VLSI COMPLEXITY OF PARALLEL FOURIER TRANSFORM ALGORITHMS

BY

TARANEH BARADARAN SEYED

Bachelor of Science
Aryamehr University of Technology
Tehran, Iran
1976

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1981

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 1989

VLSI COMPLEXITY OF PARALLEL FOURIER

TRANSFORM ALGORITHMS

Thesis Approved:

[illegible]

Thesis Advisor

[illegible]

[illegible]

[illegible]

Norman N. Durham

Dean of the Graduate College

ACKNOWLEDGEMENT

I would like to thank my major advisor Dr. Louis G. Johnson for his graceful attitude during the hardship, his thoughtful advice, and support and encouragement. I deeply owe the acheivements in my life to my husband Bijan who never compromised his love and support. My outmost gratitude to Dr. Donald Fisher for continous support and inspiration during my years at OSU. I would also like to thank the members of my advisory committee Dr. Hedrick, Dr. George, and Dr. Teague for helpful advice and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

The Discrete Fourier Transform (DFT) is one of the most widely used transforms in signal processing, image processing and computer vision. The computationality of Fourier transform however has always been one of the major issues for users and researchers. The time complexity of the sequential algorithm for calculating the discrete Fourier transform of an N-element signal is proportional to N^2. Cooley [Cooley 65] proposed a faster method for serial computation of the discrete Fourier transform known as Fast Fourier Transform or FFT. The FFT algorithm is based on the divide-and-conquer paradigm and reduces the time complexity of the computation of an N-element sequence to NlogN.

Over the last two decades a large body of knowledge has been dedicated to the processing aspects of DFT and FFT. Researchers have persued different approaches and techniques to increase the speed of the computation of the Fourier transform. Since the speed of serial computation of the Fourier transform is machine dependent, many researchers aimed at designing dedicated hardware [Gold 73], [Despain 79]. Reducing the computation time by employing faster circuitry for basic operations used in Fourier transform, namely, complex addition and multiplication is another approach persued by Taylor [Taylor 85], Cozzens [Cozzens 85], and Troung [Troung 86].

Significant advances in VLSI in recent years, which started a new era in massive parallelism, has opened yet another line of research in parallel processing of the Fourier transform [Thompson 80], [Despain 79], [Stone 71], [Bongiovanni 83], [Zhang 84], [Wold 84], [Gertner 87], and [Troung 88].

The serial and parallel processing of Fourier transform have been studied in two different contexts, namely, direct Fourier transform and fast Fourier transform. Each method offers certain degree of freedom and suffers certain restrictions on the size of the sequence, form of the input/output, and the size and capability of the basic computational units and the interconnections between them.

The area-time trade-off as a common measure for algorithmic and VLSI complexity has been used to approximate the goodness of the algorithms and their associated VLSI implementation. Thompson [Thompson 80] has derived a set of assumptions and rules for the analysis of the VLSI complexity of circuits. He has also derived a lower bound for the area-time complexity of VLSI design of circuits which solve an N-element Fourier transform problem.

In Chapter II, a detailed description of the techniques used to calculate an N-element Fourier transform is presented. Chapter III presents a set of new methods proposed by this study. In Chater IV the set of assumptions and rules used in the analysis of VLSI complexity of circuits are presented. Also different area-time measures and their applicability are discussed. Based on these assumptions, different layouts for the basic circuitry used in the VLSI design of Fourier transform solving circuitry and their area and time complexity are analyzed. And finally, an in depth analysis of the VLSI complexity of all methods under different area-time measures is presented.

CHAPTER II

FOURIER TRANSFORM COMPUTATION

Fourier transform computation is based on two different paradigms, the Discrete Fourier transform and the fast Fourier transform. This chapter is dedicated to the description of these methods and the detailed explanation of the parallel techniques to implement them.

Discrete Fourier Transform

The Direct Fourier transform of a finite sequence $\{x(n), \quad n=0,1,...,N-1\}$ may be expressed as $\{Y(n), n=0,1,...,N-1\}$ where

$$Y(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \qquad 0 <= k <= N-1 \qquad (2.1)$$

and $W_N = EXP\ (-2\pi j/N)$. DFT may also be represented in a matrix-vector multiplication form as depicted in Figure 1. The direct computation of Fourier transforms using matrix-vector multiplication has a time complexity of $O(N^2)$.

Fast Fourier Transform

Cooley [Cooley 65] has reformulated equation 2.1 and proposed a new method based on the divide-and-conquer para-

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ \vdots \\ Y_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W & W^2 & & W^{N-1} \\ 1 & W^2 & W^4 & & W^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & W^{N-1} & W^{2(N-1)} & & W^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

Figure 1. Matrix-Vector Representation of DFT.

digm. This method is called the Fast Fourier Transform or FFT. The FFT is based on the decomposition of the DFT into successively smaller DFTs. Algorithms based on the decomposition of {x(n)} are called decimation-in-time algorithms [Oppenheim 75].

The principle of decimation-in-time algorithms is best presented by considering the special case of $N=2^m$. Since N is an even number { x(n) } may be separated into two N/2-point sequences consisting of even-numbered elements in { x(n) } and odd-numbered elements in { x(n) }. Equation 2.1 is thus rewritten as:

$$Y(k) = \sum_{n \text{ even}} x(n)\, W_N^{nk} + \sum_{n \text{ odd}} x(n)\, W_N^{nk} \qquad 0 <= k <= N-1$$

Then by exploiting the characteristics of W_N^{nk}, the equation may be rewritten as

$$\begin{aligned} Y(k) &= \sum_{r=0}^{N/2-1} x(2r)\, W_{N/2}^{rk} + W_N^{k} \sum_{r=0}^{N/2-1} x(2r+1)\, W_{N/2}^{rk} \qquad (2.2) \\ &= G(k) + W_N^{k}\, H(k) \end{aligned}$$

where G(k) is the N/2-point DFT of the even-numbered points of {x(n)} and H(k) is the N/2-point DFT of the odd-numbered points of {x(n)}.

Therefore, the computation of an N-point DFT may be decomposed into two computations of N/2-point DFT. If $N = 2^m$, then the decomposition may further be applied m times until a simple 2-point DFT is reached. The 2-point DFT known

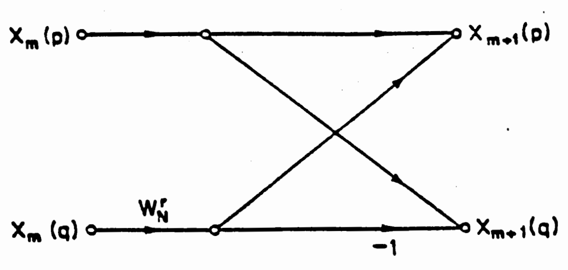


Figure 2. Two-point Butterfly

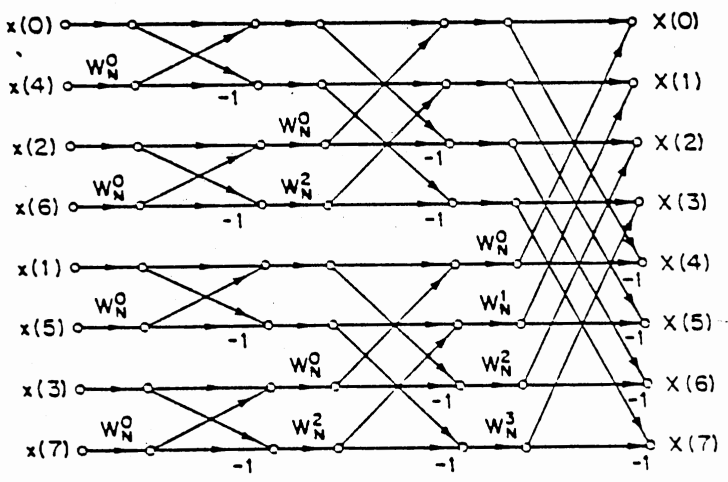


Figure 3. Flow Graph of 8-point FFT Using the Butterfly of Figure 2

as a butterfly computation is presented in Figure 2. The results of 2-point DFTs are then combined to calculate the 4, 8, ..., N point DFTs. The complete flow graph of an 8-point DFT using decimation-in-time algorithm is presented in Figure 3 [Oppenheim 75]. The computation is completed after logN (base 2) steps and each step requires N/2 butterfly operations. Therefore, a complete FFT computation of an N-point FFT has (N/2)logN computational steps which yeilds an O(NlogN) time complexity. The input data appears in bit-reversed order, but the output is in natural order. Oppenheim [Oppenheim 75] presents alternative formulations of decimation-in-time algorithms.

Another class of FFT algorithms, namely, decimation-in-frequency algorithms are based on the decomposition of {Y(n)} in the same manner. Figure 4 represents the butterfly operation for decimation-in-frequency algorithm. Figure 5 represents a complete flow graph of the computation for an 8-point sequence using decimation-in-frequency algorithm.

Recent advances in massive parallel processing promises faster computation of Fourier transforms. The following two sections are dedicated to the detailed description of parallel techniques to evaluate the Fourier transform of an N-element sequence based on the DFT and FFT respectively.

Parallel Processing of a DFT

The parallel computation of a DFT is mainly based on Equation 2.1. The symmetrical nature of the matrix of the coefficients shown in Figure 1 and the relationship between

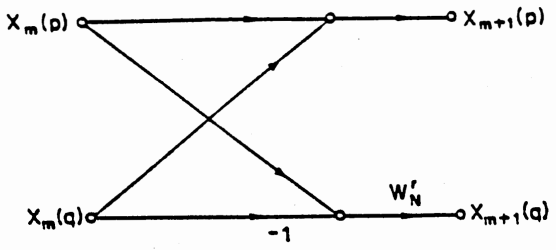


Figure 4. Butterfly for Decimation-in-Frequency

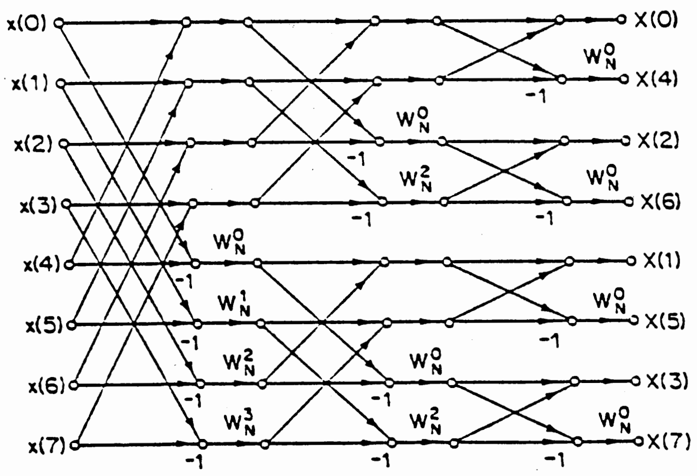


Figure 5. Flow Graph of Complete Decimation-in- Frequency Decomposition.

the elements of this matrix inspired many parallel techniques [Kung 80], [Mead 80], [Zhang 84], [Thompson 80].

The difference between these approaches stems from the assumptions on the size of the sequence and the number of processing elements used in the design. The detailed description of each technique is presented in the following section. The VLSI complexity of these techiques along with others are analyzed in Chapter IV.

N-cell DFT Pipelines

Kung and Leiserson [Mead 80] were the first to propose a pipeline of processing elements to compute the DFT of an N-element sequence. The pipeline has 2N-1 cells to compute the DFT of an N-element sequence {x(n)}. Cells operate on 50 percent duty cycle. The input sequence enters the pipeline from the leftmost cell with a 50 percent duty cycle. Zero-valued {Y(n)} enters the pipeline from the rightmost cell again with a 50 percet duty cycle. No operation is performed for the first N-1 cycles or until both x(0) and Y(0) reach the middle cell.

The middle cell is a special cell and it generates all the coefficients required by the transform. Coefficients are then propagated to both the right and the left cells. Register C in each cell is dedicated to the coefficient and is updated by the values that are generated by the middle cell and propagated by other cells. These values are labeled RA and RT.

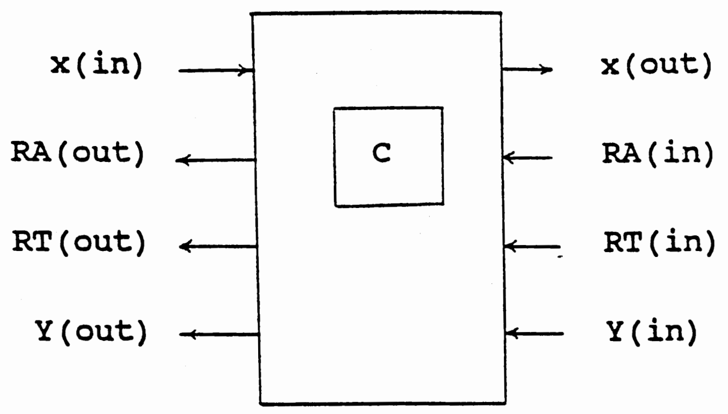
the elements of this matrix inspired many parallel techniques [Kung 80], [Mead 80], [Zhang 84], [Thompson 80].

The difference between these approaches stems from the assumptions on the size of the sequence and the number of processing elements used in the design. The detailed description of each technique is presented in the following section. The VLSI complexity of these techiques along with others are analyzed in Chapter IV.

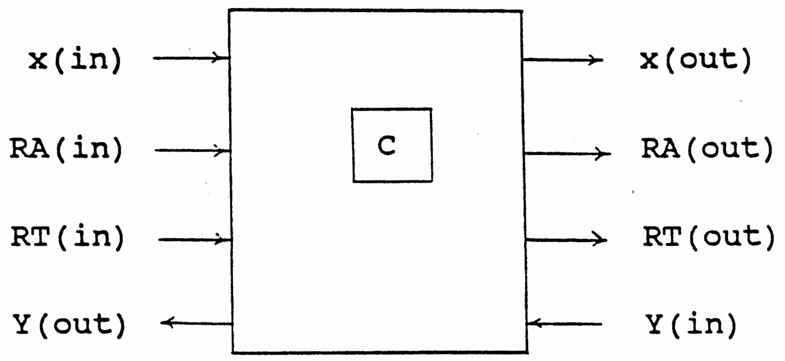
N-cell DFT Pipelines

Kung and Leiserson [Mead 80] were the first to propose a pipeline of processing elements to compute the DFT of an N-element sequence. The pipeline has 2N-1 cells to compute the DFT of an N-element sequence {x(n)}. Cells operate on 50 percent duty cycle. The input sequence enters the pipeline from the leftmost cell with a 50 percent duty cycle. Zero-valued {Y(n)} enters the pipeline from the rightmost cell again with a 50 percet duty cycle. No operation is performed for the first N-1 cycles or until both x(0) and Y(0) reach the middle cell.

The middle cell is a special cell and it generates all the coefficients required by the transform. Coefficients are then propagated to both the right and the left cells. Register C in each cell is dedicated to the coefficient and is updated by the values that are generated by the middle cell and propagated by other cells. These values are labeled RA and RT.



a) Structure of Left Cells

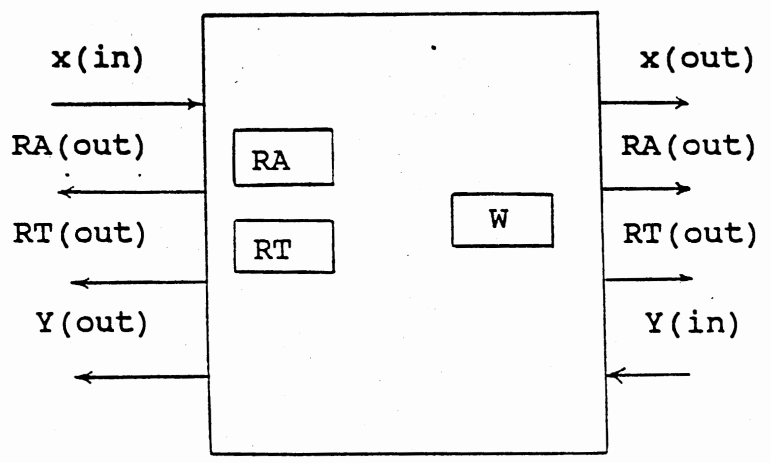


b) Structure of right cells

OPERATION

```
 Y(out):= Y(in) + C * x(in)
x(out) := x(in)
     C := RA(in) * RT(in)
RA(out) := RA(in)
RT(out) := RT(in)
```

Figure 6. The structure and Operation of right (a) and left (b) cells in 2N-1 Cell DFT Pipeline.



<u>OPERATION</u>

$$
\begin{aligned}
Y(out) &:= Y(in) + RA * x(in) \\
RA(out) &:= RA \\
RT(out) &:= RT \\
RA &:= RA * RT^{2} * W \\
RT &:= RT * W
\end{aligned}
$$

Figure 7. The Structure and Operation of the Middle Cell in a (2N-1) Cell DFT Pipeline.

When cell i is active, it receives Y from its right neighbor and x from its left neighbor and RA and RT from its left/right neighbor based on whether cell i is located at the right/left of the middle cell. Then it performs the multiply-add operation and sends Y to its left neighbor and x to its right neighbor.

The structure and the operation of cells are depicted in Figure 6. Figure 6a represents the structure of the cells to the left of the middle cell. Figure 6b represents the structure of the cells to the right of the middle cell.

The structure and operation of the middle cell is special and different from other cells. This cell generates and propagates the appropriate coefficients to all other cells. The structure and operation of the middle cell is depicted in Figure 7. Registers RA and RT in the middle cell are initially one and the register labeled W contains EXP(-2π j/N). A sequence of the operations of the pipeline for a 3-point DFT is depicted in Figure 8. Each cell is labeled with its current value of x, y, and C, the coefficient used in multiply-add operation is circled. After 2N-1 cycles through the pipeline, Y(0) is out, Y(1) will be out after 2 more cycles, and finally, it takes 2N-2 more steps after the compeletion of Y(0) to output Y(N-1). Therefore, 4N-3 cycles are required to compute an N-point DFT. Each step requires two multiplications and one addition.

There are 2N-2 simple cells, each containing a register C, a multiplier and an adder. The middle cell performs five multiplications and an addition at each step. A faster

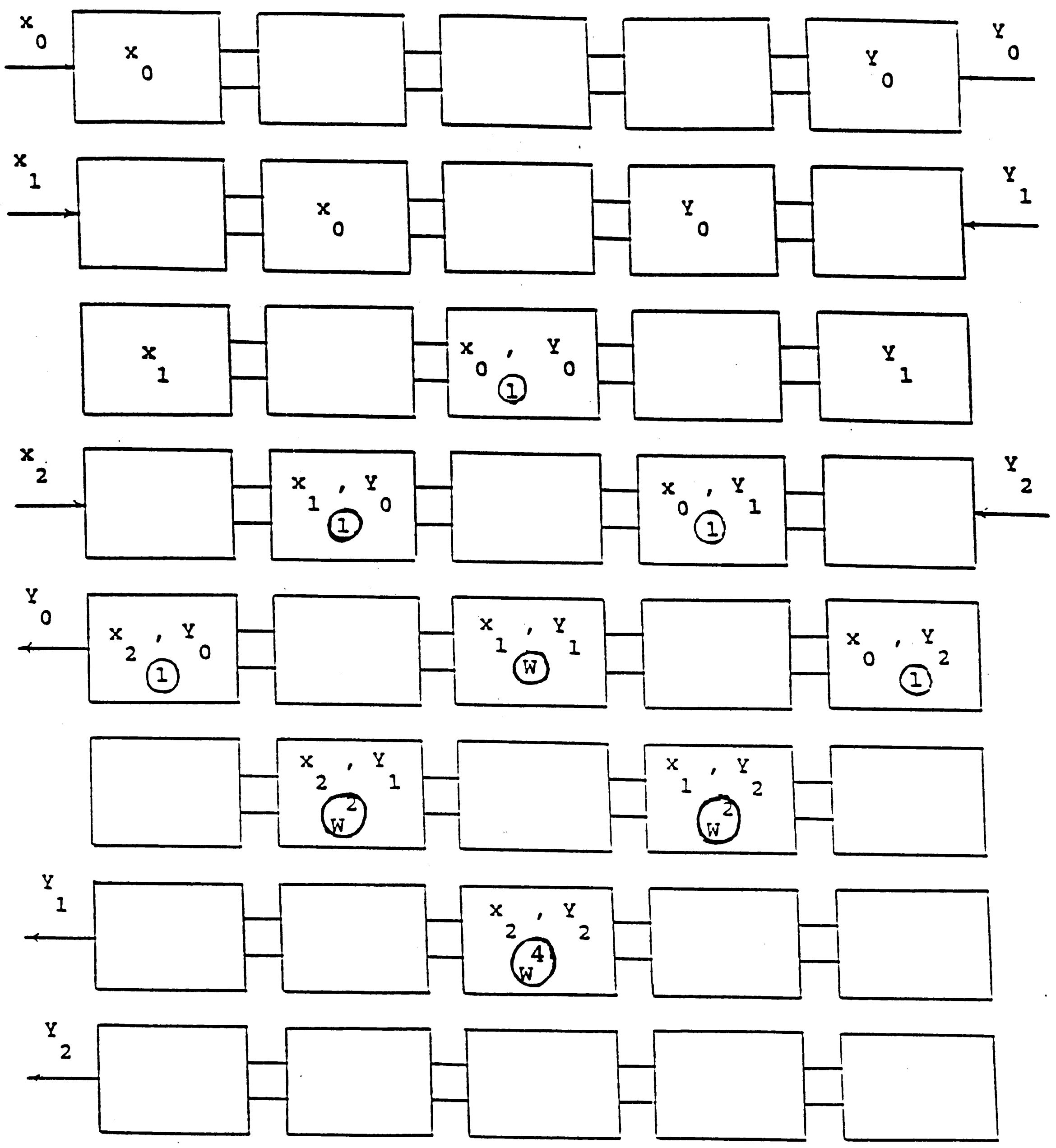


Figure 8. A Sequence of Operations of a (2N-1) Cell Pipeline for a 3-point DFT.

multiplier or several multipliers may be provided in the middle cell to avoid a slow down throughout the pipeline. Each cell requires four input and four output lines.

Basically, this design employs 2N-1 cells each containing a multiply-add circuitry and a register. This approach reaches a 4N-3 cycles of computation for an N-element DFT. The pipeline time (elapsed time between the input of two consecutive sequences) is also 4N-3 since a new computation may not start unless the previous computation is completed.

Kung [Kung 80] has proposed another linear pipeline implementation based on the recursive formulation of the DFT. Given the input sequence $\{x(n)\}$, the computation of the Fourier transform of the sequence, $\{Y(n)\}$, can be viewed as that of evaluating the polynomial

$$x(N-1)P^{N-1} + x(N-2)P^{N-2} + \dots + x(1)P + x(0)$$

At $P = 1, W, W^2, W^3, \dots, W^{N-1}$. The polynomial can be rewritten as:

$$(\dots((P\,x(N-1) + x(N-2))P + x(N-3))P + \dots + x(1))P + x(0)$$

The recurrence formula can be rewritten as follows:

$$Y^0(i) = x(N-1) \qquad 0 <= i <= N-1$$

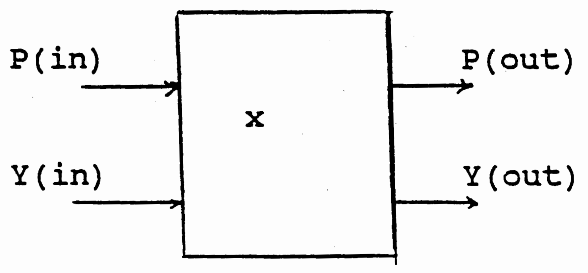
$$Y^{k}(i) = Y^{k-1}(i)\, P^{i} + x(N-1-k) \qquad 1 <= k <= N-1$$

Then $Y(i) = Y^{N-1}(i)$. The structure of a basic cell, its operation, and the structure of the pipeline is depicted in Figure 9.

The full pipeline consists of N-1 cells. The inputs Y(in) and P(in) to the leftmost cell are x(N-1) and some power of W respectively. Y(0) leaves the pipeline after N-1 cycles from the rightmost cell and Y(N-1) will leave the pipeline after N-2 more cycles. Therefore, the processing time of a sequence is 2N-3 cycles. A new sequence may start its load and computation after N cycles or immediately after Y(N-1) has left the leftmost cell. Therefore, the pipeline time is N.

The basic cells in the pipeline require a register and a multiply-add circuitry. It also needs two input and two output lines. Inter-cell connections are near-neighbor. A set of new linear pipeline approaches proposed in this research will appear in chapter III. The main advantage of linear approaches lie on the fact that the implementation is not restricted by the input size. In other words, for any given N, a linear pipeline could be constructed which solves the Fourier transform problem for any sequence {x(m)} given m <= N and assuming that appropriate P values are provided.

