response of a given system T, acting on the basis function δ, $\delta \in L(Z/n)$ is known, then the result of the system acting on any other input signal $f \in L(Z/n)$ may be deduced. We make this statement more precise. Let $h_{[0]} = h$ be the sequence obtained by applying a given LSI-FIR system T to the basis function δ:

$$T\delta_{[0]} = T\delta = h_{[0]} = h \tag{1}$$

The result of T acting on any other basis function $\delta_{[k]} \in L(Z/n)$, $k \in Z/n$ is given by

$$T(\delta_{[k]}) = T(S_n^k \delta) = S_n^k(T\delta) = S_n^k h \tag{2}$$

Since any given sequence $f \in L(Z/n)$ can be written uniquely as

$$f = \sum_{k \in Z/n} f(k)\delta_{[k]}, \quad f \in L(Z/n), \tag{3}$$

we have that

$$\begin{aligned} T(f) &= T\Big(\sum_{k \in Z/n} f(k)\delta_{[k]}\Big) \\ &= \sum_{k \in Z/n} f(k)T(\delta_{[k]}) \\ &= \sum_{k \in Z/n} f(k)TS_n^k\delta \\ &= \sum_{k \in Z/n} f(k)S_n^kT\delta \\ &= \sum_{k \in Z/n} f(k)S_n^kh \end{aligned} \tag{4}$$

The sequence h is usually termed the signal impulse response of the acting system, in this case, the system T.

We now set out to define a linear shift invariant, finite impulse response (LSI-FIR) system. Let $L(Z/a), L(Z/b)$ be subspaces of $L(Z/c)$ where a, b, c are positive integers and $c = a + b - 1$. An LSI-FIR system T_h is a mapping

$$\begin{aligned} T_h &: L(Z/a) \times L(Z/b) \longrightarrow L(Z/c) \\ &(f, h) \longmapsto T_h(f) \end{aligned} \tag{5}$$

where

$$T_h = \sum_{0 \le k < c} h(k)S_c^k \tag{6}$$

and

$$\begin{aligned} (T_h(f))(j) &= \sum_{0 \le k < c} h(k)(S_c^kf)(j) \\ &= \sum_{0 \le k < c} h_kf_{j-k} \end{aligned} \tag{7}$$

We sometimes write $T_h(f) = f * h$.

To show the shift invariance and linearity properties of the system T_h we proceed as follows:

For $f, g \in L(Z/a)$, $h \in L(Z/b)$, $\alpha, \beta \in C$, we have

$$\begin{aligned} T_h(\alpha f + \beta g) &= \sum_{0 \le k < c} h(k)S_c^k(\alpha f + \beta g) \\ &= \sum_{0 \le k < c} h(k)S_c^k(\alpha f) + \sum_{0 \le k < c} h(k)S_c^k(\beta g) \\ &= \alpha T_h(f) + \beta T_h(g) \end{aligned} \tag{8}$$

$$
\begin{aligned}
S_c T_h(f) &= S_c \sum_{0\le k<c} h(k)(S_c^k f) \\
&= \sum_{0\le k<c} h(k) S_c^k (Sf) \\
&= T_h S_c(f)
\end{aligned} \tag{9}
$$

By iterating the procedure given above, the following identity is obtained

$$
S_c^m T_h = T_h S_c^m \tag{10}
$$

We proceed to describe some more properties of LSI-FIR systems. We start by setting

$$
T_h(\delta) = \sum_{0\le k<c} h(k) S_c^k \delta = \sum_{0\le k<c} h(k)\delta_{[k]} = h, \quad \delta \in L(Z/a), \quad h \in L(z/b) \tag{11}
$$

Let $T_h, T_f \in L(Z/c)$ be any two LSI-FIR systems:

$$
T_h(f) = \sum_{0\le k<c} h(k) S_c^k f, \quad f \in L(Z/a), \quad h \in L(Z/b) \tag{12}
$$

$$
T_f(h) = \sum_{0\le k<c} f(k) S_c^k h, \quad h \in L(Z/b), \quad f \in L(Z/a) \tag{13}
$$

Noticing that $f = \sum_{0\le m<c} f(m) S_c^m \delta$, we write

$$
T_h(f) = \sum_{0\le k<c} h(k) S_c^k \Big(\sum_{0\le m<c} f(m) S_c^m \delta \Big) \tag{14}
$$

Using the linearity property of $L(Z/c)$, we write

$$
T_h(f) = \sum_{0\le m<c} \sum_{0\le k<c} f(m) S_c^m h(k) S_c^k \delta \tag{15}
$$

$$
T_h(f) = \sum_{0\le m<c} f(m) S_c^m \Big(\sum_{0\le k<c} h(k) S_c^k \delta \Big) = T_f(h) \tag{16}
$$

Thus, we arrive at the following commutation identity

$$
T_h(f) = f * h = h * f = T_f(h) \tag{17}
$$

For any basis sequence $\delta_{[k]} \in L(Z/a)$, we write

$$
\begin{aligned}
T_h(\delta_{[k]}) &= \sum_{0\le m<c} h(m) S_c^m \delta_{[k]} \\
&= S_c^k \sum_{0\le m<c} h(m) S_c^m \delta = S_c^k h
\end{aligned} \tag{18}
$$

hence,

$$T_h(\delta_{[k]}) = T_{[k]}(h) = \delta_{[k]} * h = S_c^k h \tag{19}$$

We thus can make the following correspondence

$$T_{\delta_{[k]}} \longleftrightarrow S_c^k \tag{20}$$

We also notice that

$$T_h(\delta_{[k]})(j) = (h * \delta_{[k]})(j) = h(j-k) \tag{21}$$

In order to characterize LSI-FIR systems, we start by identifying the sequence obtained by letting the system T_h act on the unit sample sequence δ. Since any n-th order sequence f can be written as a linear combination of shifted versions of δ, knowing the response $T_h(\delta)$ will help in determining $T_h(f)$. We call the unit sample response or impulse response of the system T_h the result obtained by applying T_h to the unit sample sequence δ, which sometimes is called the impulse function:

$$\begin{aligned} T_h(\delta) &= \sum_{0 \le m < c} h(m) S_c^m \delta_{[0]} \\ &= \sum_{0 \le m < c} h(m) \delta_{[m]} = h \end{aligned} \tag{22}$$

Thus, the unit sample response an LSI-FIR system T_h is the sequence h. For any given sequence $f \in L)Z/a)$, we can always write

$$f = \sum_{k \in Z/c} f(k) S_c^k \delta = \sum_{k \in Z/c} f(k) \delta_{[k]} \tag{23}$$

Evaluating f at $j \in Z/c$ results in

$$f(j) = \sum_{k \in Z/c} f(k) \delta_{[k]}(j) = \sum_{k \in Z/c} f(k) \delta(j-k) \tag{24}$$

Applying T_h to the sequence f (or, inputing the sequence f to the system T_h) results in

$$T_h(f) = \sum_{m \in Z/c} h(m) S_c^m \Big(\sum_{k \in Z/c} f(k) \delta_{[k]} \Big) \tag{25}$$

Using the linearity property of $L(Z/c)$, we obtain

$$\begin{aligned} T_h(f) &= \sum_{k \in Z/c} f(k) \sum_{m \in Z/c} h(m) S_c^m \delta_{[k]} \\ &= \sum_{k \in Z/c} f(k) S_c^k \Big(\sum_{m \in Z/c} h(m) S_c^m \delta \Big) \\ &= \sum_{k \in Z/c} f(k) S_c^k h \end{aligned} \tag{26}$$

This result states that the system T_h is completely characterized by its unit sample response sequence h.

6.4. The Discrete Fourier Operator:

We would like to make a transition from the concept of linear mappings $L(Z/a) \times L(Z/b) \rightarrow L(Z/c)$ involving three spaces to linear transformations $L(Z/n) \rightarrow L(Z/n)$. It is important to point out that if we make the identification $n = c = a+b-1$ the concept of LSI-FIR systems can be incorporated into the concept linear transformation on $L(Z/n)$. This may be accomplished by identifying subsets in $L(Z/n)$ isomorphic to $L(Z/a)$ and $L(Z/n)$ respectively. Then sequences in $L(Z/a)$ and in $L(Z/b)$ will be represented in $L(Z/n)$ as n-point sequences which will have the value zero outside their respective domain of definition (Z/a and Z/b respectively).

The space $L(Z/n)$ will be used to represent two related ideas. In the first place, $L(Z/n)$ will be thought of as representing the space of all finite n-point complex sequences with domain Z/n. On the other hand, the space $L(Z/n)$ will also be thought of as representing the space of all periodic complex sequences with period n. This latter interpretation will allow us to perform modulo n operations on the indexing set Z/n, turning this set to into an additive group of order n. Evaluations modulo n will become more clear as we delve into the properties of special operators defined on $L(Z/n)$. One of these operators is the ubiquitous discrete Fourier transform (DFT) operator. We proceed to describe this operator below. The discrete Fourier transform (DFT) of an n-point sequence f is defined in this section as a linear operator on the space $L(Z/n)$. However, before we introduce this definition, we would like to describe some preliminary concepts and definitions.

The indexing set $A = Z/n = \{0, 1, \ldots, n-1\}$ forms an abelian group with modulo n addition as the internal binary operation. Its dual $\hat{A}$ is defined as

$$\hat{A} = \{{}^n\chi_k : k \in Z/n\} \equiv \widehat{(Z/n)} \tag{1}$$

where

$$\begin{aligned} {}^n\chi_k &: Z/n \longrightarrow C \\ (j) &\longmapsto {}^n\chi_k(j) = e^{-2\pi i k \cdot (j)/n} \end{aligned} \tag{2}$$

When no ambiguities arise, we drop the superscript n from the expression ${}^n\chi_k$. The value ${}^n\chi_1(1)$ is usually written as $w_n = e^{-2\pi i/n}$. The functions χ_k are usually termed exponential sequences, characteristic sequences, or, simply, characters.

The set of functions $\hat{A}$ plays a very important role in the analysis of LSI-FIR systems. We will show that they are eigenfunctions of this type of systems. We first

show that they form an orthogonal set, spanning an n-dimensional space which we denote by $L(\hat{A}) = L((Z/n)\hat{\ })$. This is done by looking at the inner product of any two sequences in $\hat{A}$. This product is given by

$$\langle \chi_a, \chi_b \rangle, \quad a, b \in Z/n \tag{3}$$

where

$$\begin{aligned} \langle \chi_a, \chi_b \rangle &= \sum_{j \in Z/n} e^{-2\pi i j a/n} e^{+2\pi i j b/n} \\ &= \sum_{j \in Z/n} e^{-2\pi i j (a-b)/n} = 0, \quad a \neq b \end{aligned} \tag{4}$$

Thus,

$$\langle \chi_a, \chi_b \rangle = \begin{cases} 0, & a \neq b \\ n, & a = b \end{cases} \tag{5}$$

We write

$$\langle \chi_a, \chi_b \rangle = n\delta(a-b) = nS_n^b \delta_{[0]}(a) = n\delta_{[b]}(a), \quad a, b \in Z/n \tag{6}$$

A linear mapping $\tilde{F}$ may be defined between the spaces $L(Z/n) = L(A)$ and $L(\hat{A})$ as follows

$$\begin{aligned} \tilde{F}: L(A) &\longrightarrow L(\hat{A}) \\ (f) &\longmapsto \tilde{F}(f) \equiv \tilde{f} \end{aligned} \tag{7}$$

where

$$\tilde{F}(f) = \tilde{f} = \sum_{j \in Z/n} f_j \chi_j \tag{8}$$

The sequence $\tilde{f}$ can also be written as

$$\tilde{f} = \frac{1}{n} \sum_{k \in Z/n} \langle \tilde{F}(f), \chi_k \rangle \chi_k \tag{9}$$

where

$$\begin{aligned} \langle \tilde{F}(f), \chi_k \rangle &= \sum_{m \in Z/n} \Big(\sum_{j \in Z/n} f_j \chi_j \Big)(m) \chi_k^*(m) \\ &= \sum_{m \in Z/n} \Big(\sum_{j \in Z/n} f_j \chi_j(m) \Big) \cdot \chi_k^*(m) \\ &= \sum_{j \in Z/n} f_j \sum_{m \in Z/n} \chi_j(m) \chi_k^*(m) \\ &= \sum_{j \in Z/n} f_j \langle \chi_j, \chi_k \rangle \\ &= \sum_{j \in Z/n} f_j (n\delta(j-k)) = n f_k \end{aligned} \tag{10}$$

Evaluating $\tilde{F}(f)$ at $k \in Z/n$ produces

$$\begin{aligned}(\tilde{F}(f))(k) &= (\sum_{j\in Z/n} f_j\chi_j)(k) \\ &= \sum_{j\in Z/n} f_j\chi_j(k), \quad k \in Z/n\end{aligned} \tag{11}$$

Noticing that

$$\chi_j(k) = e^{-2\pi ijk/n} = w_n^{jk} = \chi_k(j) = w_n^{kj}, \tag{12}$$

we write $(\tilde{F}(f)(k) = \tilde{f}(k)$ as

$$\tilde{f}(k) = \sum_{j\in Z/n} f_j\chi_k(j) = \langle f, \chi_k^* \rangle \tag{13}$$

Since the set $\{\chi_k^* : k \in Z/n\}$ is an orthogonal basis, we can write f as

$$f = \frac{1}{n}\sum_{k\in Z/n} \langle f, \chi_k^* \rangle \chi_k^* \tag{14}$$

Thus, we can represent an n-point sequence $f \in L(Z/n)$ in two ways:

$$f = \sum_{j\in Z/n} \langle f, \delta_{[j]} \rangle \delta_{[j]} = \sum_{j\in Z/n} f_j\delta_{[j]} \tag{15}$$

$$f = \frac{1}{n}\sum_{k\in Z/n} \langle f, \chi_k^* \rangle \chi_k^* = \frac{1}{n}\sum_{k\in Z/n} \tilde{f}_k\chi_k^* \tag{16}$$

In this setting we can think of the vector sequence f as being represented with respect to two different sets of bases. We proceed to establish a relationship between these two bases. We start with the identity

$$\sum_{j\in Z/n} f_j\delta_{[j]} = \frac{1}{n}\sum_{k\in Z/n} \tilde{f}_k\chi_k^* = f \tag{17}$$

Evaluating $\langle f, \delta_{[m]} \rangle$ results in

$$\begin{aligned}\langle (\sum_{j\in Z/n} f_j\delta_{[j]}), \delta_{[m]} \rangle &= \frac{1}{n}\langle (\sum_{k\in Z/n} \tilde{f}_k\chi_k^*), \delta_{[m]} \rangle = f_m \\ &= \frac{1}{n}\sum_{s\in Z/n} (\sum_{k\in Z/n} \tilde{f}\chi_k^*)(s)\delta_{[m]}(s) \\ &= \frac{1}{n}\sum_{k\in Z/n} \tilde{f} \sum_{s\in Z/n} \chi_k^*(s)\delta_{[m]}(s) \\ &= \frac{1}{n}\sum_{k\in Z/n} \tilde{f}_k\chi_k^*(m) = \sum_{k\in Z/n} \tilde{f}_k\chi_m^*(k) \\ &= \langle f, \delta_{[m]} \rangle = \frac{1}{n}\langle \tilde{f}, \chi_m \rangle\end{aligned} \tag{18}$$