It is possible to decompose the DFT of a sequence {x(n)} to smaller DFTs and then combine the results into {Y(n)}. Gold and Bially [Gold 73] outlined a method to



P(out) := P(in)

Y(out) := Y(in) * P(in) + x

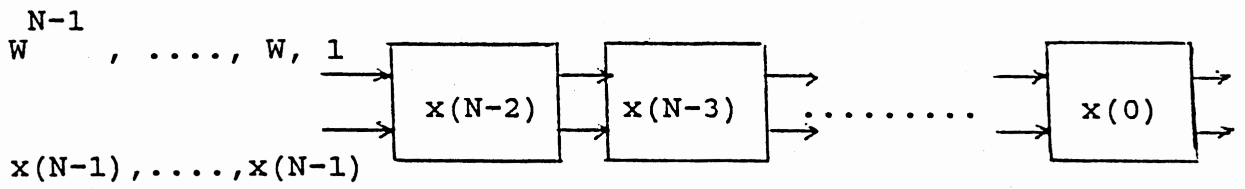


Figure 9. The Structure and Operation of Basic Cell and the Structure of the Pipeline for Recursive DFT.

factor a T-point sequence into a two-dimensional matrix. If T is a composite number, it can be factored into a product of integers. If T is a prime number, the original signal can usually be augmented with zeroes to obtain a composite number.

Let T = M x L be the number of elements in the input sequence. Then {x(n), n=0,1,...,T-1} may be rearranged into a two-dimensional matrix with M rows and L columns as shown in Figure 10a. The computational steps required to calculate the Fourier transform of a one-dimensional sequence rearranged into a matrix are as follows:

1) Calculate the DFT of each row individually, the kernel of these transforms is W_T^L
2) Multiply each term in the resultant matrix by W^{ij} where i and j are the row and column indices of each term respectively, $0 <= i <= M-1$, $0 <= j <= L-1$
3) Calculate the DFTs of each column individually, the kernel of this DFT is W_T^M

The resultant matrix is shown in Figure 10b.

Since steps 1 and 3 are performed in parallel, the computation time is proportional to M+L. Given T = MxL, M+L will be minimized if M=L or $M = T^{1/2}$. Therefore, the matrix arrangement of a one-dimensional signal is most effective when T is a complete square.

Zhang [Zhang 84] was the first to propose an N-cell

$$\begin{vmatrix} x(0) & x(M) & \cdots\cdots & x(M(L-1)) \\ x(1) & x(M+1) & \cdots\cdots & x(M(L-1)\ +1) \\ \vdots & \vdots & & \vdots \\ x(M-1) & x(2M-1) & \cdots\cdots & x(M(L-1)+M-1) \end{vmatrix}$$

(a)

$$\begin{vmatrix} Y(0) & Y(1) & \cdots\cdots & Y(L-1) \\ Y(L) & Y(L+1) & \cdots\cdots & Y(2L-1) \\ \vdots & \vdots & & \vdots \\ Y((M-1)L) & Y((M-1)L+1) & \cdots\cdots & Y((M-1)L + L-1) \end{vmatrix}$$

(b)

Figure 10. (a) Matrix arrangement of a one dimensional signal (b) resultant matrix.

mesh-connected network for DFT. His approach is based on the above formulation. Kung's recursive DFT is used to compute individual row and column DFTs.

The N-element sequence, $N = m^2$, is rearranged into an m x m matrix. The format of the rearranged input is shown in Figure 11a. Figure 11b represents the format of the output from the network. Neither the input nor the output have the natural form.

The basic cell in Kung's recursive formulation is modified to allow both row and column DFTs. The structure and operation of the basic cell is represented in Figure 12. A control signal S is used to distinguish column and row DFTs.

Process starts with calculating column DFTs. After m-1 cycles the first element of the Fourier transform is out. This output and the following outputs are multiplied by appropriate W^{ij} and shifted back to the same column to be used in row DFTs. Register x^* in the basic cell is used for shift operation. An extra cell is added to the end of each column pipeline as the last row of the matrix which is used to multiply the output of the pipeline by W^{ij} and shift it back to the pipeline.

After m more steps, column DFTs are complete and row DFTs may be initiated which will be complete after 2m-2 steps. The complete process requires 4m-2 cycles.

The complete structure of the m x m mesh-connected network is shown in Figure 13. The network has mxm basic cells and an extra row of cells to accomplish local

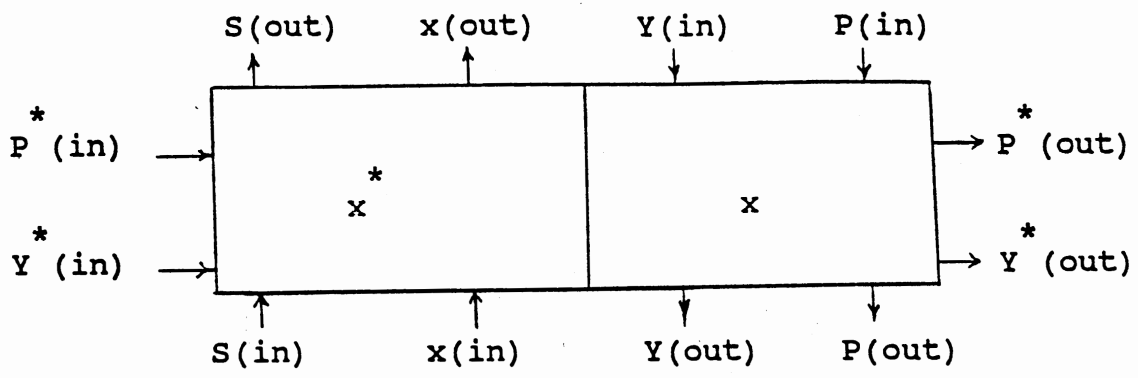
$$\begin{bmatrix} x(m-1,m-1) & x(m-1,m-2) & \cdots\cdots\cdots & x(m-1,0) \\ x(m-2,m-1) & x(m-2,m-2) & \cdots\cdots\cdots & x(m-2,0) \\ \vdots & \vdots & & \vdots \\ x(0,m-1) & x(0,m-2) & \cdots\cdots\cdots & x(0,0) \end{bmatrix}$$

(a) input

$$\begin{bmatrix} Y(0,m-1) & Y(0,m-2) & \cdots\cdots\cdots & Y(0,0) \\ Y(1,m-1) & Y(1,m-2) & \cdots\cdots\cdots & Y(1,0) \\ \vdots & \vdots & & \vdots \\ Y(m-1,m-1) & Y(m-1,m-2) & \cdots\cdots\cdots & Y(m-1,0) \end{bmatrix}$$

(b) output

Figure 11. The Input/Output Format for an N-element Mesh-connected DFT (Zhang).



S(in) = 1 (computing column DFTs and multiplying) (by W**(ij))

$x(out) = x^{*}$

$x^{*} := x(in)$

S(out):= S(in)

P(out):= P(in)

Y(out):= Y(in) * P(in) + x

S(in) = 0 (computing row DFTs)

$P^{*}(out) := P^{*}(in)$

$Y^{*}(out) := Y^{*}(in) * P^{*}(in) + x^{*}$

S(out):= S(in)

Figure 12. The Structure and Operation of the Basic Cell for N-cell Mesh-connected DFT.

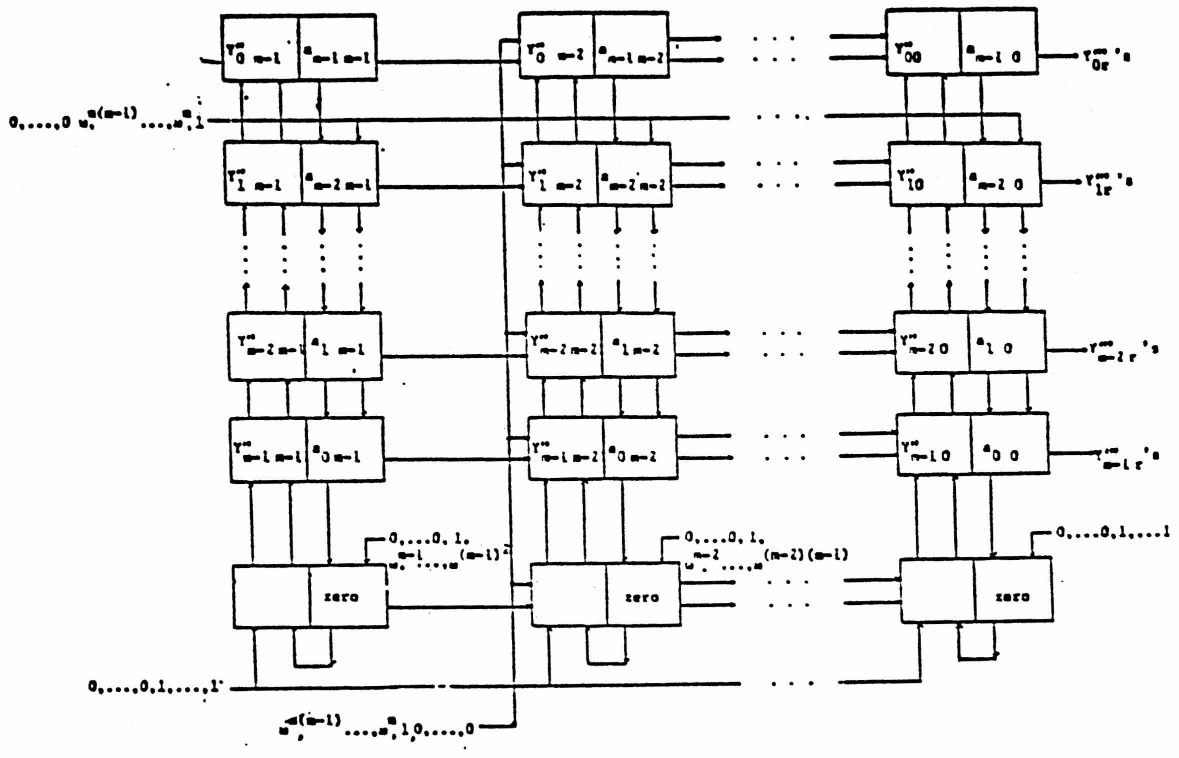


Figure 13. Mesh-connected DFT Network (Zhang).

multiplications. Two multipliers generate the appropriate Ps used in the recursive formulation of DFT for the column and row DFTs. The output of these multipliers are input to the cells in the second row of the network (for the column DFTs) and the second column of the network (for the row DFTs). M more multipliers are needed to generate the W^{ij}s, one for each column of cells. Therefore, the complete design requires N basic cells and $N^{1/2}+2$ multipliers. The basic cell as depicted in Figure 12 requires two registers x and x^*, and a multiply-add circuit. Each cell has 6 input and 6 output lines.

The design of the basic cell and the network as appeared in [Zhang 84] using a single local control signal S is incorrect. According to the specifications, the control signal S should remain 1 for 2m cycles to guarantee the correct calculation of the column DFTs and their shift back to the cells. Therefore, one-valued S must be input to the cells in the second row of the matrix from the first cycle until the last cycle of the column DFTs or for 2m consecutive cycles. At this point S must be changed to zero in all cells to inhibit any further shift and propagation of x^* (results of the column DFTs). Therefore, S must be a global signal (connected to all cells). Otherwise if S is set to zero, while directly connected to the last row of cells, other cells are still receiving the value of one and propagating the calculated x^*s which will result in a faulty outcome.

In summary, in order for this design to work correctly, S must be directly connected to all cells. The consequence of the direct connection of S to all cells is an increase in the area of the design which has a negative effect on its performance. The modified mesh-connected network with S directly connected to all cells is represented in Figure 14.

A new N-cell mesh-connected network using a different formulation and mechanism for row and column DFTs is presented in Chapter III as a part of this study.

N^2-Cell DFT Networks

Another approach to the calculation of DFT may employ a multitude of pipelines of N cells to boost the time efficiency by avoiding the recirculation of the intermediate results. Thompson [Thomson 83] refers to such a design. In this design the non-recursive linear pipeline proposed by Kung is used as the basic computational approach. 4N-3 basic pipelines constitute the complete design which unrolls the computation onto 4N-3 rows of 2N-1 cells. There are actually about $8N^2$ cells in this N^2-cell design. Thompson suggests the possibility of some reductions in the size which leaves $2N^2$ cells in the network. In Chapter III a new mechanism using only N^2 cells along with detailed description is presented.

The Fast Fourier transform is shown to calculate the Fourier transform more efficiently on a single processor system. Many authors have proposed hardware implementations consisting of many processing cells based on this algorithm.

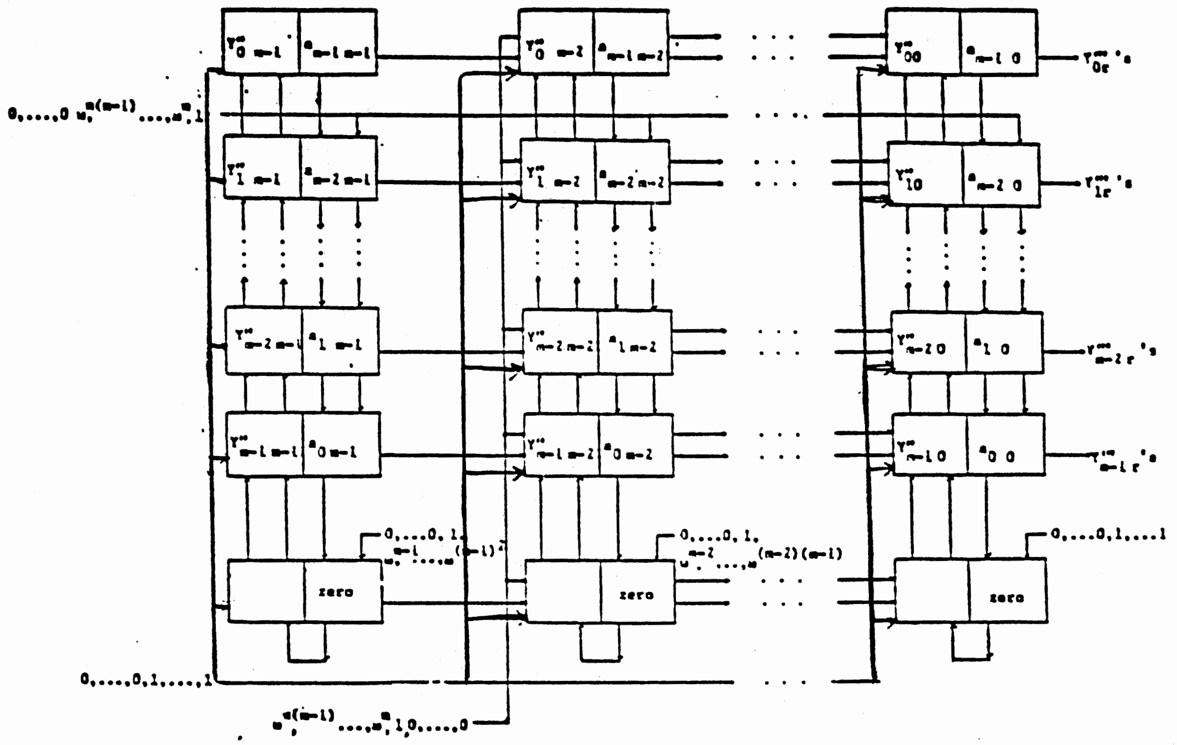


Figure 14. The Modified Design for Mesh-connected Network.

Fast Fourier Transform Networks

There are a number of versions of the FFT algorithm [Oppenheim 75]. The FFT algorithms mainly differ in the order of input, output and the coefficients used. Two algorithms, decimation-in-time and decimation-in-frequency are presented in this chapter. Either algorithm has a time complexity of O(NlogN) on a uni-processor system. Several dedicated, multi-cell designs based on the FFT are presented in the following sections.

Cascade Implementation

The cascade implementation was first proposed by Despain [Despain 79]. It is based on the decimation-in-frequency algorithm. The flow graph for an 8-point FFT based on decimation-in-frequency with inputs in normal order and outputs in bit-reversed order is presented in Figure 5. This flow graph is used to describe the details and the mechanism of the cascade implementation.

The cascade implementation consists of m = log N processing cells. Cells from left-to-right C(i), 1<= i <=m are connected to shift registers of length $2^{(m-i)}$. In other words, the left-most cell, C(1) is connected to a shift register of $2^{(m-1)}$ cells, where each cell is capable of holding an input datum or intermediate result. Cell C(2) or the second cell is connected to a shift register of length $2^{(m - 2)}$ and the last cell C(m) is connected to a shift register with one cell. Cells are arranged as a pipeline. The structure of the pipeline is depicted in Figure 15.

Each cell has two input and two output lines. The output line SR(in) is connected to the input of the associated shift register, while the other output line OUTPUT is connected to the next cell in the pipeline. One of the two input lines of each cell, SR(out), is connected to the output of the associated shift register while the other input line INPUT is connected to the output of the previous cell. INPUT to the leftmost cell is the line where the input signal enters the cascade and the OUTPUT line of the rightmost cell sends the results of the calculation out. Each cell saves the input values or the intermediate results until the associated input value or intermediate result is input to the cell for butterfly calculation. For example, in a case of 8-point FFT whose flow graph is represented in Figure 5, the leftmost cell C(1) inputs x(0), x(1), x(2), and x(3). and shifts them into the shift register of length 4. When x(4) appears on the INPUT line of C(1), its butterfly correspondent x(0) appears on SR(out), at this point the butterfly operation is performed. x(0)+x(4) is sent to the next cell while $(x(0)-x(4))W^{0}$ is sent to the shift register. In the following step, the butterfly operation is performed on x(1) and x(5) and x(1)+x(5) is sent to the next cell while (x(1)-x(5))W is sent to the shift register. The next step generates (x(2)+x(6)) and $(x(2)-x(6))\ W^{2}$. At this point the shift register of C(2) is full and SR(out) in C(2) has (x(0)+x(4)) where combined with

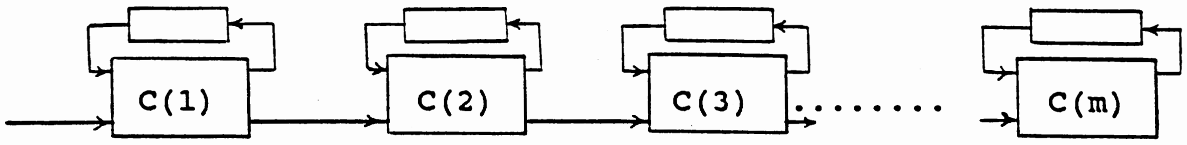
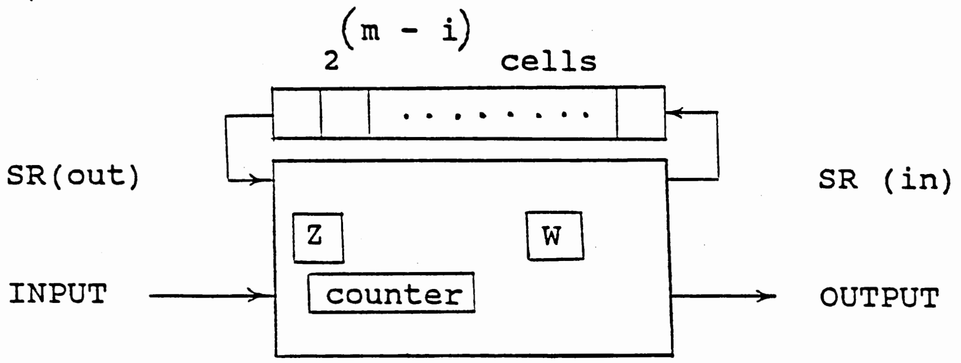


Figure 15. The Structure of the Pipeline for Cascade Implementation of FFT.



```
leftmost-bit of counter = 0
         SR(in)  :=  INPUT
        OUTPUT   :=  SR (out)


leftmost-bit of counter = 1
        OUTPUT   := SR(out)+ INPUT
        SR(in)   := (SR(out) - INPUT )* Z
```

Figure 16. The Structure and Operation of the ith Cell in Cascade Implementation of FFT.

(x(2)+x(6)) performs the topmost butterfly operation of the second column of Figure 5. During the next phase, the second butterfly from the top on the second column of Figure 5 is performed whose result combined with (x(0)+x(4)) + (x(2)+x(6)) stored in the single register of C(3) generates X(0) which leaves the system on the OUTPUT line of C(3).

Therefore, each cell C(i) performs $2^{(m-i)}$ butterfly operations every $2^{(m-i)}$ cycles. During the first $2^{(m-i)}$ cycles, cells accumulate the results of the previous cycles until their associated butterfly couples are generated, then during the following $2^{(m-i)}$ cycles the butterfly operations are performed. In general, C(1) performs the butterfly operations at the first column of Figure 5, C(2) performs those in the second column and C(3) performs the ones in the third column.

In order to keep track of cycles associated with each cell there is a (m-i)-bit counter. As long as the leftmost bit of the counter is zero, the cell simply accumulates its input values and sends the output of its shift register to the next cell. As soon as the leftmost bit of the counter becomes one and as long as it remains one ($2^{(m-i)}$ cycles) it performs the butterfly operations. The structure and operation of cell C(i) is presented in Figure 16.

One of the advantages of using a decimation-in-frequency algorithm is the ease of generating coefficients for the butterfly operations. Each cell has a register Z containing 1 initially, which is multiplied by W at every cycle. At the time of butterfly operation, this register contains the appropriate value.

A new sequence of input values may enter the cascade as soon as a slot becomes open in the shift register of the left-most cell or N cycles after the first element of the previous sequence has entered the leftmost cell. Therefore, the pipeline time of the cascade is proportional to N. The processing time of a sequence however is proportional to 2N. To this time the time needed to shift the intermediate results within the shift registers must be added. The total length of shift registers for an N-element implementation of cascade is N. The longest shift register, connected to C(1) contains N/2 cells. Therefore, the time required to shift N/2 elements must be added to the processing time of each computational step. The time required to convert the output sequence from bit-reversed order to normal order must also be added to the processing time of this design.

The cascade implementation of an FFT uses a pipeline of m=logN processing elements. However, the amount of memory used for the shift registers approaches N. Thus, although the number of processing cells is very small, the shift registers occupy an O(N) area.

The FFT Network

One of the methods to implement FFT in hardware is to lay out the flow graph of an FFT computation as depicted in Figure 3 in hardware providing a distinct cell for every butterfly computation. Consequently, the design consists of

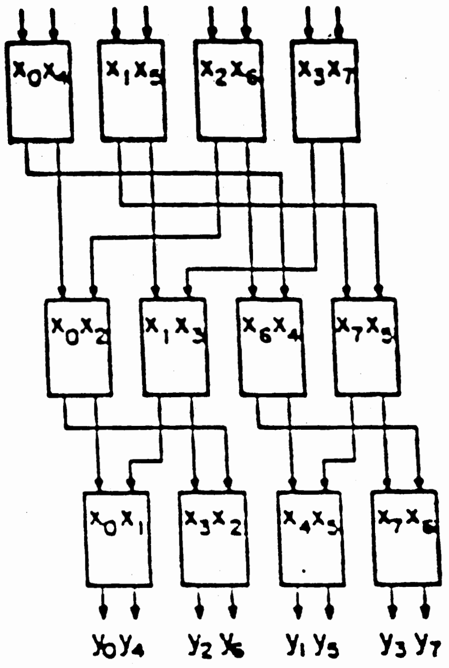


Figure 17. The FFT Network
[Thompson 83].

N/2 logN cells which could be laid out in logN rows of N/2 cells. Figure 17 adopted from [Thompson 83] represents the FFT network for N=8.

The input is in bit-shuffled order and the output is in bit-reversed order. This order seems to minimize the area required for row interconnections [Thompson 83]. Each cell performs a butterfly operation on its two inputs and coefficients are stored in a register in the cell. The basic structure and operation of an FFT network cell in the mth row is depicted in Figure 18. Additional circuitry is required to convert the input from normal order to bit-shuffled and to convert the output from bit-reversed to normal order.

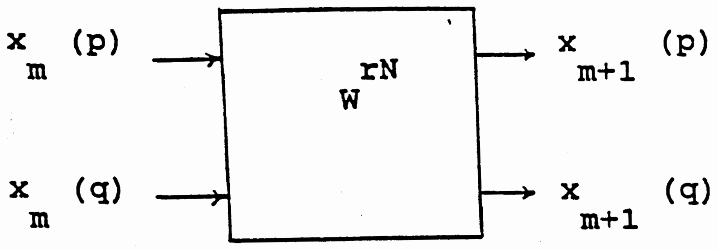
The interconnection between consecutive rows occupy wire areas. Thompson [Thompson 83] shows that in general, the connections emerging from the Kth row 0<= K <= logN-1, occupy $N/(2^{K+1})$ tracks. Consequently, a total of N-1 horizontal tracks are required to lay out the inter-row connections.

The processing time of the FFT network is logN cycles. To this time the time for inter row data transmission must be added. The pipeline time performance of the FFT network is one cycle, since a new problem can enter the network immediately after the previous problem leaves the first row.

The Perfect-Shuffle Implementation of FFT

Stone [Stone 71] was the first to point out the

$$p-q = 2^{m-1}$$



$$x_{m+1}(p) := x_m(p) + W^{rN} x_m(q)$$
$$x_{m+1}(q) := x_m(p) - W^{rN} x_m(q)$$

Figure 18. The Structure and Operation of the Basic Cell in the mth level of the FFT Network.

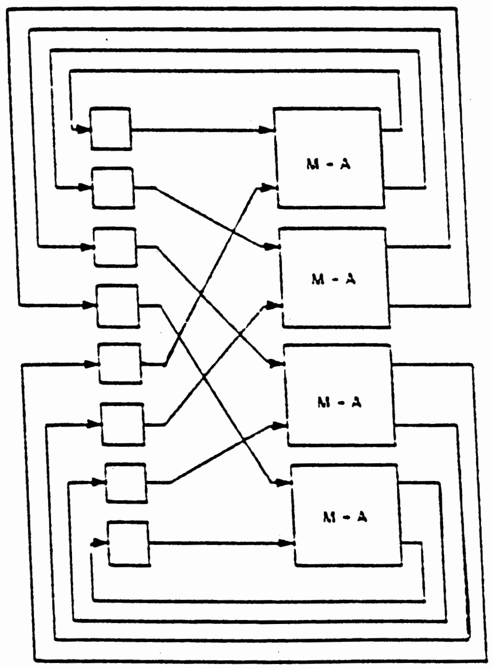


Figure 19. A Shuffle-Exchange Network for 8-point FFT (Stone 71).

importance of the perfect-shuffle connection in multi-processing systems. One of the most important applications of the perfect-shuffle network is in the implementation of the decimation-in-time algorithm for FFT. Figure 19 represents the general form of the perfect-shuffle network for an 8-point FFT.

The input to the perfect-shuffle network is in bit-shuffled order and the output is in bit-reversed order requiring extra hardware to rearrange the input and output to the normal order. The network consists of N/2 cells, each designed to perform a butterfly operation. The flow graph of FFT computation for perfect-shuffle arrangement is presented in Figure 20. The flow of inputs through the network is represented in Figure 21.

The perfect-shuffle network consists of N/2 cells each capable of performing a butterfly operation. LogN coefficients for butterfly operations, one for each stage of the computation are stored in each cell. As it is shown in Figure 21, the connection between cells is regular and unlike the FFT network it is independent of the stage of the computation. Data items circulate through N/2 cells for logN cycles before the output is ready. Therefore, the processing time of the FFT of an N-element sequence $N=2^m$, is proportional to m=log N. The time required for data recirculation must be added to this time to reflect the actual processing time. The problem of data recirculation and associated area and time complexity is addressed in chapter IV. The pipeline time for perfect-shuffle the FFT

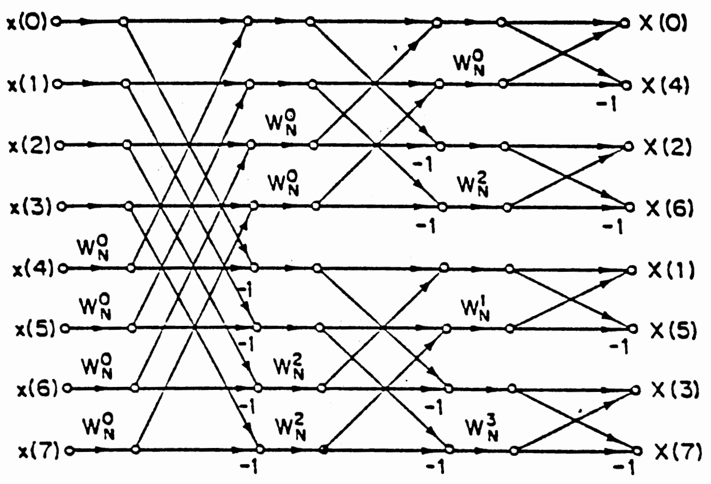


Figure 20. The Flow Diagram of FFT Algorithm for an 8-point Signal with Input in Normal Order and Output in Bit-reversed Order.

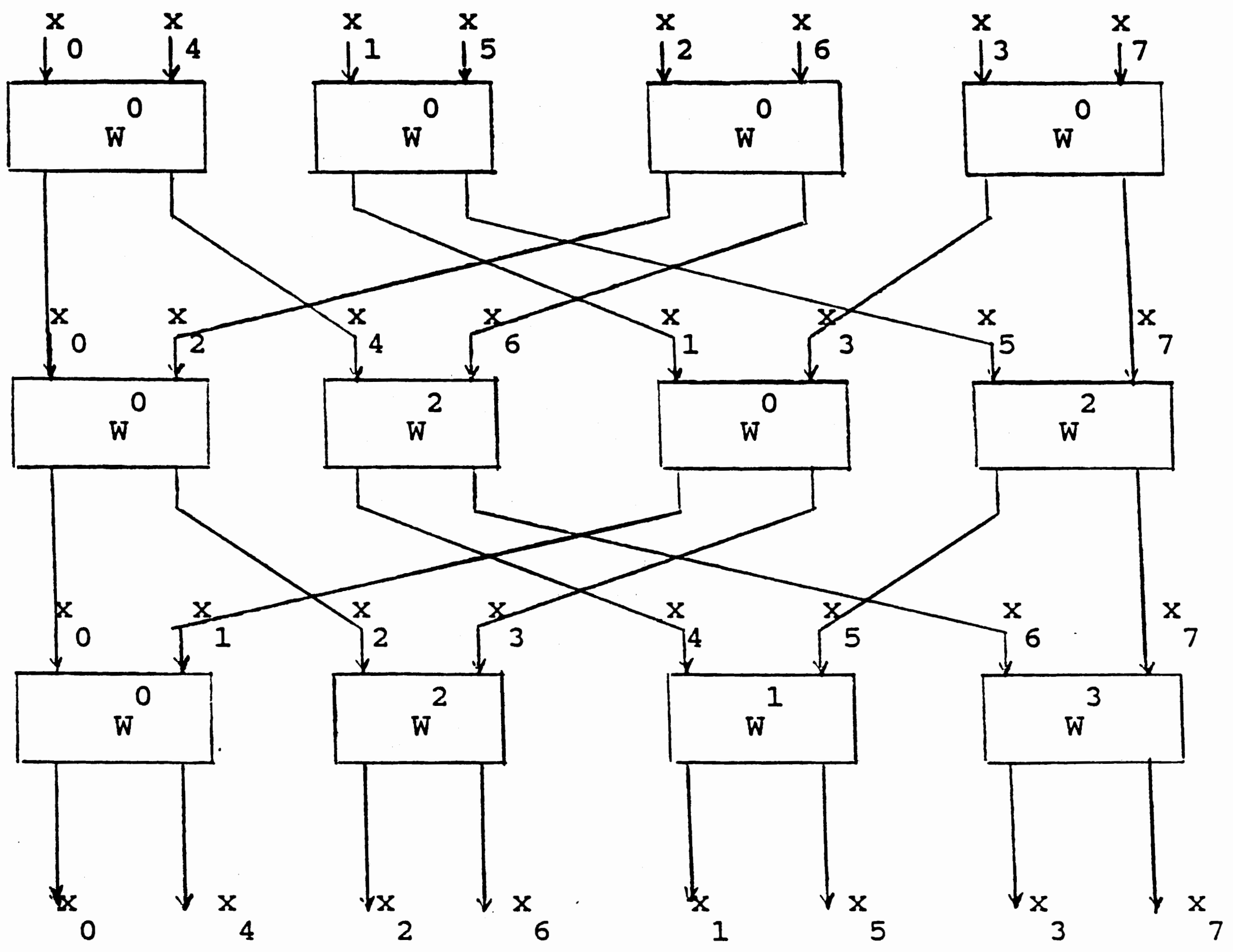


Figure 21. The Input Flow Through Perfect-Shuffle Network.

network is equal to its processing time since new sequence can enter the network only after the computation of the current sequence is completed.

The Mesh Implementation of the FFT

The mesh implementation of an FFT was first proposed by stevens [Stevens 71] for the ILLIAC IV architecture. Assuming $N=2^{2m}$ and using the decimation-in-frequency algorithm, the N-element sequence is arranged into a mesh structure of $2^m x 2^m$ elements in a row-major order. Figure 22 represents a mesh arrangement of a 16-point sequence. In order to simplify the description of the mesh implementation of the FFT algorithm, the data flow will be shown on a 16-point FFT network given in Figure 23.

As it is shown in Figure 23, at each stage of the computation N/2 butterfly operations are performed. At the first stage, the butterfly operations are performed on x(0) and x(8), x(1) and x(9), ..., and x(7) and x(15). Cell 0 receives x(8) and performs the butterfly operation, then it keeps one of the results and sends the other to cell 8. Cells 1 through 7 receive inputs from cells 9 through 15. If the difference between cell numbers is considered as the communication distance, there is an distance-8 communication between cells before the first set of butterfly operations are performed. At the completion of butterfly operations at cells 0 through 7, each cell sends one of its outputs to cells 0-3 and the other to cells 8-11. Therefore, there are

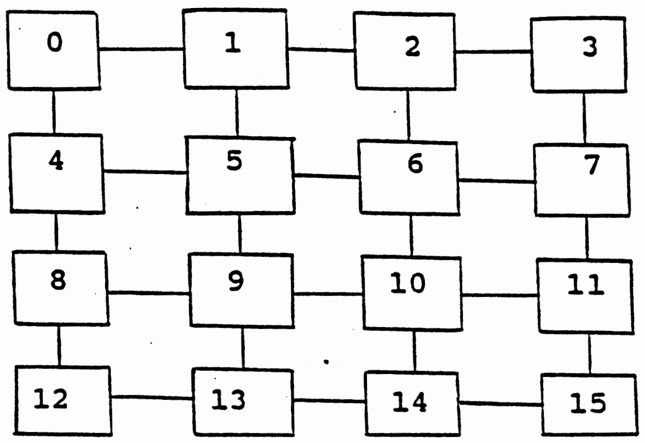


Figure 22. A Mesh Arrangement of a 16-Element Sequence.

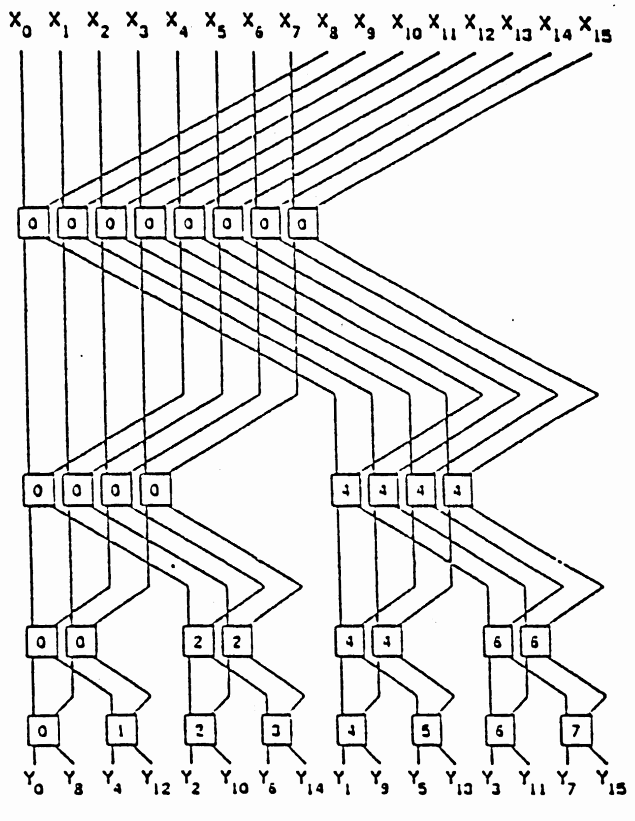

X0 X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15
0 0 0 0 0 0 0 0
0 0 0 0 4 4 4 4
0 0 2 2 4 4 6 6
0 1 2 3 4 5 6 7
Y0 Y8 Y4 Y12 Y2 Y10 Y6 Y14 Y1 Y9 Y5 Y13 Y3 Y11 Y7 Y15

Figure 23. A 16-point FFT Network (Thompson 80).

4 distance-8 and 8 distance-4 communications. Note that cells 8-15 are idle at the first stage. After completion of butterfly operations in the second row of the FFT network, cells 0-3 and 8-11 have 4 distance-4 and 8 distance-2 communications. And finally, after the third stage, there are 4 distance-2 and 8 distance-1 communications.

A careful study of Figure 22 reveals that distance-8 communications can be performed in parallel between row 1 and 3 and 2 and 4 through rows 2 and 3. If routing between near neighbors is considered to be unit routing, then distance-8 communication requires 2 unit routing. Distance-4 communication is between near neighbors on consecutive rows, therefore, it requires a unit routing. Distance-2 communication occurs between columns 1 and 3 and 2 and 4 which requires 2 unit routing. Finally, distance-1 communication is between near neighbors on columns.

The total time taken by the data movement during an FFT can be expressed in terms of unit routing performed. Thompson [Thompson 80] shows that in general in an N-cell mesh, a distance-$(N/2^k)$ communication is performed before and after the kth stage in the FFT network. The sum of the time contributions of all stages to routing will be the total routing time of the mesh.

There are logN stages in FFT network, each corresponding to a computational step in mesh. There are also 2 distance-$(N/2^k)$ communication per stage, one before and one after the computation. If $K < 1/2 \log N$, the mesh's vertical interconnections are used for routing, otherwise the horizontal connections are used. It is shown [Thompson 80] that the total routing time is proportional to $N^{1/2}$.

The processing time consists of logN cycles of butterfly computations and routing which is mainly dominated by routing. Thus it is considered to be proprtional to $N^{1/2}$. The pipeline time of mesh implementation of an FFT is also $N^{1/2}$ since a new sequence can start computation only after the current FFT is completed.

The cells in this design are not simple cells but complete processors, since data routing and stage determination and also storage for coefficients are all handled by the cells themselves, requiring larger area for basic cells.

CHAPTER III

NEW PARALLEL ALGORITHMS FOR DISCRETE FOURIER TRANSFORM

Chapter II covers the existing parallel algorithms for calculating the Fourier transform of a sequence. In this chapter, a set of new parallel algorithms are presented. These algorithms are based on the direct computation of the Fourier transform. The main computational paradigm employed is a pipeline and the designs are based on systolic arrays of cells.

The idea of systolic arrays was first proposed by Kung [Kung 79]. It is based on the decomposition of a problem into smaller problems of the same nature. A small processing cell is designed to solve the smaller instance of the problem. The cells are connected appropriately to propagate the intermediate results to the other cells in order to generate the final result. The cell interconnections are assumed to be simple and regular.

The main concept behind the idea of systolic arrays is to simplify and modularize the dedicated design of a circuit to make it amenable for VLSI implementation. The simplicity and regularity of inter-cell connections inhibits extensive wire area and complicated control.

Several pipeline systolic approaches to calculate the Fourier transform of a sequence are presented in chapter II. The new approaches proposed in this chapter are mainly inspired by certain properties of the matrix of coefficients in the matrix-vector multiplication representation of Figure 1.

The new pipeline systolic approaches are divided into three categories: N-cell linear, N-cell mesh-connected, and N^2-cell mesh-connected. The N-cell linear designs resemble the linear pipelines proposed by Kung, but they differ in the formulation, input/output format, and the structure of the basic cell. The N-cell mesh-connected design is based on these new approaches and the rearrangement of an N-element signal into an m x m matrix, $N=m^2$, as discussed in chapter II. The N^2-cell design shows an improvement over the structure proposed by Thompson [Thompson 83]. This chapter is divided into three sections, each covering the details of the designs in each category. The VLSI area-time complexity of these designs along with others presented in chapter II is presented in chapter IV.

N-Cell Linear Pipelines

The N-cell linear pipeline designs are based on the idea of N basic cells connected as a pipeline. These approaches are divided into two categories based on the form of the entry of the input sequence {x(n)} to the pipeline. The first category called On-line systolic DFT in this study, is based on the serial input of {x(n)} overlapped with the computation. The second category called In-place systolic DFT in this study, assumes the existence of the input sequence in the pipeline, one element per cell of the pipeline.

On-Line Systolic DFTs

These approaches are based on the serial input of the sequence to the pipeline from the left-most cell, one element at a time starting with the first element of the sequence {x(n)}, namely, x(0). Y(i)'s or the elements of the Fourier transform of the sequence {x(n)} reside in the registers in the processing cells, Y(0) in the leftmost cell or cell 0, Y(1) in the next cell or cell 1, and finally, Y(N-1) in the rightmost cell or cell N-1.

The elements of {x(n)} sweep the pipeline from left to right, visiting each cell, contributing appropriately to the value of Y(i) residing in that cell. The coefficients by which the elements of {x(n)} contribute to the elements of {Y(n)} are the variables of this approach whose pattern of change may be studied by careful examination of the matrix of coefficients.

The matrix-vector multiplication of Figure 1 may be rearranged as the vector summation of Figure 24. This rearrangement represents the vectors of the values by which x(i) contributes to the elements of {Y(n)}. These vectors are the columns of the matrix of coefficients. The idea may be expressed as follows:

$$\begin{vmatrix} Y_0 \\ Y_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ Y_{N-1} \end{vmatrix} = x_0 \begin{vmatrix} 1 \\ 1 \\ \\ \\ 1 \end{vmatrix} + x_1 \begin{vmatrix} 1 \\ W \\ \\ \\ W^{N-1} \end{vmatrix} + x_2 \begin{vmatrix} 1 \\ W^2 \\ \\ \\ W^{2(N-1)} \end{vmatrix} \ldots + x_{N-1} \begin{vmatrix} 1 \\ W \\ \\ \\ W^{(N-1)^2} \end{vmatrix}$$

Figure 24. Fourier Transform As Vector Summation.

$$\overline{Y} = \sum_{i=0}^{N-1} x_i \overline{M_i} \qquad 0 <= i <= N-1 \qquad (3.1)$$

Where $\overline{M_i}$ is the ith column of the matrix of coefficients. A careful study of M_i's shows that

$$\overline{M_i} = \{ W^{ij}, \quad 0 <= j <= N-1 \} \qquad 0 <= i <= N-1$$

Let $M_i(j)$ denote the jth element of M_i, then

$$\overline{M_i}(j) = 1 \qquad j = 0$$

$$\overline{M_i}(j) = \overline{M_i}(j-1) * W^i \qquad 1 <= j <= N-1$$

Therefore, if appropriate W^is are generated outside the pipeline and are input to the pipeline along with the corresponding x(i), then a simple multiplication by W^i will generate the coefficient of x(i) for the next cell.

Let C denote W^i used to generate $M_i(j+1)$ using $M_i(j)$. Then at each step through the pipeline, a cell receives x(i), $M_i(j)$ and C. The contribution of x(i) to Y(j) is made and then the coefficient for the next step is calculated by multiplying $M_i(j)$ by C.

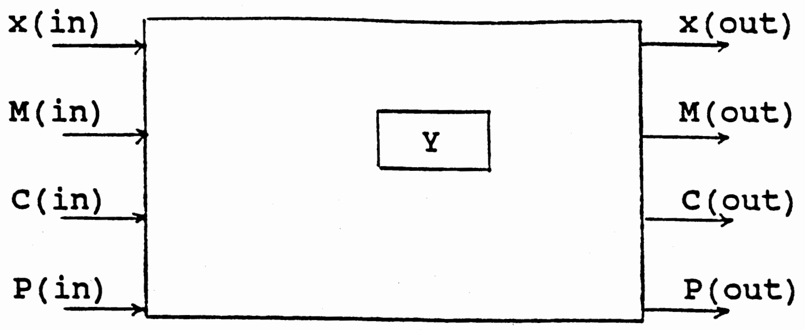
Let x(in) denote the element of {x(n)} that enters a cell, M(in) denote its corresponding coefficient, and C(in) denote W^i. These three values are the inputs to a cell at each stage. After the operation is performed, x(in) is propagated to the next cell through a line called x(out),

$M_i(j+1)$ is propagated on a line called M(out), and finally, C(in) is propagated unchanged on a line called C(out). A control signal P is added to the design to initialize Y to zero before x(0) enters the cell. The structure and operation of a basic cell and the structure of the pipeline is depicted in Figures 25a and 25b.

Figure 25a represents the structure and the operation of the basic cell. Each computational step requires two multiplications and one addition. The cell contains a single register Y to accomodate the value of the Fourier transform of the signal. Four input and four output lines are needed.

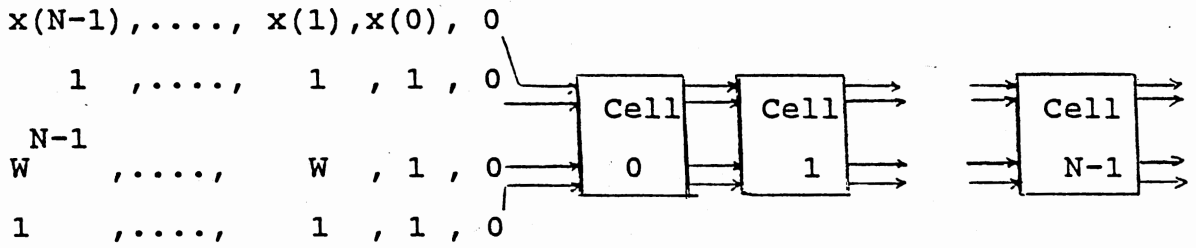
Figure 25b represents the structure of the pipeline. Input Sequence {x(n)} is fed to the pipeline sequentially in natural order. M(in) for the leftmost cell is one, since $M_i(1) = 1$ for $0<= i <=N-1$. C(in) for the leftmost cell varies following the sequence $\{1, W, W^2, \ldots , W^{N-1}\}$. This sequence may be easily generated by connecting C(in) to the output of a multiplier. P(in) follows a sequence {0, 1, ... ,1}, the zero initializes Y to zero, and the following 1's allow the computation to proceed. Other sequences {x(n)}, {M(in)}, and {C(in)} are augmented by a zero to represent the fact that they have to be delayed by one cycle to allow the initialization of Y to occur.

The pipeline requires N cells, each containing a multiplier, an adder, and a register. After N cycles, Y(0) is complete and after N-1 more cycles, the computation is completed. Therefore, this approach requires 2N-1 computational steps. If Y(i)'s are output immediately after



```
P (in)  = 0

        Y := 0
        P(out): = P (in)

P(in) = 1

        Y := Y + x (in)  *  M (in)

        M(out) := M(in) * C(in)

        x(out) := x (in)

        C(out) := C (in)
```

(a)



(b)

Figure 25. The Structure and Operation of a Basic Cell (a) and the Structure of the Pipeline for the On-line Systolic DFT.

they are completed, another sequence may start its computation after N cycles. Therefore, although the processing time of the pipeline involves 2N-1 cycles, its pipeline time may be boosted to N cycles.

In this design data entry to the system overlaps the computation. Data entry is assumed to be serial, requiring the minimum space for wires. Another feature of this design is its modularity. All cells have the same structure and function. The variable of the design, namely, C(in), is an input to the system, rather than being a built-in parameter. Therefore, an N-cell pipeline may be used to calculate the Fourier transform of any sequence of length M, $M <= N$ as long as correct W, ($W = EXP(-2\Pi j/M)$) is fed to the system and the sequence is padded with zeros to the length of N. It is also possible to expand a pipeline of N-cells to a pipeline of T cells, $T >= N$ by simply connecting the outputs of the rightmost cell of the first module to the inputs of the leftmost cell of the second module. Yet another feature of this pipeline is that Y(i)'s reside in the cells and with slight enhancement of the cells many useful operations such as filtering followed by inverse Fourier transform can be performed in-place without requiring the sequence to leave the pipeline and enter again.

One of the disadvantages of this approach is the multitude of input/output lines in the basic cell. Although this feature adds to the flexibility and modularity of the design, it causes delays due to input/output and increases the processing time. If the coefficients and the multiplicative factors used to generate them can be stored in the cells instead of being propagated, the processing time may be boosted at the cost of slightly larger cells.