Starting with the identity

$$\frac{1}{n}\sum_{k\in Z/n}\tilde{f}_k\chi_k^* = \sum_{j\in Z/n} f_j\delta_{[j]} = f \tag{19}$$

and evaluating $\langle f,\chi_k^*\rangle$ produces

$$\begin{aligned}\frac{1}{n}\langle(\sum_{k\in Z/n}\tilde{f}_k\chi_k^*),\chi_m^*\rangle &= \langle(\sum_{j\in Z/n} f_j\delta_{[j]}),\chi_m^*\rangle = \tilde{f}_m \\ &= \sum_{s\in Z/n}(\sum_{j\in Z/n} f_j\delta_{[j]})(s)\chi_m(s) \\ &= \sum_{j\in Z/n} f_j\sum_{s\in Z/n}\chi_m(s)\delta_{[j]}(s) \\ &= \sum_{j\in Z/n} f_j\chi_m(j) = \langle f,\chi_m^*\rangle\end{aligned} \tag{20}$$

We now relate the vector coordinate representations f_j, $j\in Z/n$ and $\tilde{f}_k$, $k\in Z/n$. We accomplish this by making a correspondence between the mapping $\tilde{F}$ and a linear transformation F_n defined on the space $L(Z/n)$. This new linear operator, as we shall demonstrate, performs a change of basis in $L(Z/n)$. We start identifying this change of basis by making the following definition

$$\begin{aligned}F_n: L(Z/n) &\longrightarrow L(Z/n) \\ \delta_{[j]} &\longmapsto F_n\delta_{[j]}\end{aligned} \tag{21}$$

where

$$F_n\delta_{[j]} = \chi_j, \quad j\in Z/n \tag{22}$$

Allowing F_n to operate on f gives

$$F_nf \equiv \hat{f} = F_n(\sum_{j\in Z/n} f_j\delta_{[j]}) = \sum_{j\in Z/n} f_jF_n\delta_{[j]} = \sum_{j\in Z/n} f_j\chi_j \tag{23}$$

Since F_n is an operator defined on a finite dimensional vector space, it has a matrix representation. We shall denote by

$$[w_n^{jk}]_{0\le j,k<n} \equiv F_n \tag{24}$$

the matrix of F_n with respect to the basis set $\{\delta_{[j]}: j\in Z/n\}$. We, thus, write

$$\chi_k = F_n\delta_{[k]} = \sum_{j\in Z/n} w_n^{jk}\delta_{[j]} \tag{25}$$

Since $f = \frac{1}{n}\sum_{k\in Z/n} \hat{f}_k \chi_k^*$, we write

$$f = \frac{1}{n}\sum_{k\in Z/n} \hat{f}_k \chi_k^* = \frac{1}{n}\sum_{k\in Z/n} \hat{f}_k (F_n \delta_{[k]})^* = \frac{1}{n}\sum_{k\in Z/n} \hat{f}_k \Big(\sum_{j\in Z/n} w_n^{jk}\delta_{[j]}\Big)^*$$
$$= \frac{1}{n}\sum_{k\in Z/n} \hat{f}_k \sum_{j\in Z/n} (w_n^{jk})^* \delta_{[j]} = \sum_{j\in Z/n} \Big\{\frac{1}{n}\sum_{k\in Z/n} (w_n^{jk})^* \hat{f}_k\Big\}\delta_{[j]} \qquad (26)$$
$$= \sum_{j\in Z/n} f_j \delta_{[j]}$$

Thus, the expression within curly braces represents

$$f_j = \frac{1}{n}\sum_{k\in Z/n} (w_n^{jk})^* \hat{f}_k = \frac{1}{n}\sum_{k\in Z/n} \chi_k^*(j)\hat{f}_k \qquad (27)$$

and

$$\hat{f}_k = \sum_{j\in Z/n} \chi_j(k) f_j = \sum_{j\in Z/n} (w_n^{jk}) f_j \qquad (28)$$

The n-point sequence χ_k can also be expressed as

$$\chi_k = F_n \delta_{[k]} = \sum_{j\in Z/n} \langle F_n \delta_{[k]}, \delta_{[j]}\rangle \delta_{[j]} \qquad (29)$$

Comparing this expression (29) with the expression given by (25), results in the following equality

$$w_n^{jk} = \langle F_n \delta_{[k]}\rangle \qquad (30)$$

It is important to observe here that we have been using the same symbol to denote both, an n-point sequence, say χ_k, as well as its vector coordinate representation. For instance, the vector χ_k is the vector

$$\chi_k = [\chi_k(0), \chi_k(1), \chi_k(2), \ldots, \chi_k(n-1)]^T \qquad (31)$$

where the entries represent the vector coordinates of the sequence χ_k with respect to a given basis set. Remembering from expressing (5) that, for $j, k \in Z/n$,

$$\langle \chi_j, \chi_k\rangle = \sum_{m\in Z/n} \chi_j \chi_k^* = \sum_{m\in Z/n} \chi_m(j)\chi_m^*(k) = n\delta(j-k), \qquad (31)$$

we write

$$\sum_{m\in Z/n} w_n^{mj}\Big(\frac{1}{n}w_n^{mk}\Big)^* = \sum_{m\in Z/n} w_n^{jm}\Big(\frac{1}{n}w_n^{km}\Big)^* = \delta(j-k) \qquad (32)$$

Thus, from the above relations and the fact that F_n is symmetric (see expression (12)), we obtain the following identity

$$F_n^{-1} = \frac{1}{n}F_n^* = \frac{1}{n}(F_n^T)^* \qquad (33)$$

This identity reiterates the fact that the operator F_n is a unitary operator whose action on $L(Z/n)$ can be thought of as a rotation of coordinate axes.

We now relate the concepts of LSI-FIR systems and the F_n operator. Given an n-point impulse response sequence $h \in L(Z/n)$ and an input sequence $x = \chi_k$, the output y of the LSI-FIR filter T_h becomes

$$y = h * \chi_k^* = \sum_{j \in Z/n} h(j) S_n^j \chi_k^* = T_h(\chi_k^*) \tag{34}$$

Evaluating y at $m \in Z/n$ results in

$$\begin{aligned} y(m) &= (h * \chi_k^*)(m) = \sum_{j \in Z/n} h(j)(S_n^j \chi_k^*)(m) = \sum_{j \in Z/n} h_j S_n^j \chi_k^*(m) \\ &= \sum_{j \in Z/n} h_j \chi_k^*(m-j) = \sum_{j \in Z/n} h_j e^{+2\pi i k(m-j)} = \sum_{j \in Z/n} e^{+2\pi i k m/n} e^{-2\pi i k j/n} \\ &= \chi_k^*(m) \sum_{j \in Z/n} h_j \chi_k(j) = \chi_k^*(m) \sum_{k \in Z/n} h_j \chi_j(k) = \chi_k^*(m) \hat{h}_k \end{aligned} \tag{35}$$

Rewriting the above result in operator notation, results in the following identity

$$T_h(\chi_k^*) = (F_n h)(k) \chi_k^* \tag{36}$$

This equality (36) states that the exponential sequences χ_k^*, $k \in Z/n$ are eigenfunctions of the LSI-FIR systems. Specifically, for any given system T_h, the sequence χ_k, $k \in Z/n$ is an eigenfunction, and $(F_n h)(k)$ is the associated eigenvalue.

6.5. Cyclic Convolution:

So far we have seen an input sequence $f \in L(Z/a)$ been mapped by an LSI-FIR system T_h into an output sequence $T_h(f) \in L(Z/c)$. This result was also written as $T_h(f) = f * h$, $h \in L(Z/b)$. This system operation is usually termed linear convolution; i.e., the input sequence $f \in L(Z/a)$ is convolved with the unit sample response sequence $h \in L(Z/b)$. We recall that both $L(Z/a)$ and $L(Z/b)$ are subspaces of $L(Z/c)$

There is another type of system convolution called cyclic convolution. This operation is used when the spaces $L(Z/a), L(Z/b), L(Z/c)$ are defined to be of the same dimension, namely, $L(Z/a) = L(Z/b) = L(Z/c) = L(Z/n)$. Cyclic convolution is defined in the following way

$$\begin{aligned} *: L(Z/n) \times L(Z/n) &\longrightarrow L(Z/n) \\ (f, h) &\longmapsto f * h = g \end{aligned} \tag{1}$$

where

$$g = \sum_{j \in Z/n} h_j S_n^j f \tag{2}$$

Evaluating h at a particular value $k \in Z/n$ results in

$$T(f,h)(k) = h(k) = \sum_{j \in Z/n} h(j)(S_n^j f)(k) = \sum_{j \in Z/n} h_j f(k-j) \tag{3}$$

This result resembles the result of the linear convolution operation, with the major difference being that the index evaluations are performed modulo n. In this setting, cyclic convolution can be thought of as a bilinear operation acting on the space $L(Z/n)$. We would like to point out that we have used the symbol $*$ to denote cyclic convolution, which was the same one previously used for linear convolution. The reason for doing this is that throughout the rest of this work we will concentrate mostly on the use of cyclic convolution as opposed to linear convolution. In addition, we want to keep operation notations down to a minimum.

Since cyclic convolution is a modulo n operation, it can be viewed as a linear operator acting on the space of signals $f \in L(Z/n)$:

$$\begin{aligned} T_h: L(Z/n) &\longrightarrow L(Z/n) \\ (f) &\longmapsto T_h(f) \end{aligned} \tag{4}$$

where

$$T_h(f) = \sum_{j \in Z/n} h_j S_n^j f \equiv (f) * h \tag{5}$$

This operator T_h can be thought of as representing an LSI-FIR system whose impulse response sequence (function) is h. In this way, as stated in §6.3, the system T_h is uniquely characterized by the signal h

A system T_h acting on the unit sample sequence shifted by j units produces the following response

$$\begin{aligned} T_h(\delta_{[j]})(k) &= (\delta_{[j]} * h)(k) \equiv h_{[j]} \\ &= \sum_{m \in Z/n} h_m S_n^m \delta_{[j]}(k) = \sum_{m \in Z/n} h_m \delta(k-j-m) \\ &= \sum_{m \in Z/n} h_m \delta_{[j-k]}(-m) = h(k-j) = S_n^j h(k) \end{aligned} \tag{6}$$

Thus, we conclude

$$T_h(S_n^j \delta[0]) = T_h(S_n^j \delta) = S_n^j h \tag{7}$$

It is important to remember that the response of the system T_h to the unit sample signal δ is, essentially, $T_h(\delta) = h$. The result stated above in (7) reiterates the shift invariance property of the system T_h.

If the system T_h takes as its input a shifted version of any given signal $f \in L(Z/n)$, it produces the following response

$$T_h(S_n^k f) = (S_n^k f) * h = \sum_{j \in Z/n} h_j S_n^j (S_n^k f) = \sum_{j \in Z/n} h_j S_n^{j+k} f \tag{8}$$

Setting $m = j + k$ results in

$$T_h(S_n^k f) = \sum_{m=k}^{n+k-1} h_{m-k} S_n^m f = \sum_{j \in Z/n} h_{j-k} S_n^j f = f * (S_n^k h) \tag{9}$$

We also noticed that

$$T_h(S_n^k f) = \sum_{j \in Z/n} h_j S_n^{j+k} f = S_n^k \Big(\sum_{j \in Z/n} h_j S_n^j f \Big) \tag{10}$$

Thus, we conclude

$$T_h(S_n^k f) = T_{(S_n^k h)}(f) = S_n^k (T_h(f)) \tag{11}$$

The result stated above in (11) implies

$$T_{(S_n^k h)}(S_n^j) = S_n^j (T_{(S_n^k h)}(f) = S_n^{j+k} T(T_h(f)) \tag{12}$$

We have shown previously that a given system T_h, being an LSI-FIR system, commutes with any composition S_n^j of the shift operator S_n; that is,

$$S_n^j T_h = T_h S_n^j, \quad j \in Z/n \tag{13}$$

We would like to show that any two LSI-FIR systems T_u, T_v commute; and furthermore, we would like to show that their composition $T_u \circ T_v$ results in another LSI-FIR system. We proceed as follows

The product $T_u \circ T_v$ of any two LSI-FIR systems acting on a given signal $f \in L(Z/n)$ results in

$$T_u(T_v(f)) = T_u\Big(\sum_{j\in Z/n} v_j S_n^j f\Big) \tag{14}$$

$$\begin{aligned} T_u\Big(\sum_{j\in Z/n} v_j S_n^j f\Big) &= \sum_{k\in Z/n} u_k S_n^k\Big(\sum_{j\in Z/n} v_j S_n^j f\Big) \\ &= \sum_{k\in Z/n} u_k\Big(\sum_{j\in Z/n} v_j S_n^{j+k} f\Big) \\ &= \sum_{k\in Z/n}\sum_{j\in Z/n} v_j u_k S_n^{j+k} f \end{aligned} \tag{15}$$

Continuing

$$T_u(T_v(f)) = \sum_{j\in Z/n} v_j S_n^j\Big(\sum_{k\in Z/n} u_k S_n^k f\Big), \quad f \in L(Z/n) \tag{16}$$

We conclude,

$$T_u \circ T_v = T_v T_u, \quad u, v \in L(Z/n) \tag{17}$$

We now proceed to rearrange the terms given in (17) in order to obtain a different result

$$\begin{aligned} T_u(T_v(f)) &= \sum_{k\in Z/n}\sum_{j\in Z/n} v_j u_k S_n^{j+k} f \\ &= \sum_{k\in Z/n}\Big(\sum_{j\in Z/n} v_j S_n^j u_k\Big) S_n^k f \\ &= \sum_{k\in Z/n}\Big(\sum_{j\in Z/n} v_j (S_n^j u)_k\Big) S_n^k f \\ &= \sum_{k\in Z/n} (T_v(u))_k S_n^k f \end{aligned} \tag{18}$$

Thus, we write

$$T_u(T_v(f)) = T_{(T_v(u))}(f) = T_{u*v}(f) = f * u * v \tag{19}$$

Looking closely at the action $T_h(f)$, $f \in L(Z/n)$, we notice that

$$\begin{aligned} T_h(f)(k) \sum_{j\in Z/n} h_j(S_n^j f)_k &= \sum_{j\in Z/n} h_j f(k-j) = \sum_{m=k}^{k-n+1} h(k-m) f_m \\ &= \sum_{j\in Z/n} f_j(S_n^j h)_k = T_f(h)(k) \end{aligned} \tag{20}$$

Combining this result with the previously stated result allows us to write the following expression

$$T_u(T_v(f)) = T_{u*v}(f) = T_{f*v}(v) = T_{v*f}(u) \tag{21}$$

We would like to emphasize the fact that the set of n-dimensional LSI-FIR systems forms a linear space:

$$\begin{aligned} T_{(\alpha u+\beta v)}(f) &= \sum_{j\in Z/n} (\alpha u+\beta v)_j S_n^j f, \quad \alpha,\beta \in Z/n, \quad u,v,f \in L(Z/n) \\ &= \sum_{j\in Z/n} (\alpha u_j + \beta v_j) S_n^j f \\ &= \sum_{j\in Z/n} \alpha u_j S_n^j f + \sum_{j\in Z/n} \beta v_j S_n^j f \\ &= \alpha T_u(f) + \beta T_v(f), \quad f \in L(Z/n) \end{aligned} \tag{22}$$

Thus, we write

$$T_{(\alpha u+\beta v)} = \alpha T_u + \beta T_v \tag{23}$$

We now combine the linearity property of LSI-FIR system with the cyclic convolution property of these systems. Take, for instance, the systems T_x, T_y, T_z. We evaluate the expression $T_{x*(y+z)}$ at a particular signal $f \in L(Z/n)$:

$$\begin{aligned} T_{x*(y+z)}(f) &= \sum_{j\in Z/n} (x*(y+z))_j S_n^j f \\ &= \sum_{j\in Z/n} (T_{y+z}(x))_j S_n^j f \\ &= \sum_{j\in Z/n} (T_y(x) + T_z(x))_j S_n^j f \\ &= \sum_{j\in Z/n} (T_y(x))_j S_n^j f + \sum_{j\in Z/n} (T_z(x)) \\ &= \sum_{j\in Z/n} (x*y)_j S_n^j f + \sum j \in Z/n (x*z)_j S_n^j f \\ &= T_{x*y}(f) + T_{x*z} = T_{x*y+x*z}(f) \end{aligned} \tag{24}$$

Combining some of the results obtained above allows us to write the following

$$T_x(T_y + T_z) = T_x T_{y+z} = T_{x*(y+z)} = T_{x*y+x*z} = T_x T_y + T_x T_z \tag{25}$$

6.6. Properties of the DFT operator:

This section introduces additional properties of the discrete Fourier transform operator; in particular, the section describes relationships between the DFT operator and the shift operator. It also describes DFT properties relating to exponential sequences $\chi_j : j \in Z/n$; and it states the cyclic correlation and cyclic convolution theorems. We start by giving the following definition of the Hadamard product of two signals $f, h \in L(Z/n)$:

We define the Hadamard product of $\odot$ of two n-point sequences $f, h \in L(Z/n)$ as follows

$$\begin{aligned} \odot : L(Z/n) \times L(Z/n) &\longrightarrow L(Z/n) \\ (f, h) &\longmapsto (f \odot h) \end{aligned} \tag{1}$$

where

$$(f \odot h)(k) = f_k \cdot h_k, \quad k \in L(Z/n) \tag{2}$$

Recalling Eq. (2) given in §6.4, we rewrite the Fourier transform of a standard basis function $\delta_{[j]}$, $j \in Z/n$:

$$F_n \delta_{[j]} = F_n(S_n^j \delta) = \chi_j, \quad j \in Z/n \tag{3}$$

For the particular case of $j = 0$, expression (3) becomes

$$F_n \delta = \chi_0 = u = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{4}$$

where u is called the unit sequence.

To obtain the Fourier transform of the shifted sequence $f_{[j]}$, we proceed as follows:

$$(F(f_{[j]})) = (F_n(S_n^j f)) = \sum_{m \in Z/n} (f_{[j]})(m) \chi_m = \sum_{m=0}^{n-1} f(m-j) e^{-2\pi i m k/n} \tag{5}$$

Evaluating at a particular value $k \in Z/n$ results in

$$(F_n(S_n^j f))(k) = \sum_{m \in Z/n} (f_{[j]})(m) \chi_m(k) = \sum_{m=0}^{n-1} f(m-j) e^{-2\pi i m k/n} \tag{6}$$

Allowing $a = m - j$ results in

$$
\begin{aligned}
&(F_n(S_n^j f))(k) = (F_n(f_{[j]}))(k) \\
&= \sum_{a=-j}^{n-1-j} f_a e^{-2\pi i(a+j)k/n} = \chi_j(k) \sum_{a=-j}^{n-1-j} f_a \chi_a(k) \\
&= \chi_j(k)\big(f_{-j+1}\chi_{-j+1}(k) + f_{-j+2}\chi_{j+2}(k) + \ldots + f_{-1}\chi_{-1} + f_0\chi_0(k) \\
&+ f_1\chi_1(k) + f_2\chi_2(k) + \ldots + f_{n-2-j}\chi_{n-2-j}(k) + f_{n-1-j}\chi_{n-1-j}(k)\big) \qquad (7) \\
&= \chi_j(k)\big(f_0\chi_0(k) + f_1\chi_1(k) + f_2\chi_2(k) + \ldots + f_{n-2-j}\chi_{n-2-j}(k) \\
&+ f_{n-1-j}\chi_{n-1-j}(k) + \ldots + f_{n-j+1}\chi_{n-j+1} + f_{n-j+2}\chi_{n-j+2} + \ldots + f_{n-1}\chi_{n-1}(k)\big) \\
&= \chi_j(k) \sum_{m=0}^{n-1} f_m \chi_m(k) = \chi_j(k)\hat{f}_k
\end{aligned}
$$

Since $k \in Z/n$ was chosen arbitrarily, we conclude

$$F_n(f_{[j]}) = F_n(S_n^j f) = \chi_j \odot \hat{f} \qquad (8)$$

As we noticed above, the Hadamard product is, essentially, point-wise multiplication of the given functions.

To obtain the inverse discrete Fourier transform of the shifted sequence $(\hat{f})_{[j]}$, we proceed in the following manner:

$$(F_n^{-1}((\hat{f})_{[j]})) = \frac{1}{n}\sum_{m=0}^{n-1}((\hat{f})_{[j]})(m)\chi_m^* = \frac{1}{n}\sum_{m=0}^{n-1} widehatf(m-j)\chi_m^* \qquad (9)$$

Evaluating at a particular $k \in Z/n$ results in

$$(F_n^{-1}(S_n^j \hat{f}))(k) = \frac{1}{n}\sum_{m=0}^{n-1}((\hat{f})_{[j]})(m)\chi_m^*(k) = \frac{1}{n}\sum_{m=0}^{n-1}\hat{f}(m-j)\chi_m^*(k) \qquad (10)$$

Letting $r = m - j$, results in

$$
\begin{aligned}
(F_n^{-1}((\hat{f})_{[j]}))(k) &= \frac{1}{n}\sum_{r=-j}^{n-1-j}\hat{f}(r)\chi_{r+j}^*(k) \\
&= \chi_j^*(k)\Big(\frac{1}{n}\sum_{r=-j}^{n-1-j} f(r)\chi_r^*(k)\Big) \qquad (11) \\
&= \chi_j^*(k)\Big(\frac{1}{n}\sum_{m=0}^{n-1} f_m\chi_m^*(k)\Big) = \chi_j^*(k) f_k
\end{aligned}
$$

Since the choice of $k \in Z/n$ was arbitrary, we have the following result

$$F_n^{-1}(S_n^j \hat{f}) = \chi_j^* \odot f \qquad (12)$$

or

$$(\chi_j^* \odot f)\hat{} = S_n^j \hat{f} \tag{13}$$

We notice that

$$\begin{aligned}(S_n^j \hat{f})(k) &= \sum_{m=0}^{n-1} S_n^j \hat{f}_m \delta_{[m]}(k) = \sum_{m=0}^{n-1} \hat{f}_m S_n^j \delta_{[m]}(k) \\ &= \sum_{m=0}^{n-1} \hat{f}_m \delta_{[m+j]}(k) = \sum_{m=0}^{n-1} f_m \delta_{[j]}(k-m) \\ &= (\delta_{[j]} * \hat{f})(k)\end{aligned} \tag{14}$$

Thus, Eq. (13) can be rewritten as

$$(\chi_j^* \odot f)\hat{} = \delta_{[j]} * \hat{f} \tag{15}$$

We can also rewrite (8) as follows

$$(\delta_{[j]} * f)\hat{} = \chi_j \odot \hat{f} \tag{16}$$

The generalization of (16) is obtained as follows: First,

$$h_m(F_n(\delta_{[m]} * f)) = h_m(\chi_m \odot \hat{f}) \tag{17}$$

Then, using the linearity properties of, both, $*, \odot$, as well as the linearity property of the DFT operator, we obtain the following result

$$\begin{aligned}F(h * f) &= F_n((\sum_{m \in Z/n} h_m \delta_{[m]}) * f) = \sum_{m \in Z/n} h_m F_n(\delta_{[m]} * f) \\ &= \sum_{m \in Z/n} h_m(\chi_m \odot \hat{f}) = (\sum_{m \in Z/n} h_m \chi_m) \odot \hat{f} \\ &= \hat{h} * \hat{f}\end{aligned} \tag{18}$$

This result is known as the cyclic convolution theorem; and it states that the discrete Fourier transform of the cyclic convolution of two n-point sequences is equal to the n-point Hadamard product of their respective discrete Fourier transform.

The generalization of Eq. (15) can also be obtained in the same manner as the generalization obtained for Eq. (16). First, we multiply by the scalar $\hat{h}_m \in C$:

$$\hat{h}_m F_n^{-1}(\delta_{[m]} * \hat{f}) = \hat{h}_m(\chi_m^* \odot f) \tag{19}$$

Then, using the linearity properties of $*, \odot, F_n^{-1}$, the following result is obtained

$$\begin{aligned} F_n^{-1}(\hat{h} * \hat{f}) &= F_n^{-1}((\sum_{m=0}^{n-1} \hat{h}_m \delta_{[m]}) * \hat{f}) = \sum_{m=0}^{n-1} \hat{h}_m F_n^{-1}(\delta_{[m]} * \hat{f}) \\ &= \sum_{m=0}^{n-1} \hat{h}_m (\chi_m^* \odot f) = n(\frac{1}{n} \sum_{m=0}^{n-1} \hat{h}_m \chi_m^*) \odot f \end{aligned} \tag{20}$$

Thus,

$$F_n^{-1}(\hat{h} * \hat{f}) = n(h \odot f) \tag{21}$$

or

$$F(h \odot f) = \frac{1}{n}(\hat{h} * \hat{f}) \tag{22}$$

The above result (22) states that the discrete Fourier transform of the Hadamard product of two n-point sequences is equal to the cyclic convolution of their respective discrete Fourier transforms, modified by a scalar value.

The cyclic correlation of any two n-point sequences $f, h \in L(Z/n)$ is defined as

$$g(k) = \sum_{j \in Z/n} f_j h_{j+k}, \quad k = 0, 1, 2, \ldots, n-1; \quad g \in L(Z/n) \tag{23}$$

Letting $m = j + k$ results in

$$\begin{aligned} \sum_{m=k}^{n+k-1} f_{m-k} h_m &= f_0 h_k + f_1 h_{k+1} + f_2 h_{k+2} + \ldots + f_{n-k-1} h_{n-1} \\ &+ f_{n-k} h_0 + f_{n-k+1} h_1 + \ldots + f_{n-2} h_{k-2} + f_{n-1} h_{k-1} \\ &= h_0 f_{n-k} + h_1 f_{1-k} + \ldots + h_{k-2} f_{n-2} + h_{k-1} f_{n-1} \\ &+ h_k f_0 + h_{k+1} f_1 + h_{k+2} f_2 + \ldots + h_{n-1} f_{n-k-1} \\ &= \sum_{j \in Z/n} f_{j-k} h_j = \sum_{j \in Z/n} (R_n f)_{k-j} h_j \\ &= \sum_{j \in Z/n} h_j (S_n^j (f^{(-)}))(k) = T_h(f^{(-)})(k) \\ &= (R_n f * h)(k) \end{aligned} \tag{24}$$

Thus, the cyclic correlation may be defined as

$$\begin{aligned} L(Z/n) \times L(Z/n) &\longrightarrow L(Z/n) \\ (f, h) &\longmapsto g = R_n f * h \end{aligned} \tag{25}$$

Taking the discrete Fourier transform of Eq. (24) will produce the following result

$$F_n(R_n f * h) = \widehat{R_n f} \odot \hat{h} \tag{26}$$

Using Eq. (5) given in §7, we have

$$F_n(R_n f * h) = n F_n^{-1} f \odot \hat{h} \tag{27}$$

6.7. The Reflection Operator:

An important operator is introduced at this point. We call this operator the reflection operator and denote it by the symbol R_n. Its action on the space of sequences $L(Z/n)$ will be described below. We first introduced its definition:

$$\begin{aligned} R_n &: L(Z/n) \longrightarrow L(Z/n) \\ (f) &\longmapsto R_n f = f^{(-)} \end{aligned} \tag{1}$$

where

$$(R_n f)(k) = f^{(-)}(k) = f_{n-k}, \quad k \in Z/n \tag{2}$$

Using definition (1), we establish the following result when we compose the discrete Fourier transform with the reflection operator

$$\begin{aligned} F_n(f^{(-)})(k) &= \sum_{j \in Z/n} f_j^{(-)} \chi_j(k) = \sum_{j \in Z/n} f_{n-j} \chi_j(k) \\ &= \sum_{j \in Z/n} f_{n-j} \chi_j(k) = \sum_{j \in Z/n} f_{n-j} e^{-2\pi i j k/n} \end{aligned} \tag{3}$$

Setting $m = n - j$, we obtain

$$\begin{aligned} F_n(f^{(-)})(k) &= \sum_{m=n}^{m=+1} f_m \chi_{n-m}(k) = \sum_{m=n}^{m=+1} f_m e^{-2\pi i (n-m)k/n} \\ &= e^{-2\pi i n k/n} \sum_{m \in Z/n} f_m e^{2\pi i m k/n} = n F_n^{-1}(f)(k) \end{aligned} \tag{4}$$

Thus, we have the following identity

$$F_n R_n f = n F_n^{-1} f, \quad f \in L(Z/n) \tag{5}$$

Or

$$F_n^{-1} = \frac{1}{n} F_n R_n \tag{6}$$

Noticing that

$$F_n^{-1}(F_n^{-1}) F_n = \frac{1}{n} F_n^{-1}(F_n R_n) F_n, \tag{7}$$

we write

$$F_n^{-1} = \frac{1}{n} F_n R_n = \frac{1}{n} R_n F_n \tag{8}$$

We observe that the reflection operator R_n commutes with the discrete Fourier transform operator F_n. We would like to show, however, that the reflection operator

does not commute with the shift operator S_n. Shifting by j units the sequence $f^{(-)}$, results in

$$(S_n^j(R_n f))(k) = (R_n f)(k-j) = f(j-k) \tag{9}$$

Applying the reflection operator to the sequence $f_{[j]}$ results in the following expression

$$(R_n(S_n^j f))(k) = (R_n f_{[j]})(k) = (f_{[j]})^{(-)}(k) = (f_{[j]})(-k) = f(-k-j) \tag{10}$$

From Eqs. (9) and (10), we conclude that the reflection operator R_n does not commute with the shift operator S_n.