In order to maintain multiplicative factors in the basic cells, the multiplicative factors by which {x(n)} contributes to Y(j) should be analyzed. Another look at the matrix of multiplicative factors of Figure 1 reveals another interesting property of this matrix. The ith row of this matrix represents the multiplicative factors used by the elements of {x(n)} to calculate Y(i). Let R(i) represent the ith row of the matrix of coefficients and x represent the vector representation of {x(n)}, then

$$Y(i) = \overline{R} \times \overline{x} \qquad 0 <= i <= N-1$$

Therefore, while the elements of {x(n)} are flowing through the pipeline, the values of R_i may be generated in each cell. A careful study of R_i reveals that:

$$\overline{R_i} = \{ W^{ij},\ 0 <= j <= N-1 \} \qquad 0<= i <= N-1$$

Let $R_i(j)$ denote the jth element of R_i, then

$$\overline{R_i}(j) = 1 \qquad j = 0$$

$$\overline{R_i}(j) = \overline{R_i}(j-1) * W^{i} \qquad 1 <= j <= N-1$$

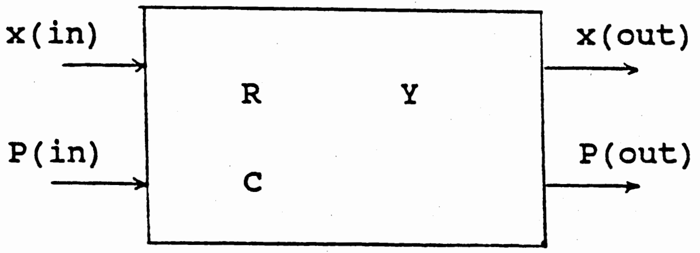
Let R denote a register in the basic cells to hold the present value of R_i and let C denote W^i, then the number of input lines to the basic cell may be reduced to one consisting only of the input value x. The control signal P is still needed to initialize Y to zero and R to 1. The structure and the operation of the basic cell and the structure of the pipeline is presented in Figure 26.

The control signal P follows the { 0,1,1,...,1 } sequence, initializing R and Y to one and zero respectively. Register C is fixed for the design at value W^i for the ith cell. Each cell contains three registers, a multiplier and an adder, an input and an output data line, and a local input and a local output control line.

The basic operation involves two multiplications and one addition, an input and an output. The processing time for the evaluation of the complete DFT is 2N-1 cycles and the pipeline time can be N if {Y(n)} is output as it becomes ready. The input/output time is reduced from three input/output per basic operation to one.

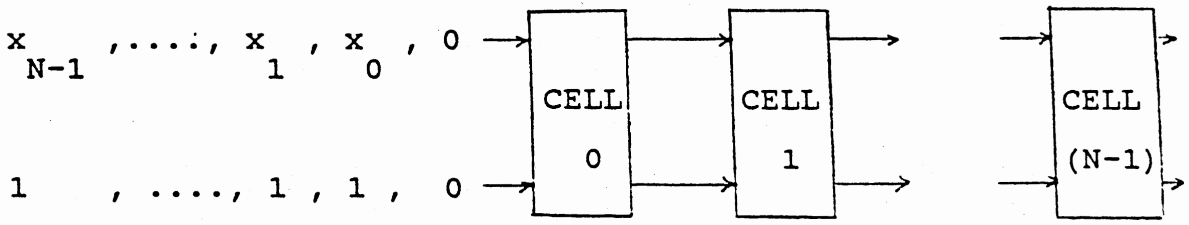
<u>In-place Systolic DFTs</u>

Kung [Kung 80] was the first to propose an in-place systolic DFT pipeline. This approach is discussed in detail in chapter II. In this section four new in-place DFT pipelines are introduced which are different from Kung's approach. In all in-place methods, it is assumed that the sequence {x(n)} is already loaded into the pipeline, one



```
P(in)  = 0

    Y := 0
    R := 1
    P(out) := P(in)

P(in) = 1

    Y := Y + R * x(in)

    R := R * C

    x(out) := x(in)

    P(out) := P(in)
```

(a)



(b)

Figure 26. The Structure and Operation of Basic Cell (a) and the Structure of the Pipeline (b) for the Modified On-line Systolic DFT.

element per cell. In Kung's approach, the sequence is loaded from right-to-left, leaving x(N-1) as the input to the left-most cell, x(N-2) at the left-most cell, and finally, x(0) at the right-most cell. Therefore, requiring only N-1 cells for an N-element DFT.

In the new approaches presented in this section, data is loaded to the pipeline of N cells in natural order, x(0) in the left-most cell, x(1) in the second cell, and x(N-1) in the right-most cell. The structure of the basic cell for the new approaches are different from Kung's approach and the number of cycles is less than Kung's approach by a factor of 1/2 for the first two designs. This reduction in the number of cycles however is at the cost of slightly larger cells. The details of these techniques are presented in the following paragraphs.

The first in-place approach in this section is based on calculating {Y(n)} stored one element per cell in natural order by allowing x's to sweep the cells and contribute to Y in each cell. Therefore, initially x(i) and Y(i), 0<=i<=N-1 are stored in the ith cell. During the first step x(i) contributes to the value of Y(i) in the ith cell. Then x(i) is propagated to the (i+1)th cell to contribute to the value of Y(i+1). The pipeline is designed to be circular so that the output of cell (N-1) may enter the cell 0. The values of {x(n)} circulate through the pipeline sweeping all the cells and contributing to the Y(i)'s stored in the cells. Therefore, the operation is completed after a complete sweep or N cycles.

Since this design is based on the complete sweep of the cells by {x(n)} stored in the cells, the coefficients by which the input values at each step contribute to the value of Y stored in the cell is different from the previous approaches.

At the first step of the computation, x(i) contributes to the value of Y(i). The coefficients used for this step are the elements of the diagonal of the matrix of coefficients of Figure 1. This sequence may be represented as:

$$\{ 1, W, W^{4}, W^{9}, \ldots\ldots, W^{(N-1)(N-1)} \}$$

The coefficients used in the second step are the values by which x(i) contributes to Y((i+1) MOD N), 0 <= i <= N-1, or the elements under the diagonal and the element at the Nth column and the first row of the matrix of coefficients. This sequence may be represented as:

$$\{ 1, W^{2}, W^{6}, W^{12}, \ldots\ldots, 1\}$$

Let R(i,j) represent the value by which x(i) contributes to Y(j) and Q(i) represent the multiplicative factor to generate R(i,j+1) or the coefficient by which x(i) contributes to Y(j+1). A careful study of these components reveals an interesting pattern. As shown before, the values of R(i,i) or the sequence of the contributions of x(i)s to

Y(j)s is as follows:

$$\{\, 1, W, W^{4}, \ldots\ldots, W^{(N-1)(N-1)} \,\}$$

Q(i) or the value used to generate R(i,j+1) from R(i,j) is W^{i} for 0 <= i <= N-1. A list of the values of R(i,i) and Q(i) for 0< =i <= N-1 is given in Figure 27.

These values may be pre-computed and stored in each cell or calculated on-the-fly before the actual DFT computation starts. On-the-fly computation adds to the flexibility of the design since W = EXP(-2 J/N) may be changed to accomodate the pipeline for the calculation of any sequence of length M <= N. It also increases the modularity of the design. However, the flexibility and modularity is attained at the price of the overhead of the added computation time.

Figure 27 shows the "on-the-fly" pattern of the computation of R(i,i)s and Q(i)s using the values of R(i-1,i-1) and Q(i-1). Basically, the following computational steps may be carried out through the pipeline to calculate and store R(i,i) and Q(i) in each cell:

$$Q(0) = 1$$

$$Q(i) = Q(i-1)*W \qquad 1 <= i <= N-1$$

$$R(0,0) = 1$$

$$R(i,i) = R(i-1,i-1)*Q(i-1)*Q(i) \qquad 1 <= i <= N-1$$

Therefore, by initializing the Q(0) and R(0,0) to 1 and

i	$R(i,i)$	$Q(i)$
0	1	1
1	W	W
2	W^{4}	W^{2}
3	W^{9}	W^{3}
4	W^{16}	W^{4}
.	.	.
.	.	.
.	.	.
N-2	$W^{(N-2)^{2}}$	$W^{(N-2)}$
N-1	$W^{(N-1)^{2}}$	$W^{(N-1)}$

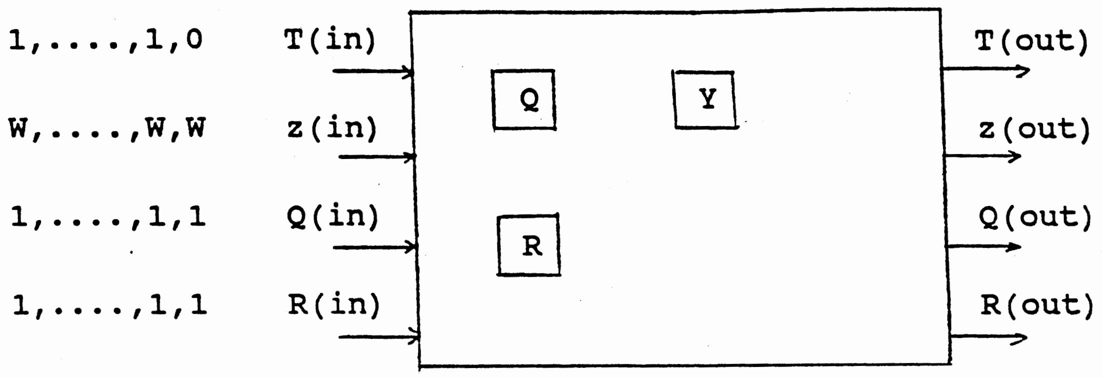
Figure 27. A List of values of R(i,i)s and Q(i)s for Different Values of i.

inputting W = EXP(-2 J/N) to the leftmost cell, the above systolic process may generate and store Q(i) and R(i,i) in the ith cell.

Let Q(in) and R(in) representing Q(i) and R(i,i) and z(in) representing W=EXP(-2πJ/N) be the set of inputs to the cell at each instance. Let Q(out) and R(out) representing Q(i+1) and R(i+1,i+1) be the set of outputs from the cell. Then the above systolic operation may be performed at each stage to generate the values of Q(i) and R(i,i). Registers Q and R in each cell are dedicated to the values of Q(i) and R(i,i). A local control signal T is added to each cell to inhibit further operations in the cells where Q and R are already computed. This signal is also used to initialize Y to zero. The structure and operation of the basic cell to generate Q(i)s and R(i,i)s is presented in Figure 28. After Q(i)s and R(i,i)s are calculated, the in-place DFT may start.

Assuming a circular pipeline of cells (last cell is directly connected to the first cell), x(i) is multiplied by R(i,i) and accumulated in Y(i). The next step is to calculate R(i,i+1) which is accomplished by multiplying R(i,i) by Q(i). Then x(i), R(i,i+1) and Q(i) are propagated to the next cell. After N iterations, Y(i) 0 <= i <= N-1 contains the ith element of {Y(n)} or the Fourier transform of {x(n)}.

The structure and operation of a complete cell to calculate the coefficients and to perform the DFT is



```
T(in) = 0            store Q(in) and R(in) and generate
                         Q(out) and R(out)

        Q := Q (in)
        R := R (in)
        Q(out) := Q(in) * z(in)
        R(out) := Q(in) * R(in) * Q(out)
        Z(out) := z(in)
        T(out) := T(in)

T(in) = 1                initialize Y and propagate T

        Y := 0
        T(out):= T(in)
```

Figure 28. The Structure and Operation of the Basic Cell to Generate Q(i)s and R(i,i)s

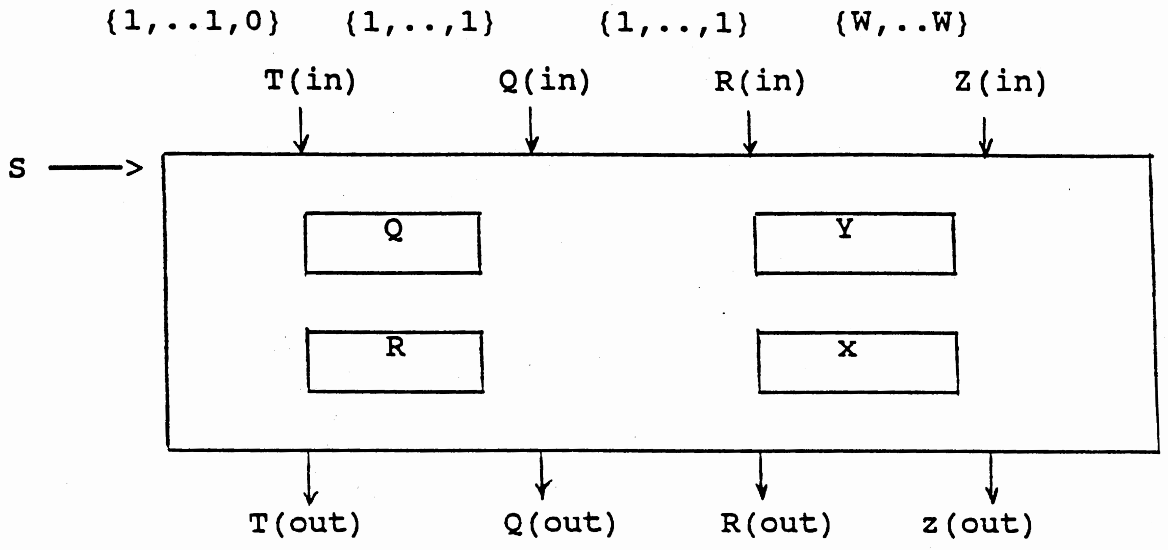
presented in Figure 29. A global signal S is used to differentiate the coefficient preparation process from the actual computation. The preparation process requires N cycles, and the calculation process requires N more cycles which yields a 2N step process for a complete DFT calculation.

The basic cell contains four registers, a multiplier, an adder, four input/output data lines and two control lines. The main reason for this added space is to attain flexibility and modularity by inputting $W = EXP(-2\Pi J/N)$ which makes the design independent of N. Sequence {Y(n)} resides in the pipeline after the completion of the process.

The number of computational steps may further be reduced if Q(i) and R(i,i) are pre-computed and stored in each cell, eliminating the preparation process and reducing the number of control signals to one. The global signal S may be used to initialize Y to zero and to load the pre-computed value of R (S=0) and then to invoke the DFT process (S=1) which is completed after N steps. The structure and operation of the basic cell for the dedicated in-place DFT is presented in Figure 30.

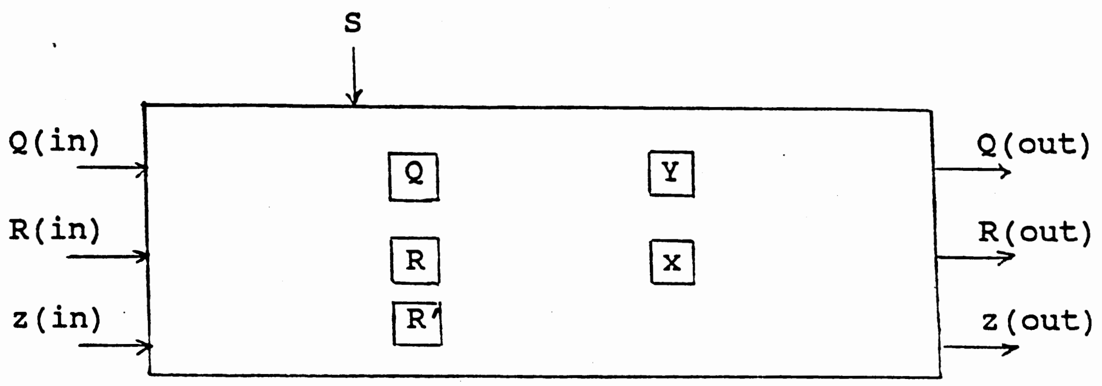
Although storing the pre-computed values in the cells eliminates the need to prepare the coefficients, it does not eliminate the need for propagating the values R and Q, since these values are related to the x's which are floating through the pipeline.

In order to reduce the data transfer rate between the cells, we may investigate the coefficients used during the



```
S = 0                                * prepare Q(i)s and R(i,i)s *

    T(in) = 0

        Q := Q(in)
        R := R(in)
        Q(out):= Q(in) * z(in)
        R(out):= Q(in) * R(in) * Q(out)
        z(out):= z(in)
        T(out) := T(in)

    T(in) = 1

        Y := 0
        T(out) := T(in)

S = 1                                  perform the in-place DFT

        Y  := Y + x * R
        Q(out) := Q
        R(out) := R*Q
        z(out) := x
        R  := R(in)
        Q := Q(in)
        x := z(in)
```

Figure 29. Structure and Operation of a Basic Cell for In-place Systolic DFT



```
S = 0              load precomputed value of R, intialize Y

     Y := 0
     R := R'

S = 1                       perform the DFT

     Y := Y + x * R

     Q(out):= Q

     R(out):= R * Q

     z(out):= x

     R   := R(in)

     Q  := Q(in)

     x  := z(in)
```

Figure 30. The Structure and Operation of the Basic Cell for Dedicated In-place DFT

process by floating values of x in correspondence with a given Y. The set of coefficients used during the four stages of the computation in each cell for N=4 is presented in Figure 31.

Obviously, these sequences have the initial value of { R(i,i) } studied in previous paragraphs. The sequence used in the second step of the computation may be generated from the first sequence by multiplying each element by W^{-i}. This generalization is true for all the following steps. Therefore, by storing W^{-i} instead of W^{i} in Q(i) the need for propagating R(i,i) and Q(i) is eliminated.

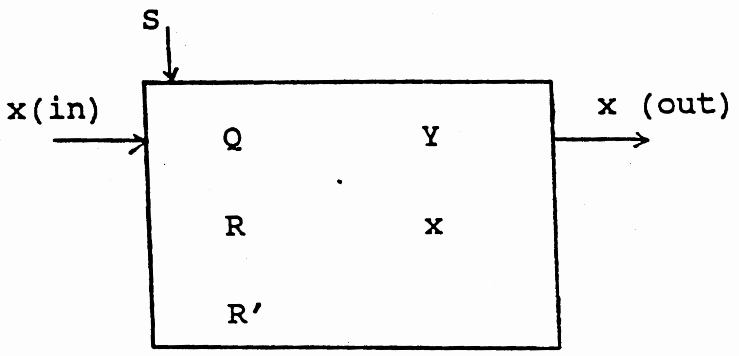
A basic cell for this approach consists of three registers Q containing W^{-i}, R′ containing the initial value of R or R(i,i), and R which holds the current value of R(i,j). Registers x and Y are used to hold the values of the sequences {x(n)} and {Y(n)}. x registers may be eliminated from the cell if parallel input of the sequence to the cells during the initial load is available.

The operation of the basic cell is the same as the dedicated design. The only difference is in the fact that R and Q are going to hold the values of the coefficient and multiplicative factor at all times, therefore, eliminating the need for their propagation.

The structure and operation of the basic cell for this approach is presented in Figure 32. As it is shown, the number of input/output data lines is reduced to one which in turn reduces the input/output time. The processing time of

STEP	Cell 0 Y_0	Cell 1 Y_1	Cell 2 Y_2	Cell 3 Y_3
1	1	W	W^4	W^9
2	1	1	W^2	W^6
3	1	W^3	1	W^3
4	1	W^2	W^6	1

Figure 31. The Set of Coefficients Used at each Cell for N=4.



```
S = 0                          * initialize R and Y

    Y := 0
    R := R'

S = 1                          * perform the DFT

    Y := Y + x * R

    R := R * Q

    x(out):= x

    x := x (in)
```

Figure 32. The Structure and Operation of the Basic Cell for In-place DFT with Minimum Communication Lines.

	Cell 0 Y_0	Cell 1 Y_1	Cell 2 Y_2	Cell 3 Y_3
STEP	(RL , RR)			
1	1	W	W^4	W^9
2	(0,1)	$(1,W^2)$	(W^2,W^6)	$(W^6,0)$
3	(0,1)	(W,W^3)	$(1,W^8)$	$(W^3,0)$
4	(0,1)	(W^2,W^4)	(W^2,W^{10})	(1,0)

Figure 33. The Sequence of Coefficients Used by Cells in a 4-cell design.

this design is N cycles. The pipeline time is also N cycles.

The pipelines for the two previous designs are circular pipelines. The disadvantage of circular pipeline is that the wire connecting the rightmost cell to the leftmost cell is in general a long wire with an O(N) length which imposes long delay and slows down the entire process. It is intresting to investigate the possibility of the float of the x's in both directions to eliminate the need for a long connection.

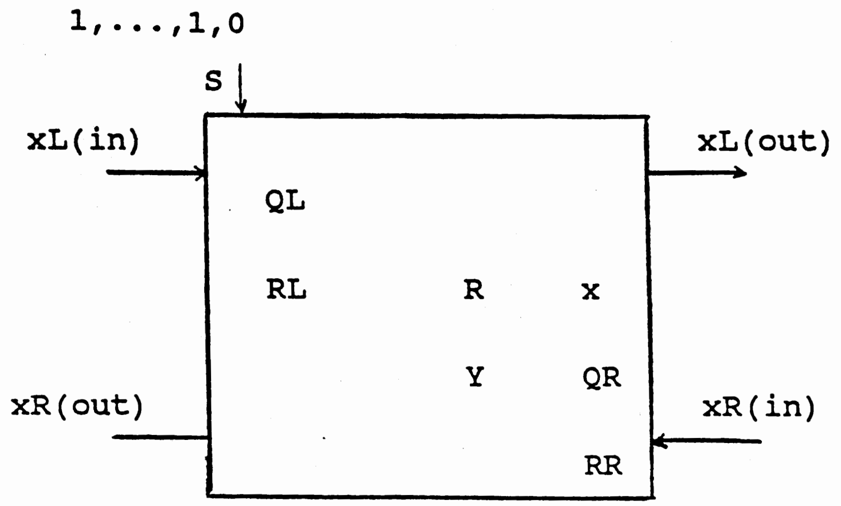
In order to allow x's to flow in both directions, the techniques used in the previous two designs are combined. Thus at the expense of larger cells and two more input/output lines, the long wire connection which adds to the space and the time complexity of the design is avoided. The values of the {x(n)} which are stored in the cells of the pipeline are now allowed to propagate to both the right and the left neighboring cells. Therefore, two sets of coefficients need to be maintained and manipulated in each cell, one for the x being input from the left neighbor xL and another for the x being input from the right neighbor xR.

The multiplicative factors associated with xL and xR are W^{-i} and W^{i} respectively. The sequence of coefficients used at each step by cells 0-3 in a 4-cell design is represented in Figure 33.

The left input to the leftmost cell and the right input to the rightmost cell are set to zero to inhibit incorrect computations. At the first step of the computation, x(i) is multiplied by R(i,i) and added to Y(i). Then x in each cell is propagated to both the right and the left neighboring cells and two x's are input to the cell, one from the left neighbor and another from the right neighbor. The coefficient for the xL and xR, RL and RR, are multiplied by W^{-i} and W^{i} respectively to attain their new values. Both coefficients are initially set to R(i,i). Then xL is multiplied by RL and added to Y. And finally, xR is multiplied by RR and added to Y.

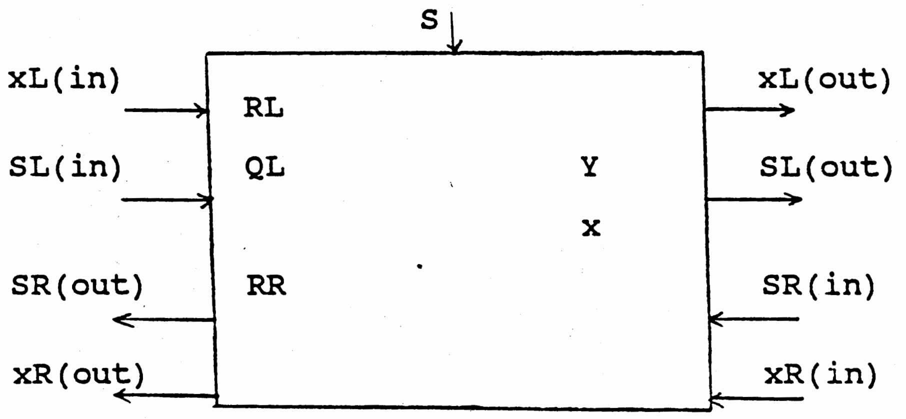
The structure and operation of the basic cell for the design is presented in Figure 34. Registers QL and QR are fixed for the design at values W^{-i} and W^{i} respectively. A global control signal S is used to initialize Y,RL, and RR and to allow x's stored in cells contribute to associated Y's (S=0) and to start the flow of x's and the computation (S=1).