We now determine the response of an LSI-FIR system T_h to an input sequence $f^{(-)} \in L(Z/n)$. We have

$$T_h(f^{(-)})(k) = f^{(-)} * h = \sum_{j \in Z/n} h_j S_n^j f^{(-)}(k) \tag{11}$$

Recalling that

$$\begin{aligned} T_h(\delta)(k) &= \sum_{j \in Z/n} h_j S_n^j \delta_{[0]}(k) = \sum_{j \in Z/n} h_j \delta_{[j]}(k) \\ &= \sum_{j \in Z/n} h_j \delta(j-k) = h(k) \end{aligned} \tag{12}$$

and that

$$T_h(f)(k) = f * h = \sum_{j \in Z/n} h_j S_n^j f(k) = \sum_{j \in Z/n} h_j f(k-j), \tag{13}$$

we write

$$T_h(f^{(-)})(k) = f^{(-)} * h = \sum_{j \in Z/n} h_j S_n^j f^{(-)}(k) = \sum_{j \in Z/n} h_j f^{(-)}(k-j) \tag{14}$$

With $f^{(-)}(k-j) = f(j-k)$, we write

$$T_h(f^{(-)})(k) = ((R_n f) * h)(k) = \sum_{j \in Z/n} h_j f(j-k) \tag{15}$$

We continue below describing the response of LSI-FIR systems whose input, or impulse response sequences have been acted upon by the reflection operator. For instance, the response $T_{(R_n f)}(h)$ to the system $T_{R_n f}$ is next determined.

$$\begin{aligned} T_{(R_n f)}(h)(k) &= \Big(\sum_{j=0}^{n-1} (R_n f)_j (S_n^j h)\Big)(k) \\ &= \sum_{j=0}^{n-1} (R_n f)_j h_{[j]}(k) = \sum_{j=0}^{n-1} f_{-j} h(k-j) \end{aligned} \tag{16}$$

If we set $m = k - j$, then

$$
\begin{aligned}
T_{(R_n f)}(h)(k) &= \sum_{m=k}^{-n+k+1} f_{-k+m} h_m \\
& f_0 h_k + f_{-1} h_{k-1} + f_{-2} h_{k-2} + \ldots + f_{1-k} h_1 \\
& + f_{-k} h_0 + f_{-k-1} h_{-1} + \ldots + f_{-n+2} h_{-n+k+2} + f_{-n+1} h_{-n+k+1} \\
& f_0 h_k + f_{n-1} h_{k-1} + f_{n-2} h_{k-2} + \ldots + f_{1-k} h_1 \\
& + f_{-k} h_0 + f_{n-k-1} h_{n-1} + \ldots + f_2 h_{k+2} + f_1 h_{k+1} \\
& h_0 f_{-k} + h_1 f_{1-k} + \ldots + h_{k-2} f_{n-2} + h_{k-1} f_{n-1} \\
& + h_k f_0 + h_{k+1} f_1 + h_{k+2} f_2 + \ldots + h_{n-1} f_{n-l-1} \\
&= \sum_{j=0}^{n-1} h_j f(j-k)
\end{aligned}
\tag{17}
$$

The above result allows us to write the following identity

$$T_h(R_n f) = T_{(R_n f)}(h) \tag{18}$$

Below, we establish some additional identities involving the reflection operator and LSI-FIR systems. We start by trying to determine the response of the system T_f to the signal input $h^{(-)}$:

$$T_f(R_n h) = f * R_n h, \quad f, h \in L(Z/n) \tag{19}$$

At any value $k \in Z/n$, we obtain

$$T_f(R_n h)(k) = \sum_{j \in Z/n} f_j h(j-k) \tag{20}$$

If we set $m = j - k$, it results in

$$
\begin{aligned}
T_f(R_n f)(k) &= \sum_{m=-k}^{n-k-1} h_m f_{m+k} \\
&= h_{-k} f_0 + h_{-k+1} f_1 + \ldots + h_{-1} f_{k-1} + h_0 f_k \\
& + h_1 f_{k+1} + \ldots + h_{n-k-2} f_{n-2} + h_{n-k-1} f_{n-1} \\
&= h_{n-k} f_0 + h_{n-k+1} f_1 + \ldots + h_{n-1} f_{n+k-1} + h_0 f_k \\
& + h_1 f_{k+1} + \ldots + h_{n-k-2} f_{n-2} + h_{n-k-1} f_{n-1} \\
&= h_0 f_k + h_1 f_{k+1} + \ldots + h_{n-k-2} f_{n-2} + h_{n-k-1} f_{n-1} \\
& + h_{n-k} f_0 + h_{n-k+1} f_1 + \ldots + h_{n-1} f_{n-1+k} \\
&= \sum_{j \in Z/n} h_j f(j+k)
\end{aligned}
\tag{21}
$$

To obtain the response $T_{(R_nh)}(f)$ of the system $T_{(R_nf)}$ to the input sequence f, we proceed as follows

$$T_{(R_nh)}(f) = f * (R_nh), \quad f, h \in L(Z/n) \tag{22}$$

Evaluating (22) at any $k \in Z/n$ results in

$$\begin{aligned} T_{(R_nh)}(f)(k) &= \sum_{j \in Z/n} (R_nh)_j (S_n^j f)(k) \\ &= \sum_{j \in Z/n} h_{-j} f_{[j]}(k) = \sum_{j=0}^{n-1} h_{-j} f(k-j) \end{aligned} \tag{23}$$

Setting $m = -j$ results in

$$\begin{aligned} T_{(R_nh)}(f)(k) = \sum_{m=0}^{-n+1} h_m f_{m+k} \\ &= h_0 f_k + h_{-1} f_{k-1} + h_{-2} f_{k-2} + \ldots + h_{-k} f_0 \\ &+ h_{-k-1} f_{-1} + \ldots + h_{-n+2} f_{-n+2+k} + h_{n+1} f_{-n+1+k} \\ &= h_0 f_k + h_1 f_{1-k} + h_2 f_{2+k} + \ldots + h_{n-k-1} f_{n-1} \\ &+ h_{n-k} f_0 + \ldots + h_{n-2} f_{n+k-2} + h_{n-1} f_{n+k-1} \\ &= \sum_{j \in Z/n} f_j h(j-k) \end{aligned} \tag{24}$$

We, thus establish the following result

$$T_{(R_nh)}(f) = T_f(R_nh) = f * R_nh \tag{25}$$

The next exercise will be to obtain the response of the system $T_{(R_nh)}$ to the input sequence $f^{(-)}$. We would like to point out that the objective of writing out explicitly the summation expression effecting the cyclic convolution operation is to get familiarity with sequence manipulation so that the DSP application examples described later on could be understood with more ease.

$$T_{(R_nh)}(R_nf) = \sum_{j \in Z/n} (R_nh)_j S_n^j (R_nf) \tag{26}$$

Evaluating (26) at any point $k \in Z/n$ results in

$$\begin{aligned} T_{(R_nh)}(R_nf)(k) &= \Big(\sum_{j \in Z/n} (R_nh)_j (S_n^j (R_nf))\Big)(k) = \sum_{j \in Z/n} h_{-j} (R_nf)_{[j]}(k) \\ &= \sum_{j \in Z/n} h_{-j} (R_nf)(k-j) = \sum_{j \in Z/n} h_{-j} f(j-k) \end{aligned} \tag{27}$$

Setting $m = -j$ results in

$$\begin{aligned} T_{(R_n h)}(R_n f)(k) &= \sum_{m=0}^{-n+1} h_m f(-m-k) \\ &= h_0 f_{-k} + h_{-1} f_{1-k} + h_{-2} f_{2-k} + \ldots + h_{-k} f_0 \\ &+ h_{-k-1} f_1 + \ldots + h_{-n+2} f_{n-k-2} + h_{-n+1} f_{n-k-1} \\ &= h_0 f_{-k} + h_{n-1} f_{-n-k+1} + h_{n-2} f_{-n-k+2} + \ldots + h_{n-k} f_{-n} \\ &+ h_{n-k-1} f_{-n+1} + \ldots + h_2 f_{-2-k} + h_1 f_{-1-k} \\ &= h_0 f_{-k} + h_1 f_{-1} + h_2 f_{-2-k} + \ldots + h_{n-k-1} f_{-n+1} \\ &+ h_{n-k} f_{-n} + \ldots + h_{n-2} f_{-n-k+2} + h_{n-1} f_{n-k-1} \\ &= \sum_{j \in Z/n} h_j f_{-k-j} \end{aligned} \tag{28}$$

Proceeding in the same manner, we now determine the response of the system $T_{(R_n f)}$ to the input signal $h^{(-)}$. We proceed in the following way

$$T_{(R_n f)}(R_n h) = \sum_{j \in Z/n} (R_n f)_j S_n^j (R_n h), \quad f, h \in L(Z/n) \tag{29}$$

Evaluating (29) at any $k \in Z/n$ results in

$$\begin{aligned} T_{(R_n f)}(R_n h)(k) &= \Big(\sum_{j \in Z/n} (R_n f)_j (S_n^j (R_n h)) \Big)(k) = \sum_{j \in Z/n} f_{-j} (R_n h)_{[j]}(k) \\ &= \sum_{j \in Z/n} f_{-j} (R_n h)(k-j) = \sum_{j \in Z/n} f_{-j} h(j-k) \end{aligned} \tag{30}$$

Setting $m = j - k$ results in

$$\begin{aligned} T_{(R_n f)}(R_n h)(k) &= \sum_{m=-k}^{n-k-1} h_m f_{-k-m} \\ &= h_{-k} f_0 + h_{1-k} f_{-1} + h_{2-k} f_{-2} + \ldots + h_{-1} f_{1-k} \\ &+ h_0 f_{-k} + h_1 f_{-k-1} + \ldots + h_{n-k-2} f_{2-n} + h_{n-k-1} f_{1-n} \\ &= h_{n-k} f_0 + h_{n-k+1} f_{-n-1} + h_{n-k+2} f_{n-2} + \ldots + h_{n-1} f_{n-k+1} \\ &+ h_0 f_{-k} + h_1 f_{-n-k-1} + \ldots + h_{n-k-2} f_{-n+2} + h_{n-k-1} f_{-n+1} \\ &= h_0 f_{-k} + h_1 f_{-1-k} + \ldots + h_{n-k-2} f_{-n+2} + h_{n-k-1} f_{-n+1} \\ &+ h_{n-k} f_0 + h_{n-k+1} f_{-n-1} + h_{n-k+2} f_{n-2} + \ldots + h_{n-1} f_{n-k+1} \\ &= \sum_{j \in Z/n} h_i f_{-k-j} \end{aligned} \tag{31}$$

And the following identity is established

$$T_{(R_n h)}(R_n f) = T_{(R_n f)}(R_n h) \tag{32}$$

We would like to determine the result of applying the reflection operator to the sequences $T_f(h), T_h(f)$. We start with the signal $T_h(f)$:

$$R_n(T_h(f)) = R_n(\sum_{j \in Z/n} h_j S_n^j f), \quad f, h \in L(Z/n) \tag{33}$$

Evaluating (33) at any value $k \in Z/n$ results in

$$\begin{aligned}(R_n(T_h(f))(k) &= \sum_{j \in Z/n} h_j (R_n(S_n^j f))(k) \\ &= \sum_{j \in Z/n} h_j (R_n(f_{[j]})(k) = \sum_{j \in Z/n} h_j f(-k-j)\end{aligned} \tag{34}$$

Also, applying R_n to $T_f(h)$ produces

$$R_n(T_f(h)) = R_n(\sum_{j \in Z/n} f_j S_n^j h), \quad f, h \in L(Z/n) \tag{35}$$

At any point $k \in Z/n$, we obtain

$$R_n(T_f(h))(k) = \sum_{j \in Z/n} f_j h_{-k-j} \tag{36}$$

Setting $m = -k - j$ results in

$$\begin{aligned}R_n(T_f(h))(k) &= \sum_{m=-k}^{-n-k-1} h_m f_{-k-m} \\ &= h_k f_0 + h_{-k-1} f_1 + h_{-k-2} f_2 + \ldots + h_{-n+1} f_{n-1-k} \\ &+ h_{-n} f_{n-k} + h_{-n-1} f_{n-k+1} + \ldots + h_{-n-k+2} f_{n-2} + h_{-n-k+1} f_{n-1} \\ &= h_{n-k} f_0 + h_{n-k-1} f_1 + h_{n-k-2} f_2 + \ldots + h_1 f_{-1-k} \\ &+ h_0 f_k + h_{n-1} f_{-k+1} + \ldots + h_{n-k+2} f_{-2} + h_{n-k+1} f_{-1} \\ &= h_0 f_{-k} + h_1 f_{-1-k} + \ldots + h_{n-k-2} f_{-n+2} + h_{n-k-1} f_{-n+1} \\ &+ h_{n-k} f_{-n} + \ldots + h_{n-k+1} f_{-n-1} + h_{n-k+2} f_{-n+2} + \ldots + h_{n-1} f_{-n-k+1} \\ &= \sum_{j \in Z/n} h_j f_{-k-j}\end{aligned} \tag{37}$$

Thus, the following identity follows

$$R_n(T_h(f)) = R_n(T_f(h)), \quad f, h \in L(Z/n) \tag{38}$$

It is worth noting that we could have used the commutativity property of the cyclic convolution operation to prove the set of identities stated above; however, we chose not to do this so that we can present in detail the work involving sequence indexing manipulations.

We now relate the reflection operator R_n and the shift operator S_n through the finite Fourier transform operator.

The finite Fourier transform of the Hadamard product $\chi_k \odot (S_n^j f)$ is now obtained:

$$F_n(\chi_k \odot S_n^j f) = \hat{\chi}_k * (f_{[j]})\hat{} \tag{39}$$

where $f \in L(Z/n)$, S_n is the shift operator and $\chi_k \in L(Z/n)$ is a characteristic sequence. Evaluating $\hat{\chi}_k$ results in

$$\hat{\chi}_k = nS_n^{n-k}\delta = n\delta_{[n-k]} * \delta \tag{40}$$

Evaluating $F_n(f_{[j]})$ results in

$$(f_{[j]})\hat{} = \chi_j \odot \hat{f} \tag{41}$$

Combining (40), (41) produces

$$\begin{aligned} F_n(\chi \odot S_n^j f) &= \hat{\chi}_k * (\chi_j \odot \hat{f}) \\ &= n\delta_{[n-k]} * (\chi_j \odot \hat{f}) = n(\chi_j \odot \hat{f})_{[n-k]} \end{aligned} \tag{42}$$

The finite Fourier transform of the Hadamard product $S_n^j \chi_k \odot f$ is obtained as follows:

$$F_n(S_n^j \chi_k \odot f) = (S_n^j \chi_k)\hat{} * \hat{f} \tag{43}$$

where $f \in L(Z/n)$, S_n is the shift operator, and $\chi_k \in L(Z/n)$ is a characteristic function. We recall that

$$(S_n^j \chi_k)\hat{} = \chi_j \odot \hat{\chi}_k = \chi_j \odot (n\delta_{[n-k]}) = n\chi_j(n-k)\delta \tag{44}$$

Substituting this result into (43) produces

$$F_n(S_n^j \chi_k \odot f) = n\chi_j(n-k)\delta * \hat{f} = n\chi_j(n-k)\hat{f} \tag{45}$$

We now describe the finite Fourier transform of the composition $R_n S_n^j R_n$, R_n and S_n the reflection and shift operators, respectively, acting on a given signal $f \in Z/n$:

$$\begin{aligned}(R_n S^j R_n f)\widehat{\ } &= R_n(S_n^j f^{(-)})\widehat{\ } = R_n((f^{(-)})_{[j]})\widehat{\ } \\ &= R_n(\chi_j \odot (f^{(-)})\widehat{\ }) = R_n(\chi_j \odot (\widehat{f})^{(-)}) \\ &= \chi_j^{(-)} \odot \widehat{f} = \chi_j^* \odot \widehat{f} \\ &= (F_n^{-1}(\chi_j^* \odot \widehat{f}))\widehat{\ } = (\delta_{[n-j]} * f)\widehat{\ }\end{aligned} \tag{46}$$

Thus, we have

$$(R_n S_n^j R_n f)\widehat{\ } = (S_n^{n-j} f)\widehat{\ } \tag{47}$$

Since the choice of $f \in L(Z/n)$ was arbitrary, we establish the following identity

$$R_n S_n^j R_n = S_n \tag{48}$$

We now determine the finite Fourier transform of the complex conjugate of a given function $f \in L(Z/n)$; and relate it to the reflection operator R_n and the finite Fourier transform of the original signal.

$$\begin{aligned}(f^*)\widehat{\ }(k) &= \sum_{j \in Z/n} f_j^* \chi_j(k) = (\sum_{j \in Z/n} f_j \chi_j^*(k))^* \\ &= (\sum_{j \in Z/n} f_j \chi_j(-k))^* = (R_n(\sum_{j \in Z/n} f_j \chi_j(k)))^* \\ &= ((R_n \widehat{f})(k))^* = R_n(\widehat{f})^*(k)\end{aligned} \tag{49}$$

We conclude

$$(f^*)\widehat{\ } = R_n(\widehat{f})^* \tag{50}$$

To elaborate further on this computation, we proceed as follows

$$\begin{aligned}F_n(f^*)(k) &= \widehat{f^*}(k) = \sum_{j \in Z/n} f_j^* \chi_j(k) \\ &= (\sum_{j \in Z/n} f_j \chi_j^*(k)) = (\sum_{j \in Z/n} f_j e^{2\pi i j k/n})^*\end{aligned} \tag{51}$$

Let $m = -j$, then

$$\begin{aligned}F_n(f^*)(k) &= (\sum_{j \in Z/n} f_j e^{2\pi i j k/n})^* = (\sum_{m=0}^{n-1} f_m^{(-)} e^{-2\pi i m k/n})^* \\ &= (f_0^{(-)}\chi_k(0) + f_{-1}^{(-1)}\chi_k(-1) + f_{-2}^{(-)}\chi_k(-2) + \ldots + f_{-n+1}^{(-)}\chi_k(-n+1))^* \\ &= (f_0^{(-)}\chi_k(0) + f_1\chi_k(1) + \ldots + f_{n-2}^{(-)}\chi_k(n-2) + f_{n-1}^{(-)}\chi_k(n-1))^* \\ &= (\sum_{j=0}^{n-1} f_j^{(-)} e^{-2\pi i j k/n})^* = \widehat{f^{(-)}}(k)\end{aligned} \tag{52}$$

Since the choice of $k \in Z/n$ was arbitrary, we have the following identity

$$\widehat{f^*} = (\widehat{f^{(-)}})^* \tag{53}$$

We also notice that

$$\begin{aligned}(f^*\widehat{)}(k) &= \sum_{j\in Z/n} f^*\chi_j(k) = \Big(\sum_{j\in Z/n} f_j\chi_j^*\Big)^* \\ &= \Big(\sum_{j\in Z/n} f_j e^{2\pi ijk/n}\Big)^* = \Big(\sum_{j\in Z/n} f_j e^{-2\pi ij(-k)/n}\Big)^* \\ &= \Big(\sum_{j\in Z/n} f_j\chi_j(-k)\Big)^* = (\hat{f}(-k))^*\end{aligned} \tag{54}$$

6.8. Matrix Representation of LSI-FIR Systems:

In this section we discuss the representation of LSI-FIR through matrices. Since each n-dimensional LSI-FIR system $T_h: L(Z/n) \to L(Z/n)$ represents a linear transformation on the space $L(Z/n)$, T_h is determined by its action on a set of basis vectors (signals) spanning $L(Z/n)$. If we choose as reference the standard basis set $\{\delta_{[j]}: j \in Z/n\}$, then each signal $T_h(\delta_{[j]}) \in L(Z/n)$ can be uniquely expressed as a linear combination of the basis set. We write

$$T_h(\delta_{[k]}) = \sum_{j \in Z/n} h_{j,k}\delta_{[j]} \tag{1}$$

where the set of scalars

$$\{h_{j,k}: j \in Z/n\}, \quad k \in Z/n \tag{2}$$

represents the vector coordinates of the given signal $T_h(\delta_{[k]})$, $k \in Z/n$, with respect to the standard basis set. The signal $T_h(\delta_{[k]})$ can also be written as

$$T_h(\delta_{[k]}) = \sum_{j \in Z/n} T_h(\delta_{[k]})(j)\delta_{[j]} \tag{3}$$

where

$$\begin{aligned} T_h(\delta_{[k]})(j) &= \sum_{m \in Z/n} h_m (S_n^m \delta_{[k]})(j) = \sum_{m \in Z/n} h_m \delta_{[k+m]}(j) \\ &= \sum_{m \in Z/n} h_m \delta(j-k-m) = h_{j-k} = S_n^k h)(j) \end{aligned} \tag{4}$$

Thus, we write

$$\begin{aligned} T_h(\delta_{[k]}) &= \sum_{j \in Z/n} h_{j,k}\delta_{[j]} = \sum_{j \in Z/n} h_{j-k}\delta_{[j]} \\ &= \sum_{j \in Z/n} (S_n^k h)(j) S_n^j \delta = T_{(S_n^k h)}(h) = S_n^k h \end{aligned} \tag{5}$$

Next, we define the matrix H_n as follows

$$H_n = [h_{j,k}]_{0 \le j,k < n} = [h_{j-k}]_{0 \le j,k < n} \tag{6}$$

The matrix H_n, thus, have the following form

$$H_n = \begin{bmatrix} h_0 & h_{n-1} & h_{n-2} & \cdots & h_1 \\ h_1 & h_0 & h_{n-1} & \cdots & h_2 \\ h_2 & h_1 & h_0 & \cdots & h_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ h_{n-1} & h_{n-2} & h_{n-3} & \cdots & h_0 \end{bmatrix} \tag{7}$$

We notice that the columns of H_n are formed by shifted versions of the coordinate vector representation of the signal h; that is, we can write H_n as

$$H_n = [I_n h, S_n h, S_n^2, \ldots, S_n^{n-1} h] \tag{8}$$

where S_n is the matrix representing the shift operator S_n; and h is the coordinate vector representation of the signal h.

We would like to describe in more details how the matrix H_n, representing the system T_h, is obtained. Starting with expression (1) above, we rewrite

$$\begin{aligned}(\delta_{[k]}) &= \sum_{j \in Z/n} h_{j,k} S_n^j \delta, \quad h_{j,k} \in C \\ &= h_{0,k}\delta_{[0]} + h_{1,k}\delta_{[1]} + \ldots + h_{n-1,k}\delta_{[n-1]}\end{aligned} \tag{9}$$

Evaluating this expression (9) at different values of $k \in Z/n$ results in the following set of identities:

$$\begin{aligned}T_h(\delta_{[0]}) &= h_{0,0}\delta_{[0]} + h_{1,0}\delta_{[1]} + \ldots + h_{n-1,0}\delta_{[n-1]} \\ T_h(\delta_{[1]}) &= h_{0,1}\delta_{[0]} + h_{1,1}\delta_{[1]} + \ldots + h_{n-1,1}\delta_{[n-1]} \\ &\vdots \\ T_h(\delta_{[n-1]}) &= h_{0,n-1}\delta_{[0]} + h_{1,n-1}\delta_{[1]} + \ldots + h_{n-1,n-1}\delta_{[n-1]}\end{aligned} \tag{10}$$

We write these identities in an array form :

$$\begin{bmatrix} T_h(\delta_{[0]}) \\ T_h(\delta_{[1]}) \\ \vdots \\ T_h(\delta_{[n-1]}) \end{bmatrix} = \begin{bmatrix} h_{0,0} & h_{1,0} & \cdots & h_{n-1,0} \\ h_{0,1} & h_{1,1} & \cdots & h_{n-1,1} \\ \vdots & \vdots & \vdots & \vdots \\ h_{0,n-1} & h_{1,n-1} & \cdots & h_{n-1,n-1} \end{bmatrix} \cdot \begin{bmatrix} \delta_{[0]} \\ \delta_{[1]} \\ \vdots \\ \delta_{[n-1]} \end{bmatrix} \tag{11}$$

We know obtain a vector-matrix representation of a cyclic convolution operation described in §6.5. Given a system T_h and a signal $f \in L(Z/n)$, the response $g = T_h(f)$ is obtained as follows

$$\begin{aligned}g = T_h(f) &= T_h\Big(\sum_{k \in Z/n} f_k \delta_{[k]}\Big) \\ &= \sum_{k \in Z/n} f_k T_h(\delta_{[k]}) = \sum_{j \in Z/n} g_j \delta_{[j]}\end{aligned} \tag{12}$$

Expanding the above sum, we obtain

$$T_h(f) = f_0 T_h(\delta_{[0]}) + f_1 T_h(\delta_{[1]}) + \ldots + f_{n-1} T_h(\delta_{[n-1]}) \tag{13}$$

where

$$\begin{aligned} f_0 T_h(\delta_{[0]}) &= f_0 h_{0,0}\delta_{[0]} + f_0 h_{1,0}\delta_{[1]} + \ldots + f_0 h_{n-1,0}\delta_{[n-1]} \\ f_1 T_h(\delta_{[1]}) &= f_1 h_{0,1}\delta_{[0]} + f_1 h_{1,1}\delta_{[1]} + \ldots + f_1 h_{n-1,1}\delta_{[n-1]} \\ &\vdots \\ f_{n-1} T_h(\delta_{[n-1]}) &= f_{n-1} h_{0,n-1}\delta_{[0]} + f_{n-1} h_{1,n-1}\delta_{[n-1]} + \ldots + f_{n-1} h_{n-1,n-1}\delta_{[n-1]} \end{aligned} \tag{13}$$

The addition of the above set of equations produces the following expression

$$\begin{aligned} g = T_h(f) &= \sum_{j\in Z/n} g_j \delta_{[j]}, \quad f \in L(Z/n) \\ &= (f_0 h_{0,0} + f_1 h_{0,1} + \ldots + f_{n-1} h_{0,n-1})\delta_{[0]} \\ &+ (f_0 h_{1,0} + f_1 h_{1,1} + \ldots + f_{n-1} h_{1,n-1})\delta_{[1]} + \ldots \\ &+ (f_0 h_{n-1,1} + f_1 h_{n-1,1} + \ldots + f_{n-1} h_{n-1,n-1})\delta_{[n-1]} \\ &= T_h(f) = \sum_{j\in Z/n} \Big(\sum_{k\in Z/n} f_k h_{j,k} \Big)\delta_{[j]} \end{aligned} \tag{14}$$

where

$$g(m) = T_h(f) = \sum_{j\in Z/n} \Big(\sum_{k\in Z/n} f_k h_{j,k} \Big)\delta_{[j]}(m) = \sum_{k\in Z/n} f_k h_{m,k} \tag{15}$$

in vector notation , we have

$$\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_j \\ \vdots \\ g_{n-1} \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{n-1} f_k h_{0,k} \\ \sum_{k=0}^{n-1} f_k h_{1,k} \\ \vdots \\ \sum_{k=0}^{n-1} f_k h_{j,k} \\ \vdots \\ \sum_{k=0}^{n-1} f_k h_{n-1,k} \end{bmatrix} \tag{16}$$

Factoring out the vector f form (16) above, we obtain the following matrix-vector representation

$$\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_j \\ \vdots \\ g_{n-1} \end{bmatrix} = \begin{bmatrix} h_{0,0} & h_{0,1} & \ldots & h_{0,k} & \ldots & h_{0,n-1} \\ h_{1,0} & h_{1,1} & \ldots & h_{1,k} & \ldots & h_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_{j,0} & h_{j,1} & \ldots & h_{j,k} & \ldots & h_{j,n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_{n-1,0} & h_{n-1,1} & \ldots & h_{n-1,k} & \ldots & h_{n-1,n-1} \end{bmatrix} \cdot \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_k \\ \vdots \\ f_{n-1} \end{bmatrix} \tag{17}$$

Recalling that $h_{j,k} = = h(j-k)$, $j,k \in Z/n$, we rewrite (17) as

$$\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_j \\ \vdots \\ g_{n-1} \end{bmatrix} = \begin{bmatrix} h_0 & h_{n-1} & \ldots & h_{n-k} & \ldots & h_1 \\ h_1 & h_0 & \ldots & h_{1-k} & \ldots & h_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_j & h_{j-1} & \ldots & h_{j-k} & \ldots & h_{j+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_{n-1} & h_{n-2} & \ldots & h_{n-1-k} & \ldots & h_0 \end{bmatrix} \cdot \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_k \\ \vdots \\ f_{n-1} \end{bmatrix} \tag{18}$$

The above matrix-vector operation $g = H_n f$ represents the cyclic convolution operation $g = f * h = T_h(f)$, where we have used the same symbols f, g to denote, both, the coordinate vector representation of the signals f, g, respectively, as well as the signals themselves; and the matrix H_n represents the system T_h:

$$\begin{aligned} H_n &= [T_h(\delta_{[0]}), T_h(\delta_{[1]}), \ldots, T_h(\delta_{[n-1]})] \\ &= [T_{\delta[0]}(h), T_{\delta_{[1]}}(h), \ldots, T_{\delta_{[n-1]}}(h)] \\ &= [I_n T_h(\delta), S_n T_h(\delta), \ldots, S_n^{n-1} T_h(\delta)] \end{aligned} \tag{19}$$

Here, again, we have used commas to separate the vectors; and we have used the same notation used for the signals in order to denote the coordinate vector representation of the signals.