It is apparent that although all the cells are fully functional at the first step, only a portion of the cells will be fully functional in the following steps. For example, the leftmost and the rightmost cells do not receive meaningful inputs on the xL and xR in the following steps and cells 1 and N-2 are not going to receive meaningful inputs on their xL and xR input lines after the second step. Therefore, in order to inhibit meaningless multiplications, two local control signals SL and SR may be added to the design to avoid multiplication by zero. SL and SR are set to 1 initially. SL(in) for the leftmost cell and SR(in) for the



```
S = 0
      Y := R*x
      RR := R
      RL := R
      xL(out):= x
      xR(out):= x

S = 1
      RR := RR * QR
      RL := RL * QL
      Y := Y + xL(in) * RL + xR (in) * RR
      xL(out):= xL (in)
      xR(out):= xR (in)
```

Figure 34. The Structure and Operation of a Basic Cell for the Bi-directional In-place Systolic DFT Pipeline.



```
S = 0
      Y := R * x
     RR := R
     RL := R

     xL(out):= x
     xR(out):=x
     SL(out):= 1
     SR(out):= 1

S = 1

     SR = 1

          RR := RR * QR
          Y := Y + RR * xL (in)

     SL = 1

          RL := RL * QL
          Y := Y + RL * xL (in)

     xL(out):= xL(in)
     xR(out):= xR(in)
```

Figure 35. The Structure and Operation of the Modified Cell for the Bi-directional In-place Systolic DFT Pipeline.

appropriate coefficients in cell i is W^{-i} $0 <= i <= N-1$. Therefore, using Q and R registers in each cell to represent the multiplicative factor and the coefficient, Q has a fixed value W^{-i} and R is initialized to W^{N**2}.

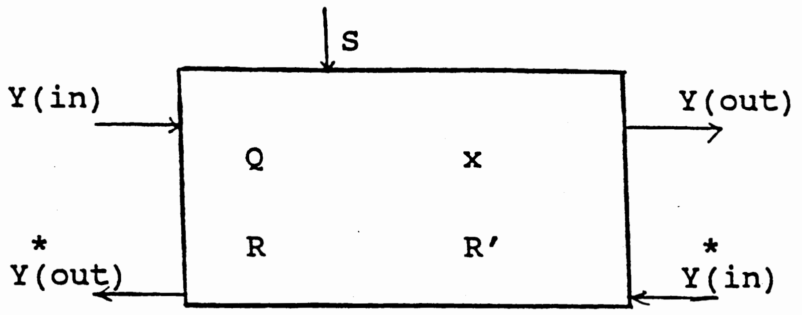
After N iterations, Y(0) is complete and is fed back to the pipeline, while Y(N-1) enters cell 0 for further computation. After N-1 more computational steps, {Y(n)} is completed and stored in cells 0 through N-1 respectively. The structure and operation of the basic cell for the linear in-place DFT pipeline is presented in Figure 36. Figure 37 represents the structure of the pipeline.

This pipeline consists of N cells, each containing four registers, two input/output data lines and one control line, a multiplier and an adder. There is no long data lines and the connections are all near neighbor. The processing and the pipeline time of this design are both 2N cycles.

Yet another technique may be used to avoid long wires by allowing zero-valued Y(i)'s to enter the pipeline from the leftmost cell and sweep the pipeline from left to right. Thus each cell should generate the sequence of the coefficients corresponding to the x(i) stored in the ith cell. These sequences are the columns of the matrix of coefficients. The ith sequence may be represented as:

$$M_i = \{ W^{ij} , 0 <= j <= N-1 \}$$

Therefore, if W^{i} is stored in each cell, and the initial



```
S = 0      * initialize and perform the first operation

    R := R'

    Y(out) := x * R

    $\overset{*}{Y}$(out) := $\overset{*}{Y}$(in)

S = 1

    R := R * Q

    Y(out) := Y(in) + x * R

    $\overset{*}{Y}$(out) := $\overset{*}{Y}$(in)
```

Figure 36. The Structure and Operation of the Basic Cell for Linear Non-circular DFT Pipeline.

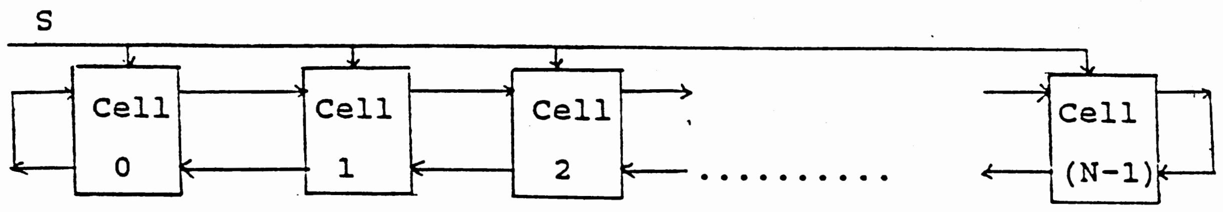


Figure 37. The Structure of The pipeline for Linear Non-circular DFT.

value of the sequense is set to 1, then the following elements of the sequence may be generated by multiplying its previous value by W^i.

Let Q and R denote the registers containing W^i and the element of the sequence. Let Y(in) denote the value of Y that enters the cell and Y(out) denote the value that exits the cell. Then the basic operation of the cell is to multiply R by x and add it to Y(in) and then generate the new R by multiplying it by Q.

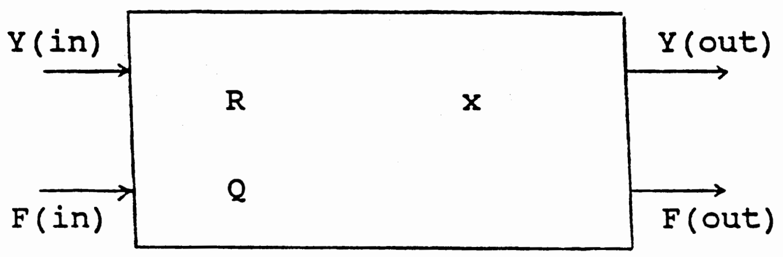
The structure and operation of the basic cell is represented in Figure 38. To assure the correct operation of the pipeline, a local control signal F is added to the design to initialize R to 1 (F=0) and to start the computation at each cell as needed (F=1).

After N iterations, Y(0) is out from the rightmost cell. If it is desired to route the result back to the pipeline, then two data lines may be added to the cells to route the output back to the pipeline by connecting the output of the rightmost cell back to itself. The modified basic cell is represented in Figure 39.

The processing time of this pipeline is 2N-1 cycles. If the resulting sequence { Y(n) } leaves the pipeline, a new sequence may start evaluation immediately after the last element of the current sequence leaves the leftmost cell. In this case the pipeline time will be N cycles.

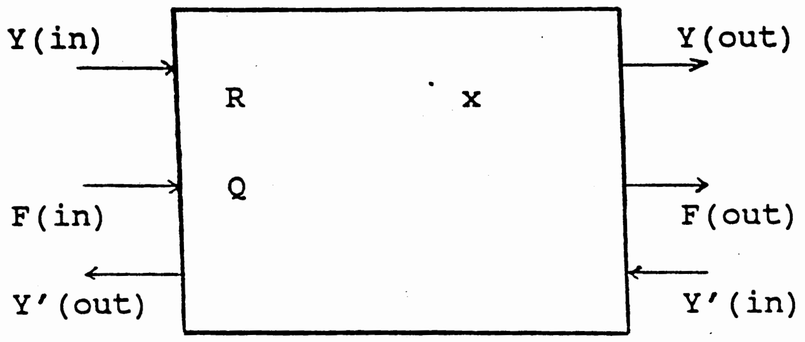
N-Cell Mesh-Connected Network

The idea of an N-cell mesh-connected DFT network was



```
F(in) = 0

        R := 1

        F(out) := F(in)

F(in) = 1

        Y(out):=Y(in) + x * R

        R := R * Q

        F(out):= F(in)
```

Figure 38. The Structure and Operation of the Basic Cell.



```
F(in) = 0
        R := 1
        F(out) := F (in)
F(in) = 1
        Y(out):=Y(in)+x * R
        R := R * Q
        F(out):= F(in)
        Y'(out):=Y'(in)
```

Figure 39. The Modified Basic Cell to Direct the Output Back to the Pipeline.

first introduced by Zhang [Zhang 84]. This approach is discussed in detail in Chapter II. The idea is based on the rearrangement of a one-dimensional sequence {x(n)} into a two-dimensional matrix given $N = m^2$.

Zhang's approach has many drawbacks. The proposed design uses an (m+1) x m network of mesh-connected cells. Input and output have an irregular form. The first row and the first column of the cells have a different form solely because of the form of Kung's recursive linear pipeline used for the basic DFT computation. A total of m multipliers are required to generate different sequences of W. And finally, a global signal S (connected to all cells) is required to assure the correct function of the network. The area used and the delay imposed by the multipliers and global signal and also excessive number of input/output lines per cell are the undesirable features of this design.

A new mesh-connected network based on the linear pipeline techniques discussed in previous sections is presented in the following paragraphs. The aim is to reduce the time and the space complexity of the design by reducing the input/output lines and avoiding the long distance connections.

The input sequence is arranged into an m x m matrix, $m = N^{1/2}$ as represented in Figure 11. The computation is divided into three phases:

1) compute the row DFTs

2) multiply the results by W^{ij} $0 <= i,j <= m-1$

3) compute the column DFTs

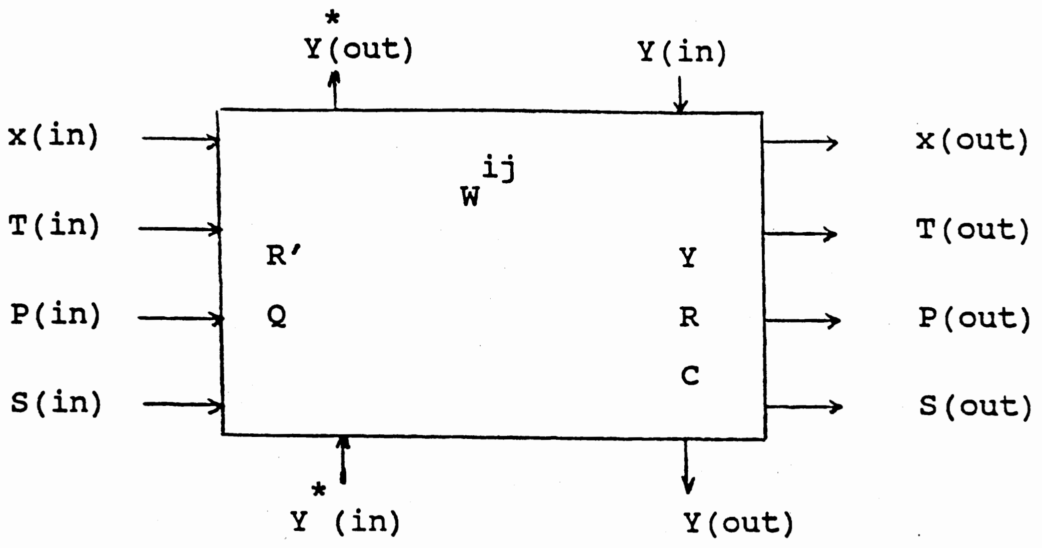
Phase 2 is combined with the initialization phase of the column DFT computation. The on-line linear DFT of Figure 26 is used for the row DFT computation. This technique allows data entry to overlap the computation. It also requires the minimum amount of input/output lines per cell.

After m iterations the Fourier transform values for column zero of the cells is computed. At this point the column DFT of column zero may start. For the column DFTs, one of the non-circular linear in-place DFTs may be used to avoid long distance connections of the other in-place DFT techniques. After 2m-1 cycles, the column DFT of column zero is complete. The computation for the entire sequence is completed after m-1 more cycles. Thus, the processing and the pipeline time of this design is 4m-2 cycles. A control signal T is added to the basic cell to stop the operation after the completion of the column DFTs at each column.

The structure and operation of the basic cell using the non-circular linear in-place DFT cell of Figure 36 is presented in Figure 40. The control signal P is used in both row column DFTS. For row DFTs, control signal P has the values {0,1,...,1}. It is set to zero to initialize the values of Y and R. It is changed to one immediately and it remains one for m cycles otr until row DFTs are completed. During this phase S is zero to indicate row DFTs. After m cycles S is changed to one to invoke column DFTs. During this phase, P is used as the global signal S in Figure 36. It is set to zero to initialize R and Y and then it is changed to one and remains one for 2m-1 cycles. During the first 3m cycles T is zero to allow row and column DFTs to proceed. After 3m cycles it is changed to one to inhibit any further calculations in the columns with completed DFTs. A register is dedicated to hold W^{ij}. Each cell consists of 6 registers, 8 input/output data lines, and 2 input/output control lines. All communications are near-neighbor. Each cell contains a multiplier and an adder. Each operational step requires 2 multiplications and an addition. The structure of the complete network is presented in Figure 41. The main advantage of this network is that the Fourier transform values remain in the network for further operations.

The structure of the basic cell may be further simplified and the pipeline time of the network may be boosted by using the cell of Figure 38 for column DFTs. Register Q in the ith cell of each row with a value of W^{i} $0 <= i <= m-1$ is used to update the initially one-valued coefficient R. A new control signal F is added which sweeps the cells from top to bottom along Y'. F is set to zero after the mth cycle to initialize R and multiply Y by C and Q and it is changed to one after that and remains one for 2m-1 cycles.

2m-1 cycles after the start of the column DFT on column zero, the associated DFT is out and a new sequence may enter



T(in) = 0 (3m cycles)

S(in)= 0 (m cycles)

ROW DFT

```
P(in) = 0
    Y := 0   R := 1
    P(out):=P(in)
    S(out):=S(in)
    T(out):=T(in)

P(in) = 1
    Y := Y + R*x(in)
    R := R * C
    P(out):= P(in)
    S(out):= S(in)
    T(out):=T(in)
```

S(in)= 1 (2m cycles)

COLUMN DFT

```
P(in)= 0
    R := R′,Y := Y*W^ij
    Y(out):=Y*R
    S(out):=S(in)
    T(out) := T(in)
    P(out) := P(in)

P(in) = 1
    R := R * Q
    Y′(out) := Y′(in)+Y*R
    Y*(out):=Y*(in)
    S(out):=S(in)
    T(out):= T(in)
    P(out):= P(in)
```

T (in) = 1 (m cycles)

* DO NOTHING *

Figure 40. The Structure and Operation of the Basic Cell for the Mesh-connected DFT Network.

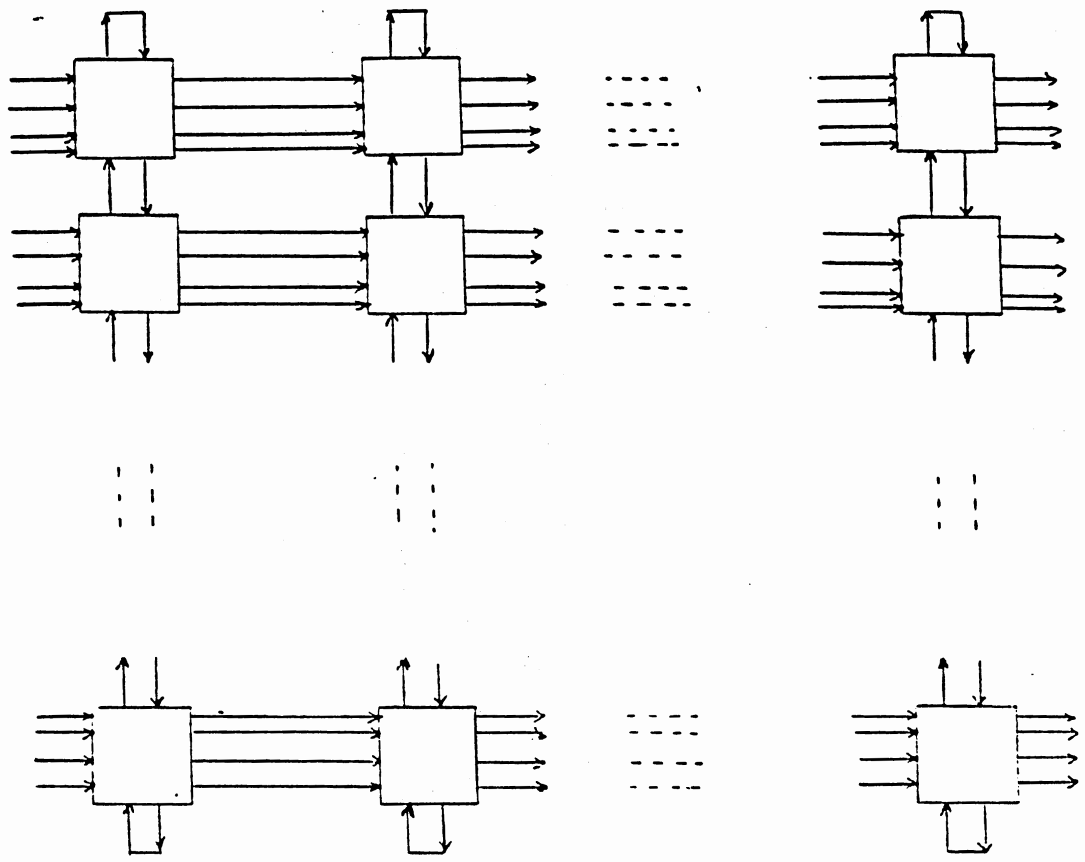


Figure 41. The Structure of N-cell Mesh-connected DFT Network.

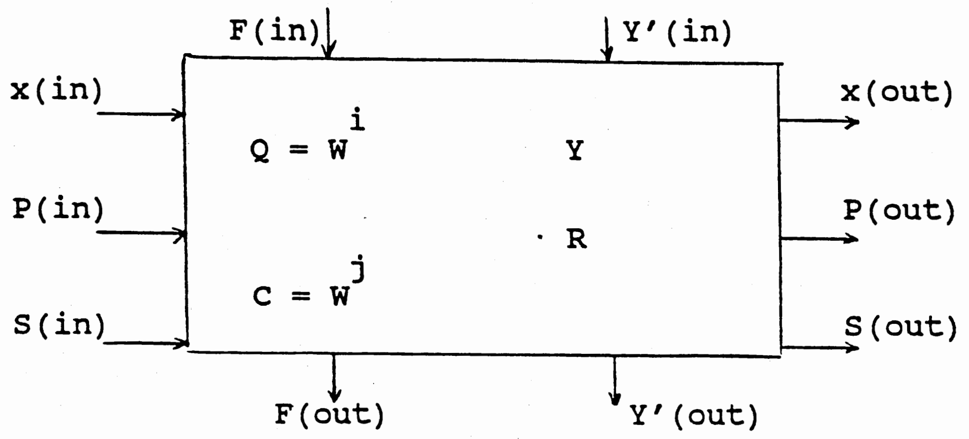
the network, boosting the pipeline time of the network to 3m. The structure and operation of the basic cell for this approach is presented in Figure 42.

It is intresting to note that the C register used as the multiplicative factor for row DFTs is fixed at value W^j for any cell at the jth column. Register Q is also fixed at value W^i for any cell at the ith row. Thus, there is no need for a separate register to hold W^{ij}, reducing the number of registers in the cell to 4. There is also no need for the control signal T in the previous design to stop the operation at columns with completed DFTs since the result is already out and a new sequence may start the computation immediately (S=0).

The basic cell with six input/output data lines and six local control lines and four registers is reasonably small. Data and control lines are all near-neighbor connected. The entire network which requires $m^2=N$ cells is presented in Figure 43. The disadvantage of this network is that the Fourier transform is transferred out of the network.

N^2-Cell Mesh-connected Network

In previous sections several methods for calculating the direct Fourier transform of a sequence of N elements using an N-element pipeline or an N-element mesh-connected network have been proposed. These techniques generate the coefficients of the transform during the computation. If N^2 coefficients used in the computation are stored in the N^2



```
S(in) = 0 (m cycles )
          ROW DFT

  P(in)= 0

      Y := 0
      R := 1
      P(out):= P(in)
      S(out):= S(in)

  P(in)= 1

      Y := Y + R * x (in)
      R := R * C
      x(out):= x(in)
      P(out):=P(in)
      S(out):= S(in)
```

```
S(in)  = 1 (2m cycles)
          COLUMN DFT

  F(in)= 0

      Y := Y * C * Q
      R := 1
      F(out):= F(in)
      S(out):= S(in)

  F(in)= 1

    Y'(out):= Y'(in)+Y*R
      R := R * Q
      F(out) := F (in)
      S(out) := S(in)
```

Figure 42. The Structure and Operation of the Cell (i,j) for the Modified Mesh-Connected DFT Network.

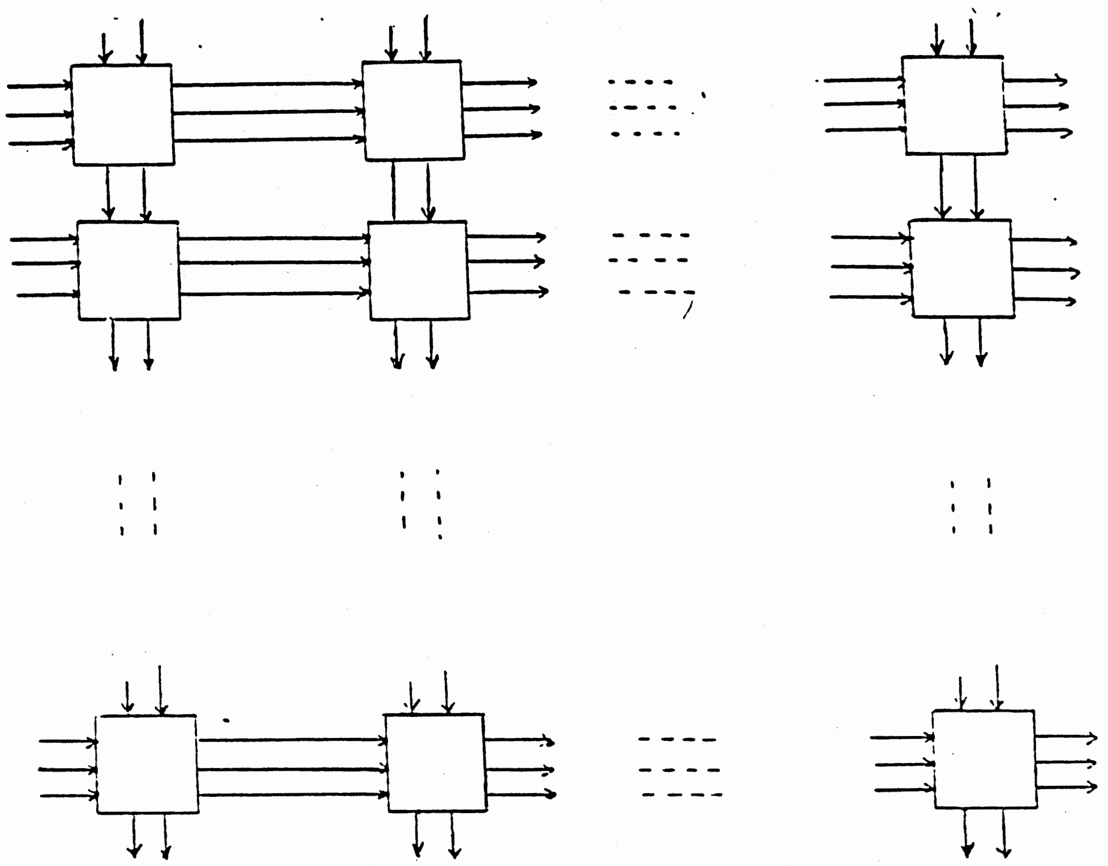


Figure 43. The Structure of the Modified N-cell Mesh-connected Network.

cells of a mesh-connected network, there will be no need to store the multiplicative factors and calculate the coefficients. This is the basis for the N^2-cell mesh-connected network to calculate the direct Fourier transform of an N-element sequence.

Thompson [Thompson 83] referred to an (4N-3)x(2N-1) cell network of mesh-connected cells to calculate the DFT using Kung's linear pipeline approach discussed in Chapter II. In this section an N^2-cell mesh-connected network is proposed which contains fewer cells.