The computation of the cyclic convolution operation

$$g = f * h = T_h(f), \quad f, h \in L(Z/n) \tag{20}$$

is now performed by direct substitution into the defining equation

$$g = T_h(f) = T_h(\sum_{k=0}^{n-1} f_k \delta_{[k]}) \tag{21}$$

and proceed in the following manner

$$\begin{aligned} T_h(f) &= T_h(\sum_{k \in Z/n} f_k \delta_{[k]}) = \sum_{k \in Z/n} f_k T_h(\delta_{[k]}) \\ &= \sum_{k \in Z/n} f_k (\sum_{j \in Z/n} h_{j-k} \delta_{[j]}) = \sum_{j \in Z/n} (\sum_{k \in Z/n} h_{j-k} f_k) \delta_{[j]} \end{aligned} \tag{22}$$

Evaluating $g \in L(Z/n)$ at a particular index value $j \in Z/n$ results in

$$\begin{aligned} g(j) &= T_h(f)(j) = \sum_{j \in Z/n} (\sum_{k \in Z/n} h_{j-k} f_k) \delta_{[j]}(j) \\ &= \sum_{j \in Z/n} (\sum_{k \in Z/n} h_{j-k} f_k) \delta = \sum_{k \in Z/n} h_{j-k} f_k \end{aligned} \tag{23}$$

6.9. Spectral Properties of LSI-FIR systems:

In this section we will describe the spectral properties of LSI-FIR systems. A shift invariant linear operator acting on an n-dimensional vector space may be represented in the frequency domain by using the concepts of eigenfunctions (eigenvectors) and eigenvalues. The eigenvalues correspond to the natural frequencies encountered in the spectral representation of the impulse response signal of a given LSI-FIR system. We will be more explicit later on in describing the relationship existing between the eigenvalues (and their associated eigenfunctions) of a given LSI-FIR operator T_h and the frequency components of the associated impulse response sequence h. We start the section describing some properties of the system $T_{\delta_{[1]}}$ which are essentially the same as the properties of the shift operator S_n.

The simplest LSI-FIR system, apart from the trivial system, i.e., the system represented by the identity operator I_n, is the system represented by the shift operator S_n. This system is sometimes called the unit delay system because its digital electronics hardware implementation may be accomplished by using a single delay element. We use the same symbol S_n to denote the matrix representation of the shift operator S_n. This matrix representation is now given.

Recalling that

$$T_{\delta_{[1]}} = \sum_{j \in Z/n} \delta_{[1]}(j) S_n^j = S^1 = S_n, \tag{1}$$

we have,

$$T_{\delta[1]}(\delta_{[k]}) = \delta_{[1]} * \delta_{[k]} = S_n \delta_{[k]} = \delta_{[k+1]} \tag{2}$$

The matrix S_n representing the shift operator S_n is obtained by allowing the vector representation (with respect to the standard basis set $\{\delta_{[k]} : k \in Z/n\}$) of the signal $T_{\delta_{[1]}}(\delta_{[k]})$, $k \in Z/n$, become the columns of the matrix S_n:

$$\begin{aligned} S_n &= \left[T_{\delta_{[1]}}(\delta_{[0]}), T_{\delta_{[1]}}(\delta_{[1]}), \ldots, T_{\delta_{[1]}}(\delta_{[n-1]})\right] \\ &= \left[\delta_{[1]}, \delta_{[2]}, \ldots, \delta_{[n-1]}, \delta_{[0]}\right] \end{aligned} \tag{3}$$

where we have separated by commas the columns of S_n for legibility. The matrix S_n becomes

$$S_n = \begin{bmatrix} 0 & 0 & \ldots & 0 & 1 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{bmatrix} \tag{4}$$

An important property of the S_n operator matrix is that any LSI-FIR system T_h may be represented by a matrix H_n which can be written as a sum of powers of the matrix S_n pre-multiplied by a diagonal matrix D_{h_j}:

$$H_n = \sum_{j \in Z/n} D_{h_j} S_n^j = \sum_{j \in Z/n} (h_j \otimes S_n^j) \tag{5}$$

where

$$D_{h_j} = \begin{bmatrix} h_j & & & \\ & h_j & & \\ & & \ddots & \\ & & & h_j \end{bmatrix}, \quad h_j = h(j), \quad j \in Z/n \tag{6}$$

and the symbol $\otimes$ stands for tensor product. We give the following simple example to illustrate this representation

<u>Example 1:</u> **Take $n = 4$. We have**

$$D_{h_0} \cdot I_4 = \begin{bmatrix} h_0 & 0 & 0 & 0 \\ 0 & h_0 & 0 & 0 \\ 0 & 0 & h_0 & 0 \\ 0 & 0 & 0 & h_0 \end{bmatrix}, \quad D_{h_1} \cdot S_4 = \begin{bmatrix} 0 & 0 & 0 & h_1 \\ h_1 & 0 & 0 & 0 \\ 0 & h_1 & 0 & 0 \\ 0 & 0 & h_1 & 0 \end{bmatrix}$$

$$D_{h_2} \cdot S_4^2 = \begin{bmatrix} 0 & 0 & h_2 & 0 \\ 0 & 0 & 0 & h_2 \\ h_2 & 0 & 0 & 0 \\ 0 & h_2 & 0 & 0 \end{bmatrix}, \quad D_{h_3} \cdot S_4^3 = \begin{bmatrix} 0 & h_3 & 0 & 0 \\ 0 & 0 & h_3 & 0 \\ 0 & 0 & 0 & h_3 \\ h_3 & 0 & 0 & 0 \end{bmatrix}$$

$$H_4 = \sum_{j \in Z/4} D_{h_j} S_4^j = \sum_{j \in Z/4} (h_j \otimes S_4^j)$$

$$H_4 = \begin{bmatrix} h_0 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_3 \\ h_3 & h_2 & h_1 & h_0 \end{bmatrix}$$

The matrix F_n, called the DFT matrix and representing the discrete Fourier transform (DFT) operator F_n, is obtained by first determining $F_n(\delta_{[k]})$, $k \in Z/n$:

$$F_n(\delta_{[k]}) = {}^n\chi_k, \quad k \in Z/n, \quad {}^n\chi_k(j) = e^{-2\pi i k j/n} = w_n^{jk} \tag{7}$$

The matrix F_n is obtained by writing the coordinate vector representation of the signal set $\{F_n(\delta_{[k]}): k \in Z/n\}$ as the columns of F_n:

$$F_n = [\chi_0, \chi_1, \chi_2, \cdots, \chi_{n-1}] \tag{8}$$

$$F_n = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_n & w_n^2 & \cdots & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & \cdots & w_n^{n-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & w_n^{n-1} & w_n^{n-2} & \cdots & w_n \end{bmatrix} \tag{9}$$

We now take an LSI-FIR system T_h, which is represented by the matrix H_n, and obtain the discrete Fourier transform of the system response to a given input signal $f \in L(Z/n)$. Taking the DFT of the response $g = T_h(f)$, and using the cyclic convolution theorem, produces the following result

$$\begin{aligned} F_n(T_h(f)) &= F_n(h) \odot F_n(f) \\ (F_n \circ T_h)(f) &= (F_n(h) \odot F_n)(f) \end{aligned} \tag{10}$$

Since the choice of $f \in Z/n$ was arbitrary, we obtain the following important result

$$FT_h = F_n(h) \odot F_n \tag{11}$$

or, emphasizing the diagonalization of T_h by the action of the F_n operator,

$$F_n T_h F_n^{-1} = F_n(h) \odot I_n \tag{12}$$

The expression $F_n(h) \odot I_n$ is denoted by $D_{F(h)} = D_{\widehat{h}}$, where a matrix representation of $D_{\widehat{h}}$ is given by

$$D_{\widehat{h}} = \begin{bmatrix} \widehat{h}_0 & & & & \\ & \widehat{h}_1 & & & \\ & & \widehat{h}_2 & & \\ & & & \ddots & \\ & & & & \widehat{h}_{n-1} \end{bmatrix}, \quad \widehat{h}_j = (F(h))(j), \quad j \in Z/n \tag{13}$$

Thus,

$$F_n T_h F_n^{-1} = D_{\widehat{h}} \tag{14}$$

The identity (14), above, states that the discrete Fourier transform operator F_n diagonalizes every operator T_h representing an LSI-FIR system. The eigenvalues of the system T_h are the evaluations $\widehat{h}(k)$, $k \in Z/n$. For the particular case when the operator representing a given LSI-FIR system is the shift operator S_n, the diagonalization process gives rise to the following result

$$F_n S_n F_n^{-1} = F_n T_{\delta_{[1]}} F_n^{-1} = D_{\widehat{\delta}_{[1]}} \tag{15}$$

Recalling that

$$\widehat{\delta}_{[1]} = F_n(\delta_{[1]}) = {}^n\chi_1 = \chi, \quad \chi(j) = e^{-2\pi i j/n} = w_n^j, \quad j \in Z/n, \tag{16}$$

we write

$$D_{\widehat{\delta}_{[1]}} = \begin{bmatrix} 1 & & & & \\ & w_n & & & \\ & & w_n^2 & & \\ & & & \ddots & \\ & & & & w_n^{n-1} \end{bmatrix}, \quad w_n = e^{-2\pi i/n} \tag{17}$$

Thus, the diagonalization of the shift operator S_n, performed by the DFT operator F, results in a matrix representation whose entries are the n-th roots of unity: $\{1, w_n, w_n^2, \ldots, w_n^{n-1}\}$.

As we discussed previously in §6.4, the discrete Fourier transform F_n can be interpreted as a unitary operator which effects a change of basis on any given n-point signal $f \in L(Z/n)$; that is, the action of the operator F_n on a signal f, whose vector representation is given, for instance, with respect to the canonical (standard) basis $\{\delta_{[0]}, \delta_{[1]}, \ldots, \delta_{[n-1]}\}$, results in a function $F(f) = \widehat{f}$ which can be interpreted as the same signal f; but, given with respect to a different basis set, the latter set formed by the eigenfunctions of the shift operator S_n.

Expression (16) states that the eigenvalues of the shift operator S_n are the n-th roots of unity; therefore, there are distinct. Furthermore, the orthogonality of the eigenfunctions makes the change of basis expressed by (15) a unitary transformation. If w_n^k, $k \in Z/n$, for instance, is a given eigenvalue associated with the eigenfunction f of the operator S_n, then the same eigenvalue w_n^k is associated with the eigenfunction Ff of the operator $D_{\widehat{\delta}_{[1]}}$. The mapping $D_{\widehat{\delta}_{[1]}} = F_n S_n F_n^{-1}$ is termed a similarity transformation. Below, we give a diagram depicting the similarity transformation $D_{\widehat{\delta}_{[1]}} = F_n S_n F_n^{-1}$:

$$\begin{array}{ccc} \left[\{\delta_{[j]} : j \in Z/n\},\ S_n\right] : L(Z/n) & \longrightarrow & L(Z/n) \\ \downarrow F_n & & \uparrow F_n^{-1} \\ \left[\{\chi_k : k \in Z/n\},\ D_{\widehat{\delta}_{[1]}}\right] : L(Z/n) & \longrightarrow & L(Z/n) \end{array}$$

6.10. Implementations of LSI-FIR Systems:

This section serves to describe formulations for hardware implementations of LSI-FIR systems or filters using fast Fourier transform algorithms. We have discussed previously the importance of LSI-FIR filters in present and future digital signal processing applications. Most applications, specially real time applications, require fast signal processing hardware in order to perform the desired tasks efficiently. This implies the necessity to implement very fast LSI-FIR filters [17], [18]. One way to obtain fast hardware implementations is by an indirect approach which consists of implementing the convolution operation performed by an LSI-FIR filter using fast Fourier transform algorithms [19] through the convolution theorem which we restate below. We propose in this work to improve this approach by using the language of tensor products as a tool to aid in obtaining simplified hardware structures. In order to improve the hardware simplification procedure, we also take into consideration properties of the impulse response sequence which characterizes the given LSI-FIR filter. To analyze the properties of the impulse response sequence, we use, in turn, some of the linear algebra tools that we developed in the previous sections dealing with LSI-FIR systems. We start this section by reviewing some basic concepts and introducing some definitions. We then proceed to describe formulations for implementations of LSI-FIR systems using FFT algorithms.

It is important to review the two main methods of representing LSI-FIR filters for the purpose of hardware implementation. We will use some of the linear algebra tools described in the previous sections on LSI-FIR systems. This will allow us to relate the results of this section with with previous works and applications on LSI-FIR systems [20], [21]. The two main methods for representing LSI-FIR filters are usually called the time domain representation and the frequency domain representation. The time domain representation follows readily when we express an LSI-FIR system as a linear combination of powers of the shift operator; that is, the time domain representation of a given LSI-FIR filter T_h is given by

$$T_h = \sum_{j \in Z/n} h_j S_n^j \tag{1}$$

The frequency domain representation is obtained by taking the Z-transform of the impulse response sequence which characterizes the given system T_h. We would

like to describe this representation in more details since it is most often used. First, let us discuss some ideas about polynomial functions and introduce the Z-transform of a n-point sequence $h \in L(Z/n)$.

Associate with an n-point sequence

$$h = [h_0 \quad h_1 \quad \ldots \quad h_{n-1}], \tag{2}$$

the $(n-1)$-th degree polynomial p_h in the indeterminate α

$$p_h = h_0 + h_1\alpha + \ldots + h_{n-1}\alpha^{n-1}, \tag{3}$$

and the polynomial function

$$p_h(z^{-1}) = \sum_{j \in Z/n} h_j z^{-1} \equiv H(z), \quad z \in C \tag{4}$$

The polynomial function $p_h(z^{-1})$ belongs to the ordered set (the order induced by the natural order of Z/n)

$$Z(h) = \{p_h((z^{-1})^k): \quad k \in Z/n\} \tag{5}$$

where

$$p_h((z^{-1})^k) = \sum_{j \in Z/n} h_j z^{-jk}, \quad k \in Z/n \tag{6}$$

and the index product $j \cdot k$ is taken modulo n. Since the elements of $Z(h)$ become complex numbers when $z \in C$ is fixed, we can think of this set as an n-point sequence in $L(Z/n)$. Thus, the k-th element of the sequence Z is given by

$$Z(h)(k) = p_h(z^{-k}) \equiv H_k(z), \quad z \in C \text{ fixed} \tag{7}$$

Another way of viewing the set $Z(h)$ is to fixed the index value k (the value of k usually chosen to be one (1)) and allow z to become a variable, taking the entire complex plane C except the origin. The expression $Z(h)$ is then called the Z-transform of the n-point sequence $h \in L(Z/n)$, and is written as

$$Z(h) = p_h(z^{-1}) = \sum_{j \in Z/n} h_j z^{-j} \equiv H(z), \quad z \in C \tag{8}$$

We notice that the set $Z(h)$ now consists of a single element, namely $H(z)$; and by allowing z to take on values on C, $H(z)$ becomes an analytic function on the entire

complex plane except at the origin. The function $H(z)$ is usually termed the system function associated with the impulse response sequence h.

We now proceed to obtain the frequency domain representation of an LSI-FIR system T_h. We first recall that given a system T_h, its action on a input sequence $x \in L(Z/n)$ results in the cyclic convolution of this sequence x and the impulse response sequence which characterizes the system T_h. That is, if we call $y \in Z/n$ the output sequence which results when T_h acts on the input sequence x, this sequence is given by

$$y = T_h x = \sum_{j \in Z/n} h_j S_n^j x = x * h \tag{9}$$

We also would like to obtain the Z-transform of the shifted sequence $x_{[j]} = S_n^j x$, which is obtained as follows:

$$\begin{aligned} Z(x_{[j]}) &= \sum_{m \in Z/n} x_{[j]}(m) z^{-m} \\ &= \sum_{m \in Z/n} x(m-j) z^{-m} \\ &= \sum_{k \in Z/n} x_k z^{-(k+j)} \\ &= z^{-j} \sum_{k \in Z/n} x_k z^{-k} = z^{-j} X(z) \end{aligned} \tag{10}$$

where $X(z)$ is the Z-transform of the n-point input sequence x.

If we now take the Z-transform of the cyclic convolution given above ($Eq.(9)$), we obtain the following result

$$\begin{aligned} Z(y) = Z(x*y) &= Z\Big(\sum_{j \in Z/n} h_j S_n^j x\Big) \\ &= \sum_{j \in Z/n} h_j Z(S_n^j x) = \sum_{j \in Z/n} h_j z^{-j} X(z) \\ &= X(z) \sum_{j \in Z/n} h_j z^{-j} = X(z) H(z) \equiv Y(z) \end{aligned} \tag{11}$$

By making an analogy between the expression

$$y = \sum_{j \in Z/n} h_j S_n^j x = x * h \tag{12}$$

and the expression

$$Y(z) = \sum_{j \in Z/n} h_j z^{-j} X(z) = X(z) H(z), \tag{13}$$

the expression

$$H(z) = \sum_{j \in Z/n} h_j z^{-j} \tag{14}$$

is usually termed the "frequency domain representation" of the LSI-FIR system T_h. Thus, the frequency domain representation of a given LSI-FIR system T_h is its associated system function $H(z)$.

The hardware implementation of either the time domain or frequency domain representations of an LSI-FIR system is usually accomplished by identifying either the shift operator S_n in Eq. (12), or the multiplication element z^{-1} in Eq. (13), with a unit delay element device in digital signal processing hardware.

If we allow z in the system function $H(z)$ to take values only on the unit circle, we obtain the Fourier transform of the n-point sequence h. Thus, at $z = e^{i\nu} = e^{2\pi i f}$, we obtain

$$H(e^{i\nu}) = p_h(z^{-1} = e^{-2\pi i f}) = \sum_{j \in Z/n} h_j e^{-2\pi i j f}, \quad 0 \leq f < 1 \tag{15}$$

The discrete Fourier transform (DFT) of the n-point sequence h is obtained by setting fixed z in the sequence $Z(h)$ to the primitive n-th root of unity $e^{2\pi i/n} = w_n^{-1}$. In this way, the k-th term of the discrete Fourier transform of h is given by

$$\hat{h}(k) = F_n(h)(k) = \sum_{j \in Z/n} h_j e^{-2\pi i j k/n} = H_k(e^{2\pi i/n}) \tag{16}$$

Since k takes on values on the set $\{0, 1, 2, \ldots, n-1\}$, the values $(e^{2\pi i/n})^k = w_n^{-k}$ form the set $U(n)$ of the n roots of unity, which are spaced uniformly on the unit circle of the complex plane. This elucidates the known fact that the DFT of an n-point sequence corresponds to the uniform sampling of its Z-transform on the unit circle.

We now restate the cyclic convolution theorem in order to present the procedure for obtaining formulations for the hardware implementation of LSI-FIR systems:

Suppose that the sequence x is the input to the LSI-FIR system T_h. The output sequence y is given by

$$y = T_h(x) = h * x \tag{17}$$

Taking the discrete Fourier transform (DFT) to the above expression results in

$$F_n(T_h(x)) = F_n(h) \odot F_n(x) \tag{18}$$

or

$$(F_n \circ T_h)(x) = (F_n(h) \odot F_n)(x) \tag{19}$$

Since the choice of $x \in L(Z/n)$ was arbitrary, we obtain the following important result

$$F_n T_h = F_n(h) \odot F_n \tag{20}$$

or, emphasizing the diagonalization of T_h by the action of the DFT,

$$F_n T_h F_n^{-1} = F_n(h) \odot I_n \tag{21}$$

The expression $F_n(h) \odot I_n$ is denoted by $D_{F(h)} = D_{\hat{h}}$, where a matrix representation of $D_{\hat{h}}$ is given by

$$D_{\hat{h}} = \begin{bmatrix} \hat{h}_0 & & & & \\ & \hat{h}_1 & & & \\ & & \hat{h}_2 & & \\ & & & \ddots & \\ & & & & \hat{h}_{n-1} \end{bmatrix}, \quad \hat{h}_j = (F(h))(j), \quad j \in Z/n \tag{22}$$

Thus, we can write

$$T_h = F_n^{-1} D_{\hat{h}} F_n \tag{23}$$

This last expression serves as the basis for the formulations of LSI-FIR systems using FFT algorithms. This is accomplished by computing, both, the discrete Fourier transform and its inverse using fast algorithms.

To obtain formulations for hardware implementations of LSI-FIR filters, we proceed as follows:

1) Perform cyclic convolution operation by using fast Fourier transform (FFT) algorithms.

2) Use tensor products formulations of FFT algorithms to express the cyclic convolution operation.

3) Utilize the properties of the tensor products to manipulate the cyclic convolution expression with the purpose of improving possible hardware implementations.

We now provide some examples where we use tensor properties and the properties of LSI-FIR systems previously described in this work in order to present formulations for possible implementations of these systems.

Example 1:

Suppose that $h \in L(Z/n)$ is the impulse response of the LSI-FIR system T_h. If n is a composite of the form $n = r \cdot s$, we can factor the matrix $H = F_n^{-1} D_{\hat{h}} F_n$ representing the system T_h in a form that uses the Cooley-Tukey decimation in frequency (DIF) algorithm for the computation of the Fourier transform; and the Cooley-Tukey decimation in time (DIT) algorithm for the computation of its inverse:

$$H = (F_s^* \otimes I_r) T_{n,s}^* (I_s \otimes F_r^*) P_{n,s} D_{\hat{h}} P_{n,s}^{-1} (I_s \otimes F_r) T_{n,s} (F_s \otimes I_r)$$

where the asterisk (*) implies complex conjugation. We can now use the result presented on the chapter on permutation matrices and write

$$\tilde{D}_{\hat{h}} \equiv P_{n,s} D_{\hat{h}} P_{n,s}^{-1} = diag\left[P_{n,s} \cdot \hat{h}\right] = \begin{bmatrix} \hat{h}_0 & & & & \\ & \hat{h}_s & & & \\ & & \hat{h}_{2s} & & \\ & & & \ddots & \\ & & & & \hat{h}_{n-1} \end{bmatrix}$$

The factorization for H becomes

$$H = (F_s^* \otimes I_r) T_{n,s}^* (I_s \otimes F_r^*) \tilde{D}_{\hat{h}} (I_s \otimes F_r) T_{n,s} (F_s \otimes I_r)$$

where in this formulation the permutations have been eliminated.

Example 2:

Let H be an n-th order matrix representing a given LSI-FIR filter T_h. As we have seen in the previous section, H is a circulant matrix given in this case by

$$H = [h_{j,k}]_{0 \le j,k <}, \quad h_{j,k} = h(j-k)$$

For n a composite of the form $n = r \cdot s$, we can partition the matrix H so that we obtain a block circulant matrix with circulant blocks. This partition is always circulant [22]; furthermore, the block size can be chosen to be either r or s. We can

generalize Eq. 5 given in section 6.9 and write a partition of H into submatrices H_j, $0 \leq j < r$ of block size S:

$$H = \sum_{j \in Z/r} S_r^j \otimes H_j$$

The submatrices, as we have stated above, are circulant; hence, they may represent lower order LSI-FIR systems. Diagonalizing the shift operator S_r and the matrices H_j simultaneously produces the following result which also appears in Davis' book [22]:

$$H = \sum_{j \in Z/r} (F_r^{-1} D_{\widehat{S_{r,\delta}}} F_r) \otimes (F_s^{-1} D_{\widehat{h}}(j) F_s$$

where $D_{\widehat{S_{r,\delta}}}$ represents the diagonalized shift operator S_r, and $D_{\widehat{h}}(j)$ is a diagonal matrix of order s corresponding to the diagonalization of the circulants H_j. Using tensor products properties, we obtain

$$\sum_{j \in Z/r} (F^{-1} \otimes F_s^{-1}(D_{\widehat{S_{r,\delta}}} \otimes D_{\widehat{h}}(j))(F_r \otimes F_s)$$

and

$$H = (F_r \otimes F_s)^{-1}[\sum_{j \in Z/r} (D_{\widehat{S_{r,\delta}}} \otimes D_{\widehat{h}}(j))](F_r \otimes F_s)$$

Using, again, tensor products properties, we write

$$H = (F_r \otimes I_s)^{-1}(I_r \otimes F_s)^{-1}[\sum (D_{\widehat{S_{r,\delta}}} \otimes D_{\widehat{h}}(j)](F_r \otimes Is)(I_r \otimes F_s)$$

A user may find this formulation advantageous if efficient hardware implementations for the Fourier factors have been obtained.

Example 3:

We start this example by introducing the following function:

$$\mu^k = \sum_{r=0}^{r-1} \delta_{[r]}, \qquad \delta \in L(Z/n)$$

Thus, we have

$$\mu^0 = \delta, \qquad \mu^{(n-1)} = \{1, 1, 1, \ldots 1\} \equiv u$$

We term this function the projection function μ (making an analogy to projection operators). We notice that taking the Hadamard product of this function and any other function $f \in L(Z/n)$ results in the function f being null at the points $k \leq j < n$:

$$(\mu^k \otimes f)(j) \equiv (\mu^k(f)(j) = \begin{cases} f(j); & 0 \leq j < k \\ 0; & k \leq j < n \end{cases}$$

Applying the shift operator S_n to the function μ^k results in

$$S_n^m(\mu^k) = (\mu^k)_{[m]},$$

with

$$(\mu^k)_{[m]} \otimes f \equiv ((\mu^k)_{[m]})(f)$$

Thus,

$$((\mu^k)_{[m]})(f)(j) = \begin{cases} f(j); & m \leq j < m+k \\ 0; & otherwise \end{cases}$$

If the function $f = \hat{h}$ is the discrete Fourier transform of the impulse response sequence $h \in L(Z/n)$ of a given LSI-FIR filter T_h, then $((\mu^k)_{[m]})$ can be thought of as a filter, whose attributes (bandwidth, passband) are determined by the parameters k and m. Thus, $((\mu^k)_{[m]})$ can be a low-pass, band-pass, high-pass, all-pass, or band-reject filter; hence, this function allows us to perform filtering operations on the impulse response sequence h. This filtering operations manifest themselves as attributes of the diagonal matrix $D_{\hat{h}}$. We will show in the next example below how to use these attributes to simplify tensor products formulations of $H = F_n^{-1} D_{\hat{h}} F_n$.

Example 4. FAST LOW-PASS FILTER:

We will take advantage of the following fact which follows readily from our results on the chapter on permutation matrices:

For $n = r \cdot s$, we have

$$P_{n,s} \begin{bmatrix} \hat{h}_0 \\ \hat{h}_1 \\ \vdots \\ \hat{h}_{s-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \hat{h}_0 \\ 0 \\ \vdots \\ \hat{h}_1 \\ 0 \\ \vdots \\ \hat{h}_{s-1} \end{bmatrix}$$

This result allows us to make the following observation

Observation 1: "Rows of zeroes" will appear on

$$P_{n,s} \cdot diag \begin{bmatrix} \hat{h}_0 \\ \hat{h}_1 \\ \vdots \\ \hat{h}_{s-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The following fact follows:

$$\begin{bmatrix} \hat{h}_0 \\ \hat{h}_1 \\ \vdots \\ \hat{h}_{s-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}^T P_{n,s}^{-1} = \begin{bmatrix} \hat{h}_0 \\ 0 \\ \vdots \\ \hat{h}_1 \\ 0 \\ \vdots \\ \hat{h}_{s-1} \end{bmatrix}^T$$

which allows us to make the next observation.

Observation 2: **"Columns of zeroes" will appear on**

$$diag \begin{bmatrix} \hat{h}_0 \\ \hat{h}_1 \\ \vdots \\ \hat{h}_{s-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \cdot P_{n,s}^{-1} \equiv D_{\hat{h}} \cdot Pn, s^{-1}$$

Now, suppose we have the following tensor products formulation for H:

$$H = (F_s^* \otimes I_r) T_{n,s}^* (I_s \otimes F_r) P_{n,s} D_{\hat{h}} P_{n,s}^{-1} (I_s \otimes F_r) T_{n,s} (F_s \otimes I_r)$$

The "columns (rows) of zeroes" appearing on the matrices discussed above can be propagated throughout the above formulation in order to obtain a simplified expression. In this way, propagating the "columns of zeroes" to the right will eventually show that if $f \in L(Z/n)$ is the input vector, then the following decimated version is only required:

$$[f_0 \quad 0 \quad \dots \quad f_s \quad 0 \quad \dots \quad f_{2s} \quad 0 \quad \dots \quad f_{n-1}],$$

and, after propagating the "rows of zeroes" to the left, the following simplified version of the formulation is obtain where the most factors are reduced to "data assignment matrices" for the routing of the input data, and the relevant factor is the matrix $(F_s \otimes I_r)$. After being masked by the "columns of zeroes" (the factor $(F_s \otimes I_r)^*$

is masked by the "rows of zeroes"), the following simplification is obtained:

$$\begin{bmatrix}
1 & 0 & \dots & 0 & 1 & 0 & \dots & \dots & 1 & 0 & \dots & 0 \\
0 & 0 & \dots & 0 & 0 & 0 & \dots & \dots & 0 & 0 & \dots & 0 \\
\vdots & & & & & & & & & & & \\
1 & 0 & \dots & 0 & w_s & 0 & \dots & \dots & w_s^{s-1} & 0 & \dots & 0 \\
0 & 0 & \dots & 0 & 0 & 0 & \dots & \dots & 0 & 0 & \dots & 0 \\
\vdots & & & & & & & & & & & \\
1 & 0 & \dots & 0 & w_s^{s-1} & 0 & \dots & \dots & w_s & 0 & \dots & 0 \\
0 & 0 & \dots & 0 & 0 & 0 & \dots & \dots & 0 & 0 & \dots & 0 \\
\vdots & & & & & & & & & & & \\
0 & 0 & \dots & 0 & 0 & 0 & \dots & \dots & 0 & 0 & \dots & 0
\end{bmatrix}$$

this last matrix tells us that the fast low-pass filter is may be implemented by taking the discrete Fourier transform F_s on the decimated input data sequence f.