Assuming an NxN network of mesh-connected cells, the elements of the matrix of coefficients may be stored one element per cell. Let register W in cell (i,j) 0 <= i,j <= N-1 contain W^{ij}. Then {x(n)} and zero-valued {Y(n)} may be input to the network through the topmost row and the leftmost column, one element at a time delayed by one cycle. x(0) enters through cell (0,0), x(1) enters through cell (0,1) delayed by one cycle, and x(N-1) enters through cell (0,N-1) one cycle after x(N-2) has entered the network throgh cell (0,N-2). Y(0) enters through cell (0,0) at the same time that x(0) enters the cell, Y(1) enters through cell (1,0) one cycle after Y(0), and Y(N-1) enters through cell (N-1,0) one cycle after Y(N-2) enters cell (N-2,0).

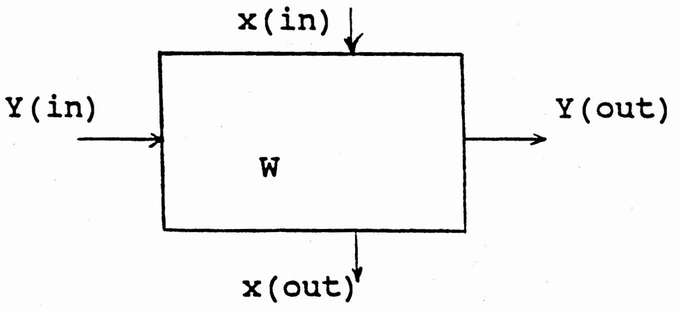
During the first cycle of the computation, x(0) and Y(0) enter cell (0,0). Y(0) is incremented by x(0) * W^{0*0}. Then x(0) is propagated to the next cell in column zero or cell (1,0) and Y(0) is propagated to the next cell in row 0 or cell (0,1). During the second cycle, Y(1) enters cell

(1,0) from the left while x(0) enters cell (1,0) from the top. Then Y(1) is incremented by x(0) * W^{1*0} and both x(0) and Y(1) are propagated. At the same time, Y(0) enters cell (0,1) from left while x(1) enters the cell from top. Then Y(0) is incremented by x(1) * W^{0*1} and they are both propagated. Therefore, the computation assumes a form of a diagonal sweep of the matrix of coefficients starting at the northwest corner and ending at the southeast corner.

The operation of the basic cell consists of a simple multiplication and addition. Each cell contains a single register and four input/output data lines. The structure and operation of the basic cell and the network is represented in Figure 44.

Y(0) leaves the network through cell (0,N-1) after N cycles, Y(1) leaves the network through cell (1,N-1) at the following cycle, and finally Y(N-1) leaves the network through cell (N-1,N-1) at the Nth cycle. Therefore, the processing time of the network is 2N cycles. However, a new sequence may enter the network immediately after x(0) and Y(0) for the current sequence leave cell (0,0) or after the first cycle. Thus, the pipeline time of the network is one cycle.

Although this technique employs large area but it is capable of accomodating for the computation of N different sequences concurrently. All the other techniques lack this capability to this extent. The only network which can simultanously process more than two sequences is the FFT



Y(out) := Y(in) + x(in) * W

x(out) := x (in)

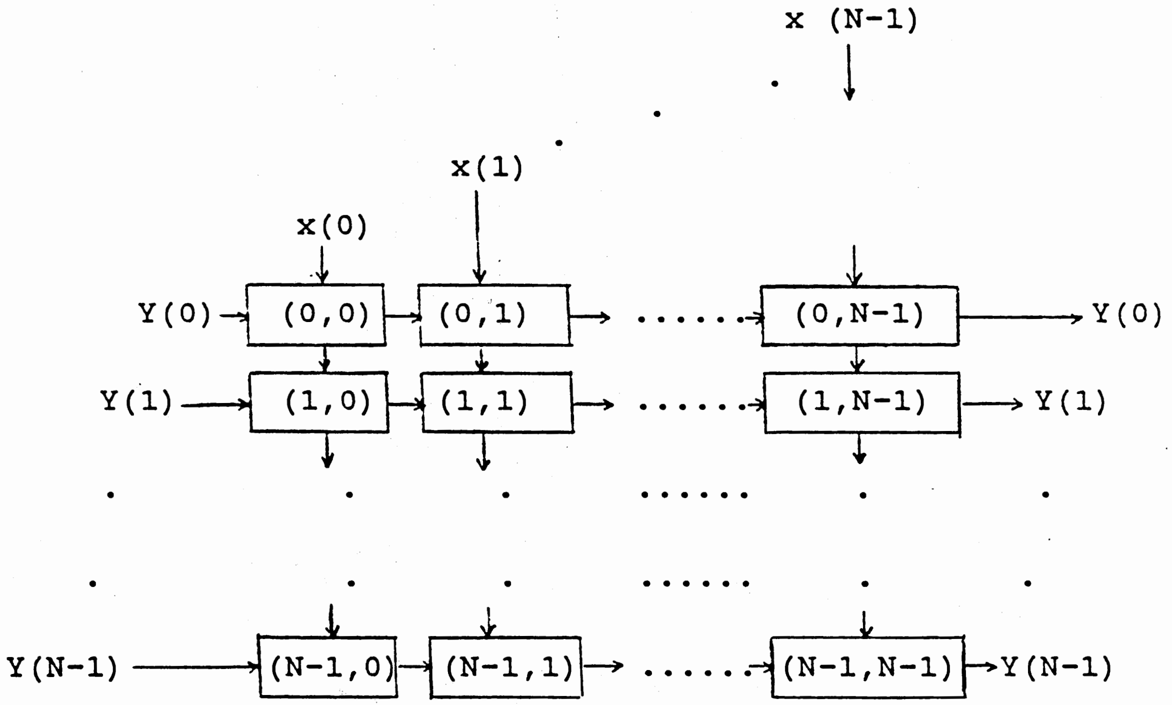


Figure 44. The Structure and Operation of Basic Cell and the Structure of the Network for the N**2-Cell DFT Network.

network which can simultanously process logN different sequences.

A summary of different methods presented in this chapter is depicted in Tables 1 and 2. Table 1 represents the physical characteristics of the designs including number of cells and the physical characteristics of the basic cell such as the number of input/output lines and the number of registers. Table 2 represents the computational characteristics of each design including the number of multiplications and additions per computational step, number of computational steps in terms of processing and pipeline steps, and the form of the input and the result.

Each design has its own advantages and disadvantages as explained throughout this chapter. Modularity is attainable at the cost of increased processing time. Fast computation is usually achieved at the cost of larger cells and inflexibility.

The best processing time performance is attained through the N-cell mesh-connected network presented in Figure 42. Although if the result is required to be kept in the network, the design of Figure 40 would be more desireable. The limitation of both methods is the size of the sequence which is limited to complete squares.

Other methods offer the possibility of dedicated implementation for any data sequence regardless of the size. The N^2-cell design has the best pipeline time performance at the cost of larger area. In-place and on-line approaches offer modular and flexible designs and also fast but

TABLE 1

PHYSICAL CHARACTERISTICS OF THE PARALLEL FOURIER TRANSFORM ALGORITHMS

METHOD OF FIGURE	NUMBER OF CELLS	NUMBER OF I/O	NUMBER OF REGISTERS
25	N	3	1
26	N	1	3
29	N	3	4
30	N	3	5
32	N	1	5
34	N	2	7
35	N	4	8
38	N	1	3
40	N	4	6
42	N	2	4
44	N^2	2	1

TABLE 2

COMPUTATIONAL CHARACTERISTICS OF THE NEW PARALLEL FOURIER TRANSFORM ALGORITHMS

METHOD OF FIGURE	NUMBER OF MULTIPLIES	PROCESSING TIME	PIPELINE TIME	RESULT
25	2	2N-1	N	resident
26	2	2N-1	N	resident
29	3	2N	2N	resident
30	2	N	N	resident
32	2	N	N	resident
34	4	N	N	resident
35	2	2N	2N	resident
38	2	2N	N	out
40	2	$4N^{1/2}$	$4N^{1/2}$	resident
42	2	$4N^{1/2}$	$3N^{1/2}$	out
44	1	2N	1	out

inflexible designs. The choice of a method however is application dependent.

CHAPTER IV

VLSI COMPLEXITY OF FOURIER TRANSFORM

Introduction

A VLSI chip is composed of transistors and interconnections. Thompson [Thompson 80] proposed a VLSI model of computation based on a graph composed of nodes representing a transistor or a small cluster of transistors and wires representing the interconnections.

Nodes are capable of storing information. A collection of nodes and wires is allowed to be a complete computing structure. The inputs and outputs to and from the computation are stored in sets of nodes called source and sink nodes respectively. A collection of nodes and wires capable of solving a problem is called a communication graph.

The VLSI model of computation is designed so that there is a direct correspondence between VLSI chips and communication graphs. Unfortunately, not all communication graphs correspond to feasible chip designs. Certain constraints are added to the VLSI model of computation to overcome this difficulty. Any communication graph that satisfies these constraints is called an admissible communication graph. A communication graph is admissible if it can be implemented as a VLSI chip of the same area and time performance [Thompson 80].

The concise definition and explanation of the VLSI model of computation and its components is presented in the following section. Two sets of assumptions are presented to define the model. One set is used to draw the lower bound measures of area and time for the communication graph. These assumptions are labeled with "L". The second set of assumptions are used to draw the upper bound measures for the communication graph and are labeled with "U".

VLSI Model of Computation

The VLSI model of computation [Thompson 80] defines the basic characteristics of a VLSI chip, namely, area, time, information, and energy. As explained before, two sets of assumptions are made which are used respectively to draw lower and upper bound complexity metrics for a communication graph.

The natural and physical units of area, time, information and energy may be defined based on the current technology, manufacturing and physical limitations. There is a natural unit of area for VLSI which is the minimal spacing between the centers of parallel wires. This spacing is 4 λ 2 [Mead 80]. The square of this length, 16 λ^2 , is a covenient area unit, large enough to contain a small transistor or one wire cross-over. The total area of a VLSI chip may be evaluated as either the mask size (smallest rectangle) or the area actually occupied by nodes and wires.

The unit of information is defined as a bit. The unit of time is defined as the bandwidth of a unit-width wire. Thus, a signal that encodes a bit has a duration of at least one time unit. Total time is measured in terms of the units of time required to solve a problem using a communication graph.

A unit of energy is defined as a product of a unit of area and a unit of time. The energy required to charge a capacitor (wire or transistor) is proportional to its capacitance which is proportional to the area. Thus the energy consumed by switching transistors in wires and gates is proportional to its area. Thus a unit of energy per unit time is consumed by a unit of area whenever it is involved in the signal transmission. The total switching energy consumed by a VLSI chip is defined as the product of the total area and the total time.

A large portion of any VLSI chip is dedicated to the conductors that distribute power and global clock signals. Since these wires do not carry information, there is no correspondence for them in the communication graph.

Notations and Metrics

The following functional notation is used throughout the rest of this chapter:

$f(n) = O\ (g(n))$: there exists a positive constant c for which $f(n) <= c\ g(n)$ for all sufficiently large n (an upper bound sufficiently large n (an upper bound measure of complexity within a constant factor)

$f(n) = \Theta(g(n))$:	there exist two positive constants c and d for which $c\,g(n) <= f(n) <= d\,g(n)$ for all sufficiently large n (an exact bound within a constant factor)
$f(n) = \Omega(g(n))$:	there exists a positive constant c for which $f(n) >= c\,g(n)$ for all sufficiently large n (a lower bound within a constant factor)
$\log x$:	the base two logarithm of x
$\log^y x$:	$(\log x)^y$
$\log\log^y x$:	$(\log(\log x))^y$

Based on the two main measures of VLSI complexity, namely, total area used by a communication graph, A, and the total time spent to solve the problem, T, a general metric is defined as AT^{2x}, $0 <= x <= 1$. This metric is used to assign different weights to the time performance of a communication graph. A special case x=1/2 gives rise to the metric AT which is a measure of total energy used. Another case where x=1 gives rise to the AT^2 metric which will be frequently referenced in this study. The general metric AT^{2x} will also be used to compare the performance of different communication graphs.

Assumptions

A communication graph is composed of nodes, and wires laid out on a grid of unit squares. A wire is a horizontal or vertical track connecting two points. A node represents a point where wires meet [Thompson 80]. Thompson [Thompson 80] defines a set of assumptions to derive the A and T measures for lower bound and upper bound complexity measures of the communication graphs. These assumptions are labeled by "L" and "U" respectively. A list and description of these assumptions appears in the following paragrphs. Almost all of the following text is adopted from [Thompson 80] unless stated otherwise.

ASSUMPTION L1: AREA. A unit square can contain one node or one wire cross-over. One wire may cross each edge of a unit square, so that nodes have a maximum of four wires.

ASSUMPTION U1: AREA. The area of a node is determined by its functionality

a) A logic node is a node with at most O(1) input wires, O(1) output wires, and O(1) area. Each of its wires is O(1) unit long. Each wire runs through at most a constant number of unit squares. Each logic node belongs to a self-timed region. A self-timed region is a set of logic nodes that receive clock pulses with nearly identical phase. all nodes within a self-timed region are in synchronization with each other [Thompson 80]. All wires connecting to a logic node must lead to or from other nodes in its self-timed region.

b) A driver node and a receiver node are associated with each wire that is more than O(1) units long or crosses the boundary of a self-timed region. A wire of length K requires a driver that occupies O(1) by O(k) unit area. Its receiver node takes up only O(1) units of area. The driver's input wire and the receivers's output wire are O(1) units long.

Arguments may be made on the use of a node with an O(K) area to drive a wire of O(K) length. Although the length of the wire could be very long the capacitance placed on it is limited and large capacitance may damage the wire.

On the other hand if the size of the driver node is increased to O(K), the other nodes connected to this node should be modified in order to be able to drive a node with O(K) area.

The optimum way to decrease the delay in an O(K) wire is to place a set of amplifiers between the O(1) area driver node and O(1) receiver node to amplify the signal to a magnitude large enough to drive the load capacitance of O(K) of the wire [Mead 80]. Mead [Mead 80] shows that if a chain of log K invertors in which each invertor is f times larger than its previous invertor is used, the total delay could be reduced to O(log K). It is shown that [Mead 80] that delay is minimized if f is chosen to be the base of natural logarithm or e.

An example of amplification chain is shown in Figure 45. The original signal from the driver is amplified through (ln K) stages of the invertors. The total area occupied by invertors is O(K) and the total delay is O(ln K). Therefore, in order to drive a wire of O(K) length in minimum delay time of O(ln K) amplifiers with the total area of O(K) must be used.

ASSUMPTION L2: TOTAL AREA. The total area of a communication graph is equal to the number of unit squares occupied by wires or nodes.

ASSUMPTION U2: TOTAL AREA. The total area of an admissible communication graph is the number of unit squares in the smallest bounding rectangle.

ASSUMPTION L3: UNIT OF TIME. A wire has at most unit bandwidth in each direction.

ASSUMPTION U3: UNIT OF TIME. A wire has at most unit bandwidth in one direction only.

ASSUMPTION L4: PROBLEM DEFINITION: Each of N input variables take on one of M different values, for a total of M^N equally likely problem instances.

ASSUMPTION U4: PROBLEM DEFINITION. log M = O(log N)

Assumption U4 restricts the word size to represent M different values of input variables to O(logN) merely to simplify the form of the upper bound measures, which would otherwise depend on M as well as N. In this study however, P = logM is used to represent the number of bits required

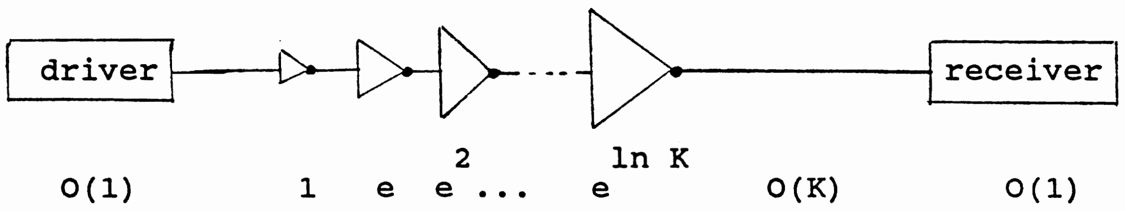


Figure 45. An Amplifier Chain to Drive a Wire of O(K).

for data representation.

ASSUMPTION L5: TRANSMISSION FUNCTION. Node states and wire signals are completely and consistently described by the transmission function associated with each node. A node with state vector S, input wires (A,...,D), and output wires (E,...,G) computes a function of the form

$$[S(t+1),E(t+\delta_E),\ldots,G(t+\delta_G)]=F[S(t),A(t),\ldots,D(t)]$$

where $_E$ and $_G$ are the non-negative delays of wires E and G.

ASSUMPTION U5: TRANSMISSION FUNCTION. The transmission function of a node is constrained by its functionality.

a) A logic node has at most O(1) bits of state and O(1) units of delay on each of its output wires.

b) The total delay through a driver-wire-receiver circuit is O(log K) if the wire is O(K) units in length. The receiver's output R(t) is the delayed version of the driver's input. The combined transmission function of the driver and the receiver is

$$R(t+\delta_W) = D(t)$$

where $\delta_W = O(\log K)$

The VLSI model of computation assumes that signals take one of the two values of zero and one. Marginal and errorneous signals which appear in real systems are not considered.

A communication graph which solves a problem with N input variables and M output variables is composed of N nodes that are dedicated to input variables which are called source nodes and M nodes that are dedicated to output variables which are called sink nodes. The initial state of source nodes is the subject of assumptions L6 and U6.

ASSUMPTION L6: SOURCE NODES. The initial state of a source node may be any function of the value of its input variable. Each input variable affects only the initial state of its source node.

ASSUMPTION U6: SOURCE NODES, INPUT REGISTERS. The initial state of the Kth node of an input register associated with the source node is the Kth bit of the binary expansion of the value of its input variable 1 <= K <= log M .

There is a one-to-one correspondence between source nodes and input variables. As there is one source node for any input variable, there is one sink node for any output variable. The function of a sink node is to collect information about the correct values for its output variables. The computation is complete when a sink node is stablized.

ASSUMPTION L7: SINK NODES. There is a fixed assertion for each sink node, relating its state to the correct values of its output variables (as a function of the values of the input variables). A computation is complete at time T if all assertions are satisfied at all times $t>=T$.

ASSUMPTION U7: SINK NODES, OUTPUT REGISTERS. The computation is complete when the Kth node of every output register contains the correct value of the Kth bit of its output variable, $1 <= K <= [\log M]$.

Problem solution is defined the same way for both the lower and upper bound measures.

ASSUMPTION L8 and U8: PROBLEM SOLUTION. A communication graph is said to solve a problem in worst-case time T if it takes no longer than T units of time to complete its computation of any problem instance. A communication graph is said to solve a problem in average time T if its average completion time over all problem instances is T.

Based on the above assumptions and the characteristics of the communication graph, Thompson [Thompson 80] derives a lower bound for Fourier transform computation. He proves that the performance of any communication graph with area A that solves a discrete Fourier transform of an N-element sequence in average time T is limited by $AT^2 >= P^2 [N/8]^2$.

He also proves that the relation $AT^{2x} = O(P^{2x} N^{1+x})$

$0 <= x <= 1$ is satisfied by any communication graph with area A that takes average time T to solve an N-element discrete Fourier transform. This lower bound is derived under the assumption that the communication graph to solve the DFT is bounded by a rectangle of area $(w-1)^2$ where $w = O(N^{1/2})$ is the minimum bisection width of the communication graph. Based on the same assumptions he proves that at least $O(PN^{3/2})$ units of energy must be dissipated by any chip solving an N-element DFT.

These lower bound measures are used as a basis for comparing the performance of DFT and FFT algorithms presented in Chapters II and III. In order to conduct a realistic analysis of the performance of the VLSI Fourier transform circuits, it is crucial to analyze the area and the time performance of the basic components used in each circuit. The asymptotic area and time performance of these basic building blocks are discussed in the following section.

The Area-Time Performance of Basic Components

The proposed VLSI designs for solving Fourier transform are mainly composed of a subset of the following basic components:

a) memory elements

b) shift register

c) adder

d) multiplier

e) processor

f) transceiver

Thus, it is important to analyze the area and time complexity of each of the above components. Since according to assumption U4, the word size to represent the input values is considered to be P=logM this measure is used in the following paragraphs to deduce the area and time complexity of the basic components.

A P-bit RAM requires an O(P) area. The access time to RAM may be reduced to as low as O(log P) if the cells are arranged in a hierarchical manner [Mead 80]. Almost all parallel Fourier transform algorithms require few memory cells if any. Thus, the memory access time is not a crucial factor in the overall time complexity of these algorithms.

Shift register is used either for data transmission or for the major component of the design. A P-bit shift register requires area of O(P). An N-bit shift requires O(P) time [Kung 81]. Therefore, the A and T measures for an P-bit shift register are bounded by O(P) and $\underline{O}$(P).

The Fourier transform computation requires complex addition and multiplication as its major computational functions. One of the most efficient forms of an adder is the carry-save adder. The basic element of a carry-save adder may be constructed using an O(1) area. Thus, a P-bit carry-save adder can be built on an O(P) area [Thompson 80]. Carry-save addition is performed in two steps requiring O(1) time.

Multiplication deserves a special consideration as the most time consuming function in evaluating the Fourier transform. Thompson [Thompson 80] shows that a P-bit multiplier built from carry-save adders will fit in an O(P) area and has a time performance of O(P). Kung [Kung 81] proves that based on convolution theorm it is possible to build a P-bit multiplier on an area of O(PlogP) with a time performance of $O(P^{1/2}\log P)$ acheiving a faster asymptotic performance than the other multipliers.

In general, Kung [Kung 81] proves that AT^{2x} performance of a P-bit multiplier is $\underline{O}(P^{1+x}N)$ for $0 \le x \le 1$. He also proves that multiplication is harder than addition. If $(AT^{2x})_M(P)$ is the AT^{2x} measure for a P-bit multiplier and $(AT^{2x})_A(P)$ is the AT^{2x} measure for an P-bit adder, then he shows that

$$\frac{(AT^{2x})_M(P)}{(AT^{2x})_A(P)} = O(P^{1/2})$$

Any one of these measures may be used in the analysis, although the slower multiplier presented by Thompson is more realistic.

One of the parallel Fourier transform algorithms; namely, the mesh implementation of FFT requires cells which are complete processors. According to Thompson [Thompson 80] a P-bit processor can be built in an area of $O(P^2)$. This micro-coded processor has O(P)-bit ALU, O(P) registers, and O(P)-bit long instruction.

Data transmission is one of the major issues in the design and implementation of any VLSI chip. Data transmission through wires can be either serial or parallel. Serial data transmission requires less space but it is slower. On the other hand parallel data transmission requires more space and it is faster. Another issue is the transmission distance. The farther the destination, the more area and time is required.

The transmission distance is divided into two categories in this study, unit-distance and N-distance. The one-bit unit-distance transmission between a driver and a receiver requires unit time. The one-bit N-distance transmission requires an O(N) time. The optimum method to improve the delay of an N-distance transmission through a chain of amplifiers was discussed in a previous section. This method requires an O(N) area for the amplifiers and reduces the delay from O(N) to O(logN).

Based on the nature of the Fourier transform algorithm serial or parallel transmission may be selected. When the time performance of the design is degraded by long-distance data transmission, amplifiers may boost the time performance at the cost of area being used by them.

A serial transceiver (transmitter receiver) requires a P-bit shift register. Thus, it can be built in an area of O(P). The time required for the transmission of one bit of information on a unit-distance wire is unit time. Therefore, the unit-distance serial transmission of P-bit data requires O(P) units of time. However, the N-distance serial transmission of P-Bit data requires an O(P) area for register plus an O(N) area for the amplifiers yeilding an O(N) area. The time required for N-distance serial transmission is the sum of the delay in the amplifiers for P-bit data which is O(P+logN).

A parallel P-bit transceiver requires a P-bit register and P wires. Thus, it requires more area than a serial transceiver. The time for P-bit parallel transceiver is O(1) since all the bits are transmitted simultanously. An N-distance parallel transceiver requires the most area among the transceivers. Each line of transmission requires O(N) area for amplifiers. Therefore, an O(PN) area is consumed by amplifiers. The time required to transmit the data however is still bounded by the delay in the amplifiers which is an O(logN).

The area-time measures for the basic VLSI building blocks of Fourier transform is summarized in Table 3. These measures will be used in the computation of the overall VLSI area-time performance of the parallel algorithms for solving Fourier transform discussed in Chapters II and III.

VLSI Complexity of the Parallel Fourier Transform Algorithms

A VLSI chip is composed of transistors and wires. The VLSI complexity theory proposed by Thompson [Thompson 80]

TABLE 3

AREA AND TIME COMPLEXITY OF THE BASIC COMPONENTS

	Area	Time
RAM	O(P)	O(log P)
shift register	O(P)	(one bit)/(time unit)
carry-save adder	O(P)	O(1)
carry-save multiplier	O(P)	O(P)
P-bit processor	$O(P^2)$	O(P)
serial unit-distance	O(P)	O(1)
serial N-distance	O(N)	O(log N)
parallel unit-distance	O(P)	O(1)
parallel N-distance	O(PN)	O(logN)

models a VLSI chip as a communication graph composed of nodes representing a transistor or a small cluster of transistors and wires. Thus the area of a chip designed to solve a problem consists of the area occupied by nodes and wires. The time required to solve a problem may be measured in different fashions. Thus it is imperative to base the analysis on a set of well-defined measures of area and time.

Let An denote the area occupied by a node and Aw denote the area occupied by a wire. Then the total area occupied by a VLSI design to solve a problem may be represented as

$$A = \sum_{\text{nodes}} An + \sum_{\text{wires}} Aw$$

This measure of total area is used for lower bound analysis. As explained previously, the area consumed by wires for power distribution and synchronization is not considered in this analysis. Unfortunately, this measure is not adequate for fabrication purposes. Instead, the total area for fabrication is considered as the area of the smallest bounding rectangle which is used in upper bound measures.

The solution time of an algorithm may be defined in many different ways. The solution time for a problem may be defined as the time elapsed between the input of the first bit and the output of the last bit. This definition of solution time is called Ts in this study. However, almost all the parallel algorithms discussed in chapters II and III use pipelines as the main architectural structure.

A pipeline is capable of accepting a new input sequence while the previous sequence is still being processed. Therefore, it is possible to simultaneously process more than one sequence of input elements through a pipeline. The pipeline time of a design referenced by Tp may be defined as the elapsed time between the input of the first bits of the two consecutive input sequences.

The processing time of a VLSI design to solve a problem is composed of the time consumed by nodes, Tn and the time consumed on transmitting the data through wires, Tw. Thus, the total time used by a VLSI chip to solve a problem may be expressed as:

$$Ts = \sum_{\text{nodes}} Tn + \sum_{\text{wires}} Tw$$

In order to simplify the comparison between the parallel algorithms discussed in chapters II and III they are classified into two categories based on their underlying architecture and not their underlying computational paradigm.

The two categories reflect the architecture and the inter-cell connection topology. The first category includes all the algorithms that use linear inter-cell connections. The cascade implementation of an FFT is included in this category since a linear connection is used to connect its cells. The second category includes all the algorithms that are based on a network of inter-connected cells. This category covers all mesh-connected designs, the FFT network, the perfect-shuffle, and the N^2-cell mesh.

In summary, the parallel algorithms to solve the one-dimensional Fourier transform of an N-element sequence are categorized as follows:

1) linear pipeline architectures (including cascade)
2) network architectures
    a) N-cell mesh
    b) Fast Fourier Transform networks
        1) FFT network
        2) perfect shuffle
    c) N^2-cell mesh

The VLSI area-time complexity of the algorithms in each category based on the area and time measures discussed in the preceeding section and the area and the time complexity of the major components is analyzed in the following sections. The given measures are all asymptotic. The comparison between the designs with the same asymptotic complexity is based on the details of the designs.

One of the major issues in VLSI design is the issue of the topology and the distance of the connections between the components. Kung [Kung 88] stresses the importance of interconnections betweem the components. He suggests that down scaling of the minimum feature size and up scaling of the maximum chip size will further accentuate the role of interconnections in a VLSI design.

He shows that although the gate delays decrease with scaling, interconnection delays remain the same. Thus, the speed at which a circuit can operate will be determined by interconnect delays rather than device delays, recommending minimal, local, and short interconnections. Therefore, in evaluating the different designs this criteria will be used as one of the determinants.

Linear Pipeline Architectures

The linear pipeline DFT architectures are analyzed in two groups based on the form of the input to the pipeline. The first group consists of all pipelines which are based on on-line algorithms allowing data input to overlap the computation. The second group consists of in-place algorithms based on the assumption that the sequence is already loaded into the pipeline.

The first on-line linear pipeline architecture for DFT is the (2N-1)-cell pipeline proposed by Kung which is presented in chapter II. This pipeline consists of (2N-1) cells, (2N-2) of which have the same basic structure. The middle cell which is the largest and also the slowest cell in the pipeline does not seriously affect the asymptotic complexity measures of the design.

The (2N-2) basic cells contain one register, one multiplier, and one adder. There are four input/output data lines per cell, requiring four input/output registers in the cell. Thus the area occupied by a basic cell, An, consists of five registers, one adder, and one multiplier. Since each component will fit in an area of O(P), the complete cell will fit in an area of O(P).

Inter-cell connection is near-neighbor, thus unit length wires will be sufficient to carry the information. The area occupied by wires Aw is O(1). Therefore, the total area occupied by the pipeline, A is (2N-2) times the area occupied by the basic cell and its connected wires, yeilding an O(PN) complexity.

This algorithm involves (4N-3) computational steps. Each step requires two multiplications and one addition, thus Tn=O(P). The time required for data transmission Tw is also O(P) for serial data transmission. Therefore, the time complexity of each step is O(P). Since (4N-3) steps are involved in the complete computation the total processing time is T has a complexity of O(PN). The pipeline time is equal to the processing time, having the same time complexity.

Two new on-line linear pipeline algorithms have been proposed as a part of this study. The basic cells for these pipelines are presented in Figures 25 and 26 respectively. These two methods are comparable to the Kung's (2N-1)-cell pipeline in the sense that they all allow data entry to the pipeline overlap the computation.

The pipeline in Figure 25 consists of N cells. Each cell requires one memory cell, three input/output registers, one multiplier, and one adder, thus An = O(P). Inter-cell connections are near-neighbor, requiring unit-distance data transmission, thus Aw = O(1). Therefore, the total area per cell is O(P). There are N cells in the pipeline, thus, A = O(PN).

The total computation requires (2N-1) steps. Each step involves two multiplications, one addition, and serial data transmission. Thus, Tn = O(P). The total processing time T is therefore O(PN). The pipeline time is also O(PN).

The other on-line linear pipeline is presented in Figure 26. This pipeline requires the least amount of input/output lines of the three on-line approaches. Each cell contains three memory cells, one input/output register, one multiplier, and one adder, thus An = O(P). Inter-cell connection is near-neighbor, thus, Aw = O(1). Therefore, the total area occupied by a cell and its wires is O(P). The total area occupied by the pipeline or A is O(PN).

A total of (2N-1) steps are required to complete the evaluation of the Fourier transform of an N-element sequence. Each step requires two multiplications and one addition, thus, Tn = O(P). Data transfer is near-neighbor, thus Tw = O(P). Therefore, the total processing time for each step is O(P). The total processing time for the pipeline, T , is O(PN).

Although the three on-line linear pipeline DFT approaches have the same asymptotic area and time complexity, the pipeline of Figure 26 has the best overall performance.

The pipelines of Figure 25 and 26 have half the number of cells of the pipeline proposed by Kung. They require smaller cells (one less memory space) and almost half the processing time. Therefore, they both have better area and time performance than Kung's (2N-1)-cell pipeline. The pipeline of Figure 26 requires less area than the pipeline of Figure 25 since it requires two less wires per cell which saves an O(N) wire area.

The other class of linear pipeline architectures for Fourier transform is the class of in-place pipeline algorithms. The main difference between the in-place and the on-line algorithms is in the form of data input. On-line algorithms allow data input to the pipeline overlap the computation. In-place algorithms on the other hand assume that data elements are already loaded into the pipeline.

The issue of the initial load of the input sequence into the pipeline should be addressed at this point. The sequence {x(n)} may be loaded to the pipeline either serially through the inter-cell communication lines or in parallel through a separate bus. Serial input requires an extra O(PN) delay time since O(PN) bits should travel through the pipeline. Since the inter-cell connections are near-neighbor an O(1) time is needed to transfer each bit, thus, the time required to load the data is O(PN).

The parallel data transmission may be established through a separate data transmission line for each cell. The wire area consumed by this method is O(N) and the delay is O(P). Dedication of N off-chip lines for data transmission may not always be practical. On the other hand if N input registers are dedicated to the input values an extra O(PN) area would be consumed.

The first alternative; namely, serial data transmission through the inter-cell communication lines of the pipeline seems more practical and economical. The price to be paid however, is the overhead of O(PN) delay before the computation can start. Therefore, in-place linear pipeline algorithms suffer from an either area or time overhead that the on-line linear pipeline algorithms do not. The VLSI complexity of the in-place linear pipeline algorithms is analyzed in the following paragraphs.

The pipeline based on the recursive formulation of DFT is presented in Figure 9. This pipeline requires (N-1) cells. Each cell requires two input/output registers, one memory cell, one multiplier, and one adder. Since all components will fit in O(P) area, then An = O(P). Cells have near-neighbor connections, thus Aw = O(1). Consequently, the total area per cell is O(P). The pipeline has (N-1) cells, therefore, A = O(PN).

This method requires (2N-3) computational steps. Each step involves one multiplication and one addition. Thus, Tn = O(P). Inter-cell connections are near-neighbior. serial data transmission between cells for P-bit data items require O(P) time. Thus Tw = O(P). Total processing time for the algorithm is T = O(PN). A new sequence may enter the pipeline after N steps, thus the pipeline time is also O(PN).

A set of new algorithms for in-place DFT is proposed in this study. The first pipeline whose basic cell is presented in Figure 29 is designed to be flexible and modular. The coefficients are calculated through the first phase of the computation, thus they may be modified to accomodate for any sequence of size M <= N.

Each cell requires four memory cells, one adder and one multiplier, thus An = O(P). Three data lines and one control line connect the consecutive cells in the pipeline, thus Aw = O(1). Therefore, the total area used by a cell and its wires is O(P). The pipeline has N cells, thus, A = O(PN). A global control line is used which occupies an O(N) area. The long wire connecting the rightmost cell to the leftmost cell also occupies an O(N) area. Although this area is asymptotically negligible, it increases the size of the design.

The computation of DFT through the pipeline requires 2N steps. Each step includes three or two multiplications based on the phase of the computation and one addition, thus Tn = O(P). The time required to input three data items is O(P). The time required to route the global control signal is O(log N), Thus, Tw = O(P+log N). The total time spent during a computational cycle is O(P). Since the entire computation requires 2N cycles, T = O(PN). The duration of the computational cycle for this approach is longer than actual time required for computation and near-neighbor inter-cell propagation since the delay on the long wire connecting the rightmost cell to the leftmost cell is O(log N) at its best.

The second new in-place algorithm allows the initial values of the coefficient to be pre-stored in the cells. The basic cell for this algorithm is presented in Figure 30. As a result of storing the coefficients in the cells, the first phase of the computation is eliminated and the entire operation requires N cycles instead of 2N cycles. This pipeline has the same area and time complexity of the pipeline of Figure 29, but requires half the number of computational cycles to calculate the Fourier transform of an N-element sequence.

The basic cell presented in Figure 32 is designed to reduce the number of input/output lines. The total wire area for inter-cell communication thus is reduced from three to one. Consequently, the total wire area for the complete design is reduced from 3N in the two previous pipelines to N. This is especially important when long wires are used since for each communication line an area with O(N) complexity must be dedicated to the amplifiers.

The pipeline with the basic cell presented in Figure 32 has the same asymptotic area and time complexity as of the other previous two pipelines.

The three linear pipeline methods presented in Figures 29,30, and 32 are all circular. The first algorithm is more flexible but it requires more computational steps. The next two pipelines both require N computational steps. The circular pipeline with the basic cell of Figure 32 occupies smaller area since it has a single input line in each cell. Therefore, among the three circular pipelines presented in this study, the pipeline with the basic cell of Figure 32 has the best area and time performance. In all three cases the result of the operation resides in the pipeline which facilitates further operations on the signal.

Two alternate methods are presented in Chapter III to avoid long wires. The first pipeline whose basic cell is presented in Figure 36 allows the Y's to sweep the cells. The outgoing Y's are routed back to the pipeline for further computations. The result resides in the cells of the pipeline. Although the long wire is eliminated, the number of computational steps is increased from N to 2N cycles and cells are not fully functional.

The second alternative has the basic cell presented in Figure 38. Zero-valued Y's enter the pipeline through the leftmost cell, sweep it from left to right and exit from the rightmost cell. The basic cell requires three memory cells, one input/output register, one multiplier, and one adder. The inter-cell connection is near-neighbor, thus, $An = O(P)$ $Aw = O(1)$, and $A = O(PN)$.

The complete process requires 2N-1 cycles, although a new sequence may enter the pipeline after N cycles. Each cycle requires two multiplications, one addition, and one input, thus $Tn = O(P)$. Inter-cell connections are near-neighbor, thus, $Tw = O(1)$. And finally, the complete process has $T = O(PN)$ time complexity. This cell may be modified to route the results back to the pipeline as represented in Figure 39.

The comparison between the three in-place methods, namely, Kung's recursive method, pipeline with the basic cell of Figure 32 and the one with the basic cell of Figure 38 is rather complex and involves many issues such as the form of output, area, computation time, and finally the propagation time.

All three aforementioned pipelines use N basic cells. The result of the computation in the first and the third pipeline leaves the pipeline. If two more lines are added to the basic cell, the result may be redirected back to the pipeline which results in an increase in the number of input/output registers by one. After this modification, the two last pipelines will have the same area for the basic cell. Kung's pipeline has the smallest cell among the three with one less memory cell per basic cell.

The second pipeline uses a long wire which adds an O(N) area to the pipeline for the O(N) long wire and its amplifiers. The long wire also causes an O(log N) delay through the pipeline for the propagation. Thus, the cycle time of the second pipeline is longer than the cycle time of the other two by a factor of O(log N). On the other hand, the second pipeline requires N cycles of computation instead of (2N-3) cycles for the first pipeline and (2N-1) for the third.

A complete cycle for the basic cell is composed of the time required for multiplication tm, the time required for addition ta, and the time required for propagation tp. Then the total computation time for each method T(i) 1 <= i <=3 can be expressed as:

$$T(1) = \sum_{1}^{2N-3} (tm + ta + tp)$$

$$T(2) = \sum_{1} (2tm + ta + tp)$$

$$T(3) = \sum_{1}^{2N-1} (2tm + ta + tp)$$

tp = P for the first and the third method. however, it is P+logN for the second method due to the delay in the long wire. Then

$$T(1) = (2N-3)tm + (2N-3)ta + (2N-3)P$$

$$T(2) = 2Ntm + Nta + N(P + \log N)$$

$$T(3) = 2(2N-1)tm + (2N-1)ta + (2N-1)P$$

The first and the second pipeline are compareable. Let Td = T(1)-T(2), then

$$Td = -3tm + (N-3)ta + NP - N\log N - 3P$$

ignoring the constant factors,

$$Td = N(ta + P - \log N) = O(N)$$

Thus the second method has a time performance better than the first method by a factor of O(N) at the cost of extra area of O(N) for the wires and amplifiers.

The two pipelines selected from the two on-line and in-place categories as having better area-time performance in their category may also be compared. The best pipeline in the category of on-line pipelines is presented in Figure 26 and the best pipeline in the category of in-place pipelines is presented in Figure 32.

Both pipelines require N cells. The first pipeline has smaller cells, and no long wires. Thus, the complete pipeline is smaller than the other pipeline by a factor of O(N).

Let T(i) 1<= i <=2 denote the total computation time for the two pipelines. If the time required for the initial load is added to the computation time, then

$$T(1) = \sum_{1}^{2N-1} (2tm + ta + tp)$$

$$T(2) = N \log N + \sum_{1}^{N} (2tm + ta + tp)$$

Since tp = logN for the first pipeline and P+logN for the second pipeline, then

$$T(1) = 2(2N-1)tm + (2N-1)ta + (2N-1)\log N$$

$$T(2) = N\log N + (2N)tm + (N)ta + N(P+\log N)$$

Therefore, the difference between T(1) and T(2) ignoring constant terms is

$$Td = (2N)tm + (N)ta - PN$$

Since tm = O(P) and assuming that the time required for multiplication is comparable to the time required for data transmission, the above equation indicates that the on-line pipeline is slower than the in-place pipeline.

In summary, although all linear pipeline methods have the asymptotic area and time complexity of O(PN), it is shown that the in-place pipeline of Figure 32 has a better overall performance.

The cascade implementation of FFT is the last design that is classified under linear pipeline architectures, solely for its basic architectural characteristics.

The cascade pipeline is composed of logN basic cells. Each cell contains one multiplier, one adder, two memory cells, one counter, and two input/output registers. Therefore, a total of four memory cells is required in each basic cell. The complete cell will fit in an O(P) area, thus, An = O(P). Inter-cell connections are near-neighbor, thus, Aw = O(1). Therefore, the total area occupied by the cells is O(P logN). The ith cell in the pipeline however is connected to a shift register of length $2^{(m-i)}$ (m=logN). The area occupied by the shift registers is the sum of the number of elements in the shift registers or 2^m=N multiplied by the size of each element which is O(P). Thus the total area occupied by the pipeline is O(P logN+ PN) which is an O(PN).

The computation involves (2N-1) cycles although the pipeline time is only N cycles. Each computational cycle involves at least one multiplication although half the cycles involve two multiplications and two additions. Each cycle involves a P-bit shift. Thus, the total time may be expressed as:

$$T = \sum_{1}^{2N-1} (2tm + 2ta + ts + tp)$$

Since the cycle time should be longer than the longest operation, ignoring the constant factors,

$$T = 4Ntm + 4Nta + 4PN$$

Obviously, the total time used by cascade is twice the time used by other linear pipelines except Kung's (2N-1)-cell pipeline. To this time we must add the time required to convert the output from bit-reversed order to normal order.

The area used by cascade however is smaller than the area used by the other pipelines since the main area is occupied by shift registers and not adders and multipliers. Thus, it is reasonable to claim that cascade occupies an area at least three times smaller than the other linear pipelines.

The main disadvantage of cascade however lies on its inflexibility. The design is limited to solving the Fourier transform of sequences of size $N=2^m$. It is not easily expandable since its hardware is especially designed for a specific value of N.

A summary of physical and computational characteristics of the linear pipelines studied in this section is presented in Tables 4 and 5. Table 4 represents a list of physical characteristics of the pipelines and also the asymptotic area complexity of the basic cell and the entire pipeline. Table 5 represents the computational characteristics of the pipelines and the asymptotic time complexity of the basic cycle and the complete computation.

Since the asymptotic area and time complexity of all pipelines for solving Fourier transform is the same, an approximation to the area and time of each design may assist in the evaluation of the performance of each pipeline.

TABLE 4

PHYSICAL CHARACTERISTICS OF THE LINEAR PIPELINE ARCHITECTURES

method	#cells	#I/O reg	#memory cells	total memory	Aw	An	A	other
1) Kung's non-recursive	2N-1	4	1	5	4	O(P)	O(PN)	
2) Figure 25	N	3	1	4	3	O(P)	O(PN)	
3) Figure 26	N	1	3	4	1	O(P)	O(PN)	
4) Kung's recursive	N-1	2	1	3	2	O(P)	O(PN)	
5) Figure 29	N		3	4	4	O(P)	O(PN)	O(N) long wire
6) Figure 30	N		5	5	3	O(P)	O(PN)	O(N) long wire
7) Figure 32	N		5	5	1	O(P)	O(PN)	O(N) long wire
8) Figure 36	N	2	4	6	2	O(P)	O(PN)	
9) Figure 38	N	1	3	4	1	O(P)	O(PN)	
10) cas-cade	logN	2	2	4	2	O(P)	O(PlogN)	O(PN)

TABLE 5

COMPUTATIONAL CHARACTERISTICS OF THE LINEAR PIPELINE ARCHITECTURES

method	#steps	#multi	#add	Tw	Tn	T	pipeline	other
1)	4N-3	2	1	O(P)	O(P)	O(PN)	O(PN)	
2)	2N-1	2	1	"	"	"	"	
3)	2N-1	2	1	"	"	"	"	
4)	2N-3	1	1	"	"	"	"	initial load O(PN)
5)	2N	3(2)	1	"	"	"	"	"
6)	N	2	1	"	"	"	"	" O(logN) long wires
7)	N	2	1	"	"	"	"	"
8)	2N	2	1	"	"	"	"	O(PN) initial load
9)	2N-1	2	1	"	"	"	"	
10)	2N-1	2	2	"	"	"	"	

This approximation is based on the assumption that all major components occupy an areaof P. Thus the area of a cell with 5 memory cells, one multiplier, and one adder is approximated as 7P. Then the total area is approximated by multiplying the number of cells by the area of the cell.

The processing time is approximated by assigning P units of time to multiplication and inter-cell serial transmission, P+logN for N-distance transmission, and unit time for addition. The total time is approximated by multiplying the number of steps required by the total cycle time. The summary of these approximations along with the approximation for pipeline time, the restrictions on the size of the sequence and also the advantages and disadvantages of the ten linear pipelines are listed in Table 6.

Some of the pipelines are easily expandable due to their flexible design. Flexibility and modularity is considered to be one of the major evaluation criteria. The other issue is the form of the output. The pipelines which save the result in the pipeline are considered more favorably, since as it was previously mentioned slight enhancements in the cells will allow many useful operations to take place in the cell without a need to transfer the result out and then route it back again. The multitude and the length of the inter-cell connections is also considered. Extensive wire area or long wires are considered as a negative factor.

As it shown in Table 6, the first pipeline occupies the largest cell area and the cascade implementation of FFT

TABLE 6

APPRIOXIMATED AREA AND TIME MEASURES FOR LINEAR PIPELINE ARCHITECTURES

pipeline	approximated area		approximated time	pipeline time	N	advantage	disadvantage
	cells	wires					
1)	14PN	8N	16PN + 4N	16PN + 4N	-	easily expandable	result is out too much time and area
2)	6PN	3N	6PN + 2N	3PN + N	-	easily expandable result remains	too much time and area
3)	6PN	N	6PN + 2N	3PN + N	-	result remains few wires	hard to expand
4)	5PN	2N	5PN + 2N	3PN + N	-	easily expanable	result is out
5)	6PN	4N	7PN+2NlogN +2N	6PN+ 2N+2NlogN	-	result remains easy to expand	too much time and area
6)	7PN	3N	3PN + N+NlogN	3PN + N +NlogN	-	result remains fast	hard to expand
7)	7PN	N	3PN + N +2NlogN	3PN + N +2NlogN	-	result remains fast few wires	hard to expand
8)	8PN	2N	7PN + 2N	7PN + 2N	-	result remains	hard to expand too much area
9)	6PN	N	7PN + 2N	4PN + N	-	few wires	result is out
10)	PN	2logN +N	6PN + 4N	3PN + 2N	2^m	small area	very hard to expand

occupies the smallest cell area. These two pipelines are also associated with the two extremes for the wire area. The pipeline of Figure 32 has the least wire area and computational time among the comparable pipelines. although the choice of one pipeline over others is application dependent, Table 6 gives a better insight to the comparative area-time measures of each pipeline.

Fourier Transform Networks

All parallel Fourier transform architectures with non-linear interconnections are classified as Fourier transform networks. The first class of parallel Fourier transform architectures is the mesh-connected architectures. Three mesh-connected architectures are covered in this study. Two mesh-connected architectures are covered in Chapter II and one is covered in Chapter III.

The mesh-connected architecture proposed by Zhang [Zhang 84] and the one proposed by this study are based on the rearrangement of an N-element one-dimensional signal into a two dimensional matrix assuming that $N = m^2$. The third mesh-connected architecture is based on the FFT algorithm assuming that $N = 2^{2m}$.

Another class of Fourier transform network architectures includes the FFT network which is the hardware implementation of the FFT flow graph, and the perfect-shuffle interconnection network which has a more regular architecture than the FFT network.

Finally, the N^2-cell mesh-connected network is the only one of its kind with the capability of concurrently handling N different sequences. The VLSI complexity of these networks is analyzed in the following sections.

N-cell Mesh-Connected Architectures. Zhang [Zhang 84] was the first to propose a mesh-connected architecture for DFT. The architecture is based on the fact that $N = m^2$. The architecture is composed of $N = m \times m$ basic cells. A set of m extra cells are added to the architecture as the (m+1)th row of the network to multiply the outgoing result by W^{ij} and route it back to the network. Thus, the complete network consists of $(m+1) \times m$ basic cells. A set of (m+2) multipliers are also required to generate the appropriate powers of P needed by the Kung's linear pipeline to perform column and row DFTs and to generate appropriate powers of W to be used by the last row of the cells. Thus, the total number of cells used in the architecture is $((m+1)m+m+2)=(m^2+2m+2)$ cells. Assuming that the transceiver in each cell can receive and transmit data in all directions, at least three input/output registers are required in each cell. Each cell also contains two memory cells, one multiplier, and one adder. Thus $An = O(P)$. Each node is associated with 5 data lines. Thus, $Aw = O(1)$.