Example 5:

We begin this example by expressing the k-th term of the cyclic convolution $y = h * f$ in inner product form:

$$\begin{aligned}
y(k) &= \sum_{j \in Z/n} h_j (S_n^j)(k) = \sum_{j \in Z/n} h_j f(k-j) \\
&= \sum_{j \in Z/n} h_j f(-(j-k)) = \sum_{j \in Z/n} h_j f^{(-)}(j-k) \\
&= \sum_{j \in Z/n} (S_n^k f^{(-)})(k) = \sum_{j \in Z/n} h_j (f^{(-)})_{[k]}(j) = \langle (f^{(-)})_{[k]}, h^* \rangle
\end{aligned}$$

The inner product expression clearly shows the folding(hence, the name convolution) of the function f around the origin; this results in $f^{(-)}$ which is then shifted by k units.

Cyclic correlation, which was previously defined as

$$g(k) = \sum_{j \in Z/n} h_j f(j-k) \bmod n, \qquad k \in Z/n,$$

can also be written in inner product form, resulting in

$$g(k) = \langle f_{[k]}, h^* \rangle, \qquad k \in Z/n$$

As, we recall, the inner product $\langle f, h^* \rangle$ can tell us about the similarity (likeness) or dissimilarity (ambiguity) between the two signals f and h. For instance, if $\langle f, h^* \rangle = 0$, we may conclude that f and h are completely dissimilar. This conclusion, however, may be erroneous, since f and h may be similar up to a shifting operation

performed on one of the signals. This is why the operation $\langle f_{[k]}, h^* \rangle$ is a better tool for identifying similarity (hence, the name correlation) between the functions f and h.

If the signal h is the impulse response of a given LSI-FIR system T_h, then, as it is known, the operation $\langle f_{[k]}, h^* \rangle$ can be thought of as a detection operation performed on the incoming signal f. This is accomplished by allowing

$$f = R_n(h) = h^{(-)}$$

In this way, the convolution operation $y = h * f$ becomes

$$y(k) = \langle (f^{(-)})_{[k]}, h^* \rangle = \langle h_{[k]}, h^* \rangle, \qquad k \in Z/n$$

The signal f is said to be detected when $y(k)$, $k \in Z/n$ is a maximum. In practical applications [21], the input signal f is much longer than the filter T_h and the value k is used to indicate how much shifting was required (the identified delay) before the signal was detected. We present in the next example a simple detector which we call the fast tone correlator.

Example 6. FAST TONE CORRELATOR: For $n = r \cdot s$

Let $h \in L(Z/n)$ be the impulse response sequence of the filter T_h. Let its Fourier transform $\widehat{h} = F(h)$ be a decimated sequence of the form

$$\widehat{h} = [\widehat{h}_0 \quad 0 \quad 0 \quad \ldots \quad \widehat{h}_1 \quad 0 \quad 0 \quad \ldots \quad \widehat{h}_2 \quad 0 \quad 0 \quad \ldots \quad \widehat{h}_{r-1}]^T$$

satisfying the condition

$$P_{n,r}^{-1} \cdot \widehat{h} = [\widehat{h}_0 \quad \widehat{h}_1 \quad \widehat{h}_2 \quad \ldots \quad \widehat{h}_{r-1} \quad 0 \quad \ldots \quad 0]^T$$

Now, if we use the following tensor products formulation for $H = F_n^{-1} D_{\widehat{h}} F_n$

$$H = (F_s^* \otimes I_r) T_{n,s}^* (I_s \otimes F_r^*) P_{n,s} D_{\widehat{h}} P_{n,s}^{-1} (I_s \otimes F_r) T_{n,s} (F_s \otimes I_r)$$

we notice that the decimated sequence $\widehat{h}$ produces "rows of zeroes" to the right and "columns of zeroes" to left; furthermore, the effect of the permutation matrices is to gather these "rows of zeroes" ("columns of zeroes"). After the cancellation, the

following simplified formulation results:

$$\begin{bmatrix} I_s & 0 & \dots & 0 \\ I_s & 0 & \dots & 0 \\ \vdots & & & \\ I_s & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} D^0_{n,r} & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix} \begin{bmatrix} F^*_r & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix} P_{n,s} D_{\widehat{h}} P^{-1}_{n,s} \cdot$$
$$\cdot \begin{bmatrix} F_r & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix} \begin{bmatrix} D^0_{n,r} & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix} \begin{bmatrix} I_r & I_r & \dots & I_r \\ 0 & 0 & \dots & 0 \\ \vdots & & & \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

This result agrees with the well known result that decimation and periodization of signals are related through the discrete Fourier transform. The simplified formulation states that a pre-addition, followed by forward and inverse DFT F_r, followed by a data assignment operation, results in the fast tone correlator. This example may aid the computational task of some engineering applications [23].

We used the formulation

$$H = (F^*_s \otimes I_r) T^*_{n,s} (I_s \otimes F^*_r) P_{n,s} D_{\widehat{h}} P^{-1}_{n,s} (I_s \otimes F_r) T_{n,s} (F_s \otimes I_r)$$

for most of our examples above. Other simplifications may be obtained by using other tensor products formulations of the additive FFT algorithms as presented in chapter IV.

Conclusion:

Below, we present the objectives which we set out to accomplish in this work. We feel we were successful in accomplishing these tasks, setting up the stage for further work on software and hardware implementations of additive fast Fourier transform algorithms, and fast digital signal processing software and hardware implementations using LSI-FIR filters:

1) To present tensor products formulations of commonly known additive fast Fourier transform (FFT) algorithms.

2) To describe and provide guidelines for computer implementation of FFT algorithms, using tensor products language as an analytical tool to accomplish this task.

3) To present a mathematical characterization of linear shift invariant, finite impulse response (LSI-FIR) systems.

4) To use the tensor products language as a tool to aid in the implementation of LSI-FIR systems using FFT algorithms.

REFERENCES:

[1] Tolimieri, R. , "Multiplicative Characters and the Discrete Fourier Transform," *Adv. in Appl. Math*, vol. 7, No. 3, Academic Press Inc., 1986, pp. 344-380.

[2] Tolimieri, R., "The Algebra of the Finite Fourier Transform and Coding Theory," *Trans. Amer. Math. Society*, vol. 287, No. 1, January 1985, pp. 253-273.

[3] Tolimieri, R., "The Construction of Orthonormal Bases Diagonalizing the Discrete Fourier Transform," *Advances in Appl. Math.*, vol. 5, 1984, pp. 56-86.

[4] Auslander, L., Tolimieri, R., "Is Computing the Finite Fourier Transform Pure or Applied Mathematics?," *Bull. Amer. Math. Society (N.S.)*, vol. 1, 1979, pp.847-897.

[5] Cooley, J. W., Tukey, J.W., "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comp.*, vol. 19, 1965, pp. 297-301.

[6] Pease, M.C., "An Adaptation of the Fast Fourier Transform for Parallel Processing," *J. ACM*, vol. 15, 1968, pp. 252-264.

[7] Corinthios, M.J., "A Fast Fourier Transform for High-Speed Signal Processing," *IEEE Trans. Computers*, C-20, vol. 8, 1971, pp. 843-846.

[8] Temperton, C., "Self-Sorting Mixed-Radix Fast Fourier Transforms," *Journal of Computational Physics*, vol. 52, 1983, pp. 1-23.

[9] Blahut, R.E., *Fast Algorithms for Digital Signal Proceesing*, Addison Wesley Publishing Company, 1985, pp. 283-321.

[10] Kahaner D., "Matrix Description of the Fast Fourier Transform," *IEEE Trans. Audio Electroacoust.*, vol. 18, 1970, pp. 442-450.

[11] Singleton, R.C., "On Computing the Fast Fourier Transform," *J. ACM*, vol. 10, 1967, pp. 647-654.

[12] Gentleman, W.M., Sande, G., "Fast Fourier Transforms- For Fun and

Profit," *Proc. AFIPS, Joint Computer Conference*, vol. 29, 1966, pp. 563-578,

[13] Korn, D.G., Lambiotte, J.J., "Computing the Fast Fourier Transform on a Vector Computer," *Math. Comp.*, vol. 33, 1979, pp. 977-992.

[14] Cochran, W.T., et al., "What is the Fast Fourier Transform?," *IEEE Trans. Audio Electroacoust.*, vol. 15, 1967, pp. 45-55.

[15] Agarwal,R.C., W. Cooley, J.W., "An Efficient Vector Implementation of the FFT Algorithm on IBM 3090VF," *ICASSP 86*, 1986, pp. 249-252.

[16] Henrici, P., "Fast Fourier Methods in Computational Complex Analysis," *SIAM REVIEW*, vol. 21 , No. 4, Oct. 1979 pp. 481-527

[17] Kwan, H.K, Tsim, M.T., "High Speed 1-D FIR Digital Filtering Architectures Using Polynomial Convolution," *ICASSP 87*, pp. 1863-1866.

[18] Winograd, S., "Arithmetic Complexity of Computations," *CBMS-NSF* Regional Conference Series in Applied Mathematics, *SIAM*, 1980.

[19] Whelchel, J.E. Jr., Guinn, D.F., "FFT Organizations for High-Speed Digital Filtering," *IEEE Trans. Audio Electroacous.*, vol. AU-18, 1970, pp. 159-170.

[20] Bruun, G., "Z-Transform DFT Filters and FFT's," *IEEE Trans. ASSP*, vol. 26, No. 1, Feb. 1978, pp. 56-63.

[21] Echard, J.D., Broostyn, R.R., "Digital Filtering for Radar Signal Processing Applications ," *IEEE Trans. Audio Electroacoust.*vol. AU-20, No. 1, March 1972, pp. 42-52.

[22] Davis, P.J., *Circulant Matrices*, John Wiley & Sons, New York, 1979 .

[23] Stein, S., "Algorithms for Ambiguity Function Processing," *IEEE ASSP*, vol. ASSP-29, No. 3, June 1981, pp. 588-599

LITERATURE CONSULTED

Adams, J.W., "A new FFT Approach To The Interpolation of Discrete-Time Signals," *ICASSP 86*, **Tokyo, pp. 213-215.**

Alliney, S., *Linear Operators and Discrete Transforms, Signal Processing II: Theories and Applications*, **H. W. Schüssler (editor), EURASIP, 1983, Elsevier Science Publishers B. V. (North Holland).**

Andrews, H , Caspari, K, "A Generalized Technique for Spectral Analysis," *IEEE Trans. Comput.*, **vol. C-19, 1970, pp. 16-25.**

Andrews, H.C., Kane, J., "Kronecker Matrices, Computer Implementation, and Generalized Spectra," *Journal of ACM*, **vol. 17, 1970, pp. 260-268.**

Arazi, B., "Two-dimensional Digital Processing of One-dimensional Signal," *IEEE Trans. ASSP*, **vol. 2, 1974, pp. 81-86.**

Bergland, G., "A Guided Tour of the Fast Fourier Transform," *IEEE Spectrum*, **1969, pp. 41-52.**

Bringham, E.O., "The Fast Fourier Transform ," *IEEE Spectrum*, **1967, pp. 63-70.**

Bruce, J.D., "Discrete Fourier Transforms, Linear Filters, and Spectrum Weighting," *IEEE Trans. on Audio and Electroacoust.*, **1968, pp. 495-499.**

Buijs, H.L., Pomerlau, A., Fournier, M., Tam, W.G., "Implementation of A Fast Fourier Transform (FFT) for Image Processing Applications," *IEEE Trans. ASSP*, **vol. 22, 1974, pp. 420-424.**

Conrinthios, M.J., "A Parallel Radix-4 Fast Fourier Transform Computer," *IEEE Trans. Comput.*, **vol. C-24, 1975, pp. 80-92.**

Cooley, J.W., Lewis, P.A.W., Welch, P.D., "Historical Notes On The Fast Fourier Transform ," *IEEE Trans. Audio Electroacoust.*, **vol. AU-15, 1967, pp. 76-79.**

De Boor, C., "Efficient Computer Manipulation of Tensor Products," *ACM Trans. on Math. Soft.*, **vol. 5, 1979, pp. 173-182.**

Drubin, M., "Kronecker Product Factorization of FFT Matrix," *IEEE Trans. on Comput.* , **1971, pp. 590-593.**

Glassman, J.A., "A generalization of the Fast Fourier Transform," *IEEE Trans. on Comput.*, **vol. C-19, 1970, pp. 105-116.**

Harris, F.J., "The discrete Fourier Transform Applied to Time Domain Signal Processing," *IEEE Trans. Comm. Society Mag.*, **vol. 20, May 1982.**

Ritter, G.X., Gader, P.D., "Algebra Techniques for Parallel Image Processing," *Journal of Parallel and Dist. Comp.*, **vol. 4, 1987, pp. 8-44.**

Rose, D.J., "Matrix Identities Of The Fast Fourier Transform," *Linear Algebra Appl.*, **vol. 29, 1980, pp. 423-443.**

Singleton, R.C., "An Algorithm for Computing the Mixed-Radix Fast Fourier Transform," *IEEE Trans. Audio Electroacoust.*, **vol, 17, 1969, pp. 93-103.**

Stockham, S.G. Jr., "High-Speed Convolution and Correlation," *AFIPS Proc.*, **Spring Joint Conf., 1966, vol. 28, pp. 229-233.**

Swartztrauber, P.N., "FFT Algorithms for Vector Computers," *Parallel Computing*, **vol. 1, North Holland, 1984, pp. 45-63.**

Theilheimer, F., "A Matrix Version of the Fast Fourier Transform," *IEEE Trans. Audio Electroacoust.*, **vol. 19, 1969, pp. 158-161.**

Uhrich, M.J., "Fast Fourier Transform without Sorting," *IEEE Trans. Audio Electroacoust.*, **vol. 17, 1969, pp. 170-172.**

Veenkant, R.L., "A Serial Minded FFT," *IEEE Trans. Audio Electroacous.*, **vol. 20, 1972, pp. 180-185.**