The wire area used by this architecture for long wires directing the global signal S and the powers of P generated by the two multipliers is significant. Since S is a global signal, it has to be directed to all the nodes. Thus, it requires one horizontal and m vertical long wires of O(m). Therefore, to the wire area required by the architecture the amount $mP + m^2$ must be added (the cell area is assumed to be a rectangle of logN by 1).

The processing time of this network involves 4m cycles. To this time the time required to load the sequence which is at least O(m) must be added. Each computational cycle involves one multiplication and one addition. Thus Tn = O(P). Inter-cell connections are near-neighbor, thus, Tw = O(logN). Therefore, the total processing time T is O(mP). The pipeline time for this network is the sum of the processing time and the time required for the initial load, since a new computation may not start unless the previous computation is completed and the new sequence is completely loaded. Thus, the pipeline time is approximately 5mP which is O(mP).

The N-cell mesh-connected network for DFT proposed by this study which is presented in Figure 42 is an attempt to reduce the area and the time overhead and to avoid long wires as much as possible. All control signals are local, the initial load is overlapped with the computation, and the inter-cell data communication is minimized. The basic cells however are slightly largerthan Zhang's network.

The complete network consists of N basic cells arranged as an m x m matrix. No extra cell or multiplier is required. Thus, the initialization of multipliers and their synchronization with the basic cells is not an issue. Each cell contains one input/output register since at any cycle only one data item is input or output. Each cell contains four memory cells, one multiplier, and one adder. Thus, An = O(P). Each cell is connected to two data lines, thus, Aw = O(1). No other extra area for long wires or multipliers are needed. Therefore the total area used by the architecture is $O(PN^{1/2})$.

The complete computation requires 4m cycles. Although a new sequence may start loading and computation after 3m cycles. Therefore, although processing time involves 4m cycles, the pipeline time only involves 3m steps. Each cycle consists of two multiplications and one addition. Thus, a complete cycle requires an O(P) time or Tn = O(P). Inter-cell connection and control signal transmission is near-neighbor which requires an O(P) time, thus, Tw = O(P). Therefore, the complete processing time has an O(m P) complexity. No other time overhead is involved.

The last mesh-connected network based on the FFT network was first proposed by Stevens [Stevens 71] for the Illiac IV computer. The size of the sequence N is restricted to 2^{2m}. The network consists of N cells arranged as a $2^m \times 2^m$ matrix. Data is loaded in a row-major order as depicted in Figure 22 prior to the computation.

Each cell is a message driven processor capable of creating new messages, addressing it to another cell in the network, forwarding a message along the shortest path to its destination, and finally receiving a message. The functional aspects of this network is discussed in Chapter II. Thompson [Thompson 80] shows that the computation time for this network is basically dominated by the time required for routing. He suggests that the transceiver in each node should have parallel transmission to other nodes and serial transmission to the multiply-add unit. The proposed transceiver occupies an $O(P^2)$ area. Using the large parallel transceiver reduces the parallel transmission time to $O(\log P)$ but adds an area of P for wires to each processor in each direction. The area occupied by each node also contains logN coefficients being used in logN steps of the computation, multiply-add circuit and control circuit. Thompson proves that the complete processor will fit in an $O(P^2)$ area. Thus the complete network occupies an $O(P^2 N)$ area for cells and an O(PN) area for wires.

The computation involves log N cycles. Each cycle consists of one multiplication, two additions, and routing. The routing distance depends on the stage of the computation as explained in Chapter II. Thompson [Thompson 80] shows that each node in the kth stage performs two $(N/2^k)$-distance routing. When $K < (\log N)/2$, the mesh's vertical interconnections are used. The horizontal interconnections are used otherwise. He proves that the total routing time is proportional to $(N^{1/2} \log P)$.

The physical and computational characteristics of the three mesh-connected networks are summarized in Tables 7 and 8. Table 9 represents a rough approximation of the time and the area used by each architecture. The approximation is based on the assumptions made in the previous section.

Comparing the physical and computational characteris-

TABLE 7

PHYSICAL CHARACTERISTICS OF MESH-CONNECTED NETWORKS

	Zhang	This Study	Stevens
# cells	$N + 2N^{1/2} + 2$	N	N
# I/o registers	3	1	2
# memory cells	2	4	log N
total memory	5	5	2 + logN
Aw	5	2	4P
An	O(P)	O(P)	$O(P^2)$
A	O(PN)	O(PN)	$O(P^2N)$
other	$PN^{1/2} + N$ (wires)		4PN (wires)

TABLE 8
COMPUTATIONAL CHARACTERISTICS
OF MESH-CONNECTED
NETWORKS

	Zhang	This Study	Stevens
#steps	$4N^{1/2}$	$4N^{1/2}$	logN
# multi-plications	1	2	1
# additions	1	1	2
Tw	O(P+logN)	O(P)	O(logP)* (number of routing steps)
Tn	O(P)	O(P)	O(P)
T	$O(PN^{1/2})$	$O(PN^{1/2})$	$O(N^{1/2} \log P)$
other	$O(PN^{1/2})$ (initial load)		$O(N^{1/2})$ (routing) $O(PN^{1/2})$ (initial load)

TABLE 9

APPROXIMATED AREA AND TIME MEASURES FOR MESH CONNECTED NETWORKS

	Zhang	This Study	Stevens
cells	$(7PN + 7PN^{1/2})$	$7PN$	$P^{2}N$
wires	$5N$	$2N$	$2PN$
approx time	$8PN^{1/2} + 4N^{1/2} + 8N^{1/2} \log N^{1/2}$	$12PN^{1/2} + 4N^{1/2}$	$(4N^{1/2} \log P + 4P^{2}$
approx pipeline time	$9PN^{1/2} + 4N^{1/2} + 8N^{1/2} \log N^{1/2}$	$9PN^{1/2} + 3N^{1/2}$	$(4N^{1/2} \log P + 4P^{1/2} + PN^{1/2}$
N	m^{2}	m^{2}	2^{2m}
Advantage		no long wires easy to expand	
dis advantage	excessive hardware long wire		large basic cells, extensive area, hard to expand

tics of the three networks, it is apparent that the network proposed by this study requires less cell area than the other two networks since it is reasonable to assume that P > 7. It also requires less wire area. It is difficult to find a good approximation for the time used by Stevens's mesh-connected network since the time used by the processor to create, receive and direct a message, fetch the coefficient, etc is not known. The approximation given in Table 9 includes the time for multiplication and routing. The approximated time of the other two networks are comparable. However, the pipeline time of the network proposed by this study is better than Zhang's network by a factor of 4m. Considering the other advantages of this network and the fact that the structure is extremely modular, this network is considered to have a better overall performance.

Fast Fourier Transform Networks. Two different fast Fourier transform networks are discussed in Chapter II. The first network is the hardware replica of the FFT flow graph called FFT network. The second architecture is based on the perfect-shuffle network proposed by Stone [Stone 71].

The FFT network is composed of logN levels of N/2 cells. Each cell receives two inputs, performs a butterfly operation, and sends two outputs to two other cells. The coefficient for the butterfly operation is stored in the cell. Each cell contains one multiplier and one adder, thus, An = O(P). Thompson [Thompson 80] shows that the connections emerging from the kth row (k=0,1,..., logN - 1) occupy

$N/2^{k+1}$ tracks, as it is shown in Figure 17. He shows that a total of N/2 horizontal tracks are necessary and sufficient for laying out the interconnections between the first two rows. The connections between the second and the third row occupy N/4 horizontal tracks. Thus a total of N-1 horizontal tracks are required to lay out the interrow connection.

Each track is a long wire. Thus, in order to obtain the minimum delay, appropriate amplification is required. The wires between the first two rows cross the middle line, therefore, their length is proportional to N. Since each wire needs an amplifier with O(N) area, then an $O(N^2)$ area is required for the amplifiers between the first and the second row. There are N/4 tracks between the second and the third row which although do not cross the middle line but they occupy N/4 wire area. Thus, they each require N/2 area for amplifiers, a total of $N^2/8$ area. Hence the total area occupied by wires and amplifiers is $O(N^2)$. If the cells are laid out as a rectangle of O(P) tall and O(1) wide, then the bounding rectangle would require an $O(N^2)$ area. The area occupied by amplifiers is a very important factor in this network.

A total of (logN) cycles is required to complete the computation. Although a new sequence may enter the network after the current sequence leaves the first row or after the first cycle. A computational cycle involves the time for data input, one multiplication, and one addition. Thus, Tn = O(P). The time required for propagation assuming that amplifiers are added to the netwok is Tw = O(log N). Thus, the total computation time T is O(logN(P+logN)). The pipeline time however is O(P+log N). FFT network is capable of processing logN sequences of data simultanously.

The perfect-shuffle network was first proposed by Stone [Stone 71]. Thompson [Thompson 80] discusses the VLSI complexity of the FFT using the perfect-shuffle interconnection network. Thompson [Thompson 80] proposes a planar layout for the network. He shows that the N/2 cells in the network can be partitioned into logN equivalence classes, B(k), k=0,1,...,logN-1, where each class contains the cells with the same number of 1's in their binary expansion. For example, for N=16, there are 8 cells and the equivalence classes are as follows:

B(0) = { cell 0 }

B(1) = { cells 1,2, and 4 }

B(2) = { cells 3,5, and 6 }

B(3) = { cell 7 }

It is easy to verify that cells in B(k) are connected to other cells in B(k) and to cells in B(k-1) and B(k+1). Clustering the cells by their equivalence classes helps limit the length of the wires. Unfortunately, the equivalence classes are rather large. Thompson [Thompson 80] shows that the largest class has a size $O(N/\log^{1/2} N)$. Consequently, the wires connecting this class to other classes may be as long as $O(N/\log^{1/2} N)$. Figure 46 adopted from [Thompson 80] represents the planar embedding of the perfect-shuffle network for N=16.

The equivalence classes B(0) and B(3) contain input values {x(0),x(1)} and {x(14),x(15)} respectively. Equivalence classes B(1) and B(2) each contain three cells and accomodate for 6 input values. The connections between two consecutive classes are represented with large rectangular boxes. The size of the box depends on the size of the equivalence classes it is connecting. The width of the box is the sum of the widths of the two classes. The height of the box is proportional to the number of the connections between the two classes which is equal to the sum of the cells in two classes. Since the wires used in these connections could be as long as $O(N/\log^{1/2} N)$, adequate amplification is needed to achieve the O(logN) delay in the line. The area occupied by the amplifiers to drive an $O(N/\log^{1/2} N)$ wire is $O(N/\log^{1/2} N)$. Therefore, if amplifiers are added to each wire, the interconnection box for the two largest classes (B(1) and B(2) in Figure 46) would require $O(N/\log^{1/2} N) * O(N/\log^{1/2} N)$ area which is $O(N^2/\log N)$. Thus the total area occupied by the network is O(N) wide due to the width of the cells and O(N/logN) tall due to the interconnections, consequently, $A = O(N^2/\log N)$.

The complete computation requires logN steps. Each step requires one multiplication, two additions, and the time required for routing which is O(P). Thus Tn = O(P). The long wire transmissions are localized thus Tw = O(log N). Therefore, the complete process requires $T = O(P\log N + \log^2 N)$

processing time. The pipeline time is equal to the processing time since a new sequence may not enter the network unless the previous computation is completed. Perfect-shuffle network is capable of handling only one sequence at a time.

The physical and computational characteristics of the two networks are summarized in Tables 10 and 11. Table 12 represents the approximated area and time for the networks and also the restrictions on the size of the sequence and the advantages and disadvantages of each network.

The approximated time and area calculated for FFT network and perfect-shuffle network represented in Table 12 shows that the two networks have almost the same area-time performance. FFT network occupies a slightly larger area. Perfect-shuffle network on the other hand has a longer pipeline time. Both networks are restricted to the sequences of size $N=2^m$ and they are both very hard to expand. FFT network will be used as the best network in this category in the comparative analysis among different classes of architectures merely for its better pipeline time.

N^2-cell Mesh-Connected Network

The N^2-cell mesh architecture proposed by this study has intresting properties which the other network architectures lack. This network is capable of concurrently processing N different data sequences. The pipeline architectures are capable of simultanously processing at most two sequences. Other mesh-connected arcitectures are capable of

TABLE 10

PHYSICAL CHARACTERISTICS OF THE FOURIER TRANSFORM NETWORKS

	FFT network	perfect-shuffle
# cells	N/2 logN	N/2
# I/O registers	2	2
# memory cells	1	logN
total memory/cell	3	2+logN
Aw	$N/2^{K+1}$ horizontal tracks for wires emerging from the kth row	$O(N^{1/2}/\log N)$
An	O(P)	O(PlogN)
A	$O(N^2)$	$O(N^2)$

TABLE 11

COMPUTATIONAL CHARACTERISTICS OF THE FOURIER TRANSFORM NETWORKS

	FFT network	perfect-shuffle
# steps	$\log N$	$\log N$
# multiplications/step	1	1
# additions/step	2	2
Tw	$O(P+\log N)$	$O(P+\log N)$
Tn	$O(P)$	$O(P)$
T	$O(P\log N+\log^2 N)$	$O(P\log N+\log^2 N)$
pipeline time	$O(P+\log N)$	$O(P\log N+\log^2 N)$
other	delay through amplifiers $O(\log N)$	delay through amplifiers $O(\log N)$

TABLE 12

APPROXIMATED AREA AND TIME MEASURES
FOR FOURIER TRANSFORM NETWORKS

	FFT network	perfect-shuffle
cells	$5PN/2 \log N$	$PN/2(\log N+4)$
wires	N^2	N^2
time	$2P\log N + \log^2 N$ $\log^2 N$	$2P\log N + 2\log N+$ $\log^2 N$
pipeline time	$P+\log N$	$2P\log N +2\log N$ $\log^2 N$
N	2^m	2^m
advantage	fast capable of handling logN differnt sequences simultanously	fast
disadvantage	too much wire and amplifier area hard to expand	very difficult to expand

processing one or two sequence at a given instance. FFT network is the only network capable of simultanously processing logN sequences. This characteristics may be considered as an important determinant in the evaluation of the Fourier transform architectures if the flow of data is comparable with the speed of the computation.

The N^2-cell mesh architecture is composed of N^2 basic cells connected as a grid of NxN cells. Each cell contains one multiplier, one adder, one memory cell, and two input/output registers. Thus An = O(P). Inter-cell connections are all near-neighbor, therefore, Aw = O(1). Thus, the total network should fit in an area of A = $O(PN^2)$.

A total of 2N-1 cycles are required for the completion of the processing of a sequence. The pipeline time however is only one cycle. Each cycle consists of data input, one multiplication, and one addition, thus, Tn = O(P). The inter-cell connections are near-neighbor and serial, thus, Tw = O(1). Therefore, T = O(PN). The pipeline time on the other hand is O(P).

Table 13 summarizes the area-time performance of the best architectures in each category. A, T, and Tp denote the total area, total time, and pipeline time respectively. Measures AT, AT^2, and AT^{2x} are evaluated using both processing and pipeline time.

Linear pipelines and N^2-cell mesh may be used to calculate the Fourier transform of any given sequence regardless

TABLE 13

COMPARATIVE ANALYSIS OF DIFFERENT FOURIER TRANSFORM ARCHITECTURES

	Linear pipeline	mesh (this study)	FFT Network	N^2-cell mesh
A	$O(PN)$	$O(PN)$	$O(N^2)$	$O(PN^2)$
T	$O(PN)$	$O(PN^{1/2})$	$O(\log^2 N+P\log N)$	$O(PN)$
Tp	$O(PN)$	$O(PN^{1/2})$	$O(P+\log N)$	$O(P)$
AT	$O(P^2N^2)$	$O(P^2N^{3/2})$	$O(PN^2\log N+N^2\log^2 N)$	$O(P^2N^3)$
ATp	$O(P^2N^2)$	$O(P^2N^{3/2})$	$O(PN^2+N^2\log N)$	$O(P^2N^2)$
AT^2	$O(P^3N^3)$	$O(P^3N^2)$	$O(P^2N^2\log^2 N+N^2\log^4 N)$	$O(P^3N^4)$
ATp^2	$O(P^3N^3)$	$O(P^3N^2)$	$O(N(\log^2 N+P))$	$O(P^3N^2)$
AT^{2x}	$O(N^{1+2x}P^{1+2x})$	$O(N^{1+x}P^{1+2x})$	$O(N^2(P\log N+\log^2 N)^{2x})$	$O(N^{2+2x}P^{1+2x})$
ATp^{2x}	$O(N^{1+2x}P^{1+2x})$	$O(N^{1+x}P^{1+2x})$	$O(N^2(P+\log N)^{2x})$	$O(P^{1+2x}N^2)$
N	unrestricted	m^2	2^m	unrestricted

of its size. The other two networks however, are restricted to specific values of N.

Linear pipelines and N-cell mesh-connected DFT network proposed by this study occupy the smallest area. FFT network on the other hand has the best processing time. The N^2-cell mesh has the best pipeline time. The N-cell mesh-connected DFT network has the best AT, ATp, and AT^2 performance among all the architectures. Mesh-connected DFT network and N^2-cell mesh -connected network have the same ATp^2 performance. The FFT network has better overall ATp^2 performance. However, it is only capable of simultaneously processing logN different sequences. Thus, when the capacity of concurrent data processing is a determinant, N^2-cell mesh is more appropriate choice at the cost of slightly higher AT^2 measure. Mesh-connected DFT network also has the best AT^{2x} performance.

Comparing the results of this study with the lower bound for AT,AT^2, and AT^{2x} measures deduced by Thompson [Thompson 80], it is apparent that all measures deduced for the N-cell mesh-connected DFT are only a factor of P away from the lower bounds. Considering the other advantages of this design; namely, simplicity and ease of VLSI implementation, modularity and simplicity and regularity of interconnections it is selected to be the most efficient design.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

The comparative analysis of the VLSI area-time complexity of the parallel algorithms for Fourier transform presented in Chapter IV shows that the N-cell mesh-connected designs have better overall performance. The simplicity of the basic cell and also interconnections in this category is also one of the major advantages of these designs. The modular layout provides flexibility and allows ease of expansion. The regularity of interconnections is also an advantage in lieu of VLSI implementation.

The issues that would be raised for actual VLSI implementation are not addressed in this study. The physical characteristics of the available technology and the goals of the design will determine the actual VLSI layout.

The decomposition of the one dimensional signal to three or more dimensions is also a possiblity which may be addressed in the context of existing planar VLSI circuits or the possibility of future non-planar VLSI circuits.

REFERENCES

Bongiovanni G. " A VLSI Network for Variable Size FFT's". IEEE Transactions on Computers, Vol. C-32, No.8, Aug 1983, pp. 756-760.

Cooley J. W. and J. W. Tukey. "An Algorithm for the Machine Computation of Complex Fourier Series." Math. Comput. Vol. 19, Apr. 1965, pp. 297-301.

Chow P., Z.C. Vranesic, and J.L. Yen. "A Pipelined Distributed Arithmetic PFFT Processor." IEEE Transactions on Computers, Vol. C-32, No.12, Dec. 1983, pp. 1128-1136.

Cozzens J. and L. Finkelstein. "Computing the Discrete Fourier Transform Using Residue Number Systems in a Ring of Algebraic Integers." IEEE Transactions on Information Theory, Vol. IT-31, No.5, Sep. 1985, pp. 580-588.

Despain A. "Very Fast Fourier Transform Algorithms Hardware Implementation." IEEE Transactions on Computers, Vol. C-28, No.5, May 1979, pp. 333-341.

Gertner I. and M. Shamash. "VLSI Architecture for Multidimensional Fourier Transform Processing." IEEE Transactions on Computers, Vol. C-36, No.11, Nov. 1987, pp. 1265-1274.

Gold Ben and Theodore Bially. "Parallelism in Fast Fourier Transform Hardware." IEEE Transactions on Audio and ElectroAcoustics, Vol. AU-21, No.1, Feb. 1973, pp 5-16.

Groginsky H.L. and G.A. Works. "A Pipeline Fast Fourier Transform." IEEE Transactions on Computers, Vol. C-19, No.11, Nov. 1970, pp. 1015-1019.

Kung H. T. "Special-Purpose for Signal Processing opportunity in very large Scale Integration (VLSI), " Real-time Signal Processing 111, 1980.

Kung H. T. and R. P. Brent. "The Area-Time Complexity of Binary Multiplication." JACM, Vol 28, No.3, July 1981, pp. 521-534.

Kung H. T. "Why Systolic Architectures." Computer, Jan 1982, pp. 37-46.

Kung S. Y. "VLSI array Processors." 1st Ed. Prentice-Hall, Englewood Cliffs, 1988.

Mead C. and L. Conway. "Introduction to VLSI Systems." Addison-wesley, 2nd Ed., 1980, chapter 8.

Norton A. and A. Silberger. "Parallelization and Performance Analysis of the Cooley-Tukey FFT Algorithm for Shared Memory Architecture." IEEE Transactions on Computers Vol. C-36, No.5, May 1987, pp. 581-591.

Oppenheim Alan V. and Ronald W. Schafer. "Digital Signal Processing" 1st Ed. Prentice-Hall, 1975.

Owens R. and M. J. Irwin. "The Arithmetic Cube." IEEE Transactions on Computers, Vol. C-36, No.11, Nov. 1987, pp. 1342-1348.

O'Leary Dianne P. "Systolic Array for Matrix Transpose and other Reorederings." IEEE Transactions on Computers, Vol. C-36, No.1, Jan. 1987, pp. 117-122.

Reed I. and I. K. Troung. "A New Hybrid Algorithm for Computing a Fast Discrete Fourier Transform." IEEE Transactions on Computers, Vol. C-28, No.7, July 1979, pp. 487-492.

Stevens j. " A Fast Fourier Transform Subroutine for ILLIAC IV." Center for Advanced Computing, University of Illinois, Technical Rep. 1971.

Stone H. "Parallel Processing With Perfect Shuffle." IEEE Transactions on Computers, Vol. C-20, No.2, Feb. 1971, pp. 153-155.

Taylor F. J. and G. Papadourakis, A. Skavantzos, and A. Stouraitis. "A Radix-4 FFT Using Complex RNS Arithmetic." IEEE Transactions on Computers, Vol. C-34, No.6, June 1985, pp. 573-576.

Thompson Clark. "A Complexity Theory for VLSI". Dissertation, Carnegie-Mellon University, Aug. 1980.

Thompson T. "Fourier Transforms in VLSI." IEEE Transactions on Computers, Vol. C-32, No.1, Nov. 1983, pp. 1047-1057.

Troung T. K., J. J. Chang, I. S. Hsu, D. Y. Pei, and I. S. Reed. "Techniques for Computing the Discrete Fourier Transform Using the Quadratic Residue Fermat Number System." IEEE Transactions on Computers, Vol. C-35, No.11, Nov. 1986, pp. 1008-1012.

Troung T. K., I. S. Reed, I. Hsu, H. Shyu, and H. M. Shao. "A Pipeline Design of a Fast Prime Factor DFT on a Finite Field." IEEE Transactions on Computers, Vol. C-37, No.3, March 1988, pp. 266-273.

Wold E. and A.Despain. "Pipeline and Parallel FFT Processors for VLSI Implementation." IEEE Transactions on Computers, Vol. C-33, No.5, May 1984, pp. 414-425.

Zhang N. "Multi-Dimensional Systolic Networks For Discrete Fourier Transform." Proceedings of the 11th International Conference on Computer Architecture, Ann Arbor, Michigan, 1984, pp. 215-222.

VITA

Taraneh Baradaran-seyed

Candidate for the Degree of

Doctor of Philosophy

Thesis: VLSI COMPLEXITY OF PARALLEL FOURIER TRANSFORM

Major Field: Computer and Information Sciences

Biographical:

Personal Data: Born in Tehran, Iran, October 18, 1952

Education: Graduated from Aryamehr University of Technology, Tehran, Iran, in January 1976; received Master of Science degree in Computer Science from Oklahoma State University in December 1981; Completed requirements for the Doctor of Philosophy degree at Oklahoma State University in May 1989.

Professional Experience: Computer Instructor, ISIRAN Institute, Tehran, Iran; Graduate Teaching Assistant; Department of Computer and Information Sciences, Oklahoma State University, 1979-1984 Assistant Professor; Kearney State College, 1984-1986; Instructor; University of New Haven, 1986-1988; Assistant Professor; Southern Connecticut State University; 1988-present.