CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

VLSI SYSTOLIC ARRAY ARCHITECTURE FOR THE COMPUTATION OF THE DISCRETE FOURIER TRANSFORM

by

Jean-Angelo Beraldin

A thesis
presented to the University of Ottawa
in partial fulfillment of the
requirements for the degree of
Master of Applied Sciences
in
Electrical Engineering

OTTAWA, Ontario, 1986

ISBN 0-315-33281-6

The University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign, below, and give address and data.

ABSTRACT

Discrete Fourier transform systolic array realizations by means the modified Goertzel algorithm are described in this thesis. The new algorithm which eliminates the need for a reverse data sequence and external hardware overhead yields efficient systolic array mappings. The two systolic arrays introduced, a semi and pure-systolic array, are characterized by regular, modular cell interconnections, thus making the arrays compatible with VLSI design guidelines. The two arrays require a total of N identical basic cells. Each basic cell communicates only with its nearest neighbors. A cell is built around some simple registers, four adders and four multipliers. The fact that only one static twiddle factor is needed per cell implies that a multiplier can be replaced by a stored product ROM. The resulting systolic arrays have an effective throughput rate of one DFT coefficient per clock cycle. The arrays are compared to other DFT systolic arrays regarding complexity and real-time implementation. A theoretical analysis and computer simulations are carried out to predict the error performance of the proposed systolic arrays and other existing systolic arrays.

ACKNOWLEDGEMENTS

The author wishes to express his most sincere appreciation and gratitude to his supervisor, Dr. Willem Steenaart for his guidance and encouragement in this research work.

Financial assistance obtained for this research work from the National Sciences and Engineering Research Council of Canada and the University of Ottawa is gratefully acknowledged.

The author wishes to acknowledge the help of Mrs. Tyseer Aboulnasr for constructive technical review.

Also acknowledged is the assistance provided by Computer Innovations, Sherway Gardens, Etobicoke for the printing of this thesis.

Finally, the author wishes to thank his wife Antonietta for her constant encouragement during his studies.

CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER I

INTRODUCTION

Fourier analysis is a useful tool to describe physically realizable time-domain waveforms in terms of their frequency content. This analysis technique is composed of two different tools, the Fourier series and the Fourier integral. The first one applies to periodic time functions and the latter to aperiodic time functions. The conditions under which the Fourier series or the integral exist are given by the Dirichlet conditions [14].

The use of such analysis tools dates back in the late eighteenth century, though originally not named after J.B.J.Fourier (1768-1830) [1]. To name a few applications, D.Bernouilli (1700-1782) expressed the form of a vibrating string as a series of sine and cosine terms with arguments of both time and distance in 1753 and J.B.J. Fourier in 1822 (year of publication) showed how a mathematical series of sine and cosine terms could be used to analyse heat conduction in solid bodies. In these examples the analysis was carried out on continuous time functions. But Fourier analysis can also accommodate sampled waveforms. The technique is known as the discrete Fourier transform DFT [2]. Efficient methods of computing

the DFT were proposed by C.F.Gauss in 1805 for the interpolation of orbits of celestial bodies and by F.Carlini in 1828 for harmonic analysis of barometric pressure. More historical notes can be found in [3] and the references mentioned in the article.

More recently with the advent of digital computers in the late forties, there has been widespread interest in Fourier analysis in signal processing applications in areas such as communications, radar, sonar, seismology and medical imaging systems [4]. Incontestably the DFT is a key algorithm in frequency-domain description of signals and linear system functions. Scientists and engineers have focused their attention mainly in finding efficient ways to calculate the discrete Fourier transform on general computers (so-called Von Neumann machines). In a way, as mentioned in [3], some of these computer algorithms were reinvented. The most popular family of algorithms to evaluate the DFT on a general purpose computer is known as the fast Fourier transform (FFT). The first one was published in the mid-sixties and is called the Cooley-Tuckey FFT [5]. This computer algorithm requires a number of complex operations proportional to $N \log_2 N$ instead of N^2 for the computation of the DFT. This algorithm was certainly a breakthrough for the evaluation of a DFT on machines equipped with a single arithmetic logic unit. Other FFT

algorithms are like the Sande-Tuckey , prime-factor and Winograd FFT algorithms [14-19].

Today,with the advancement in microelectronics particularly in very large scale integration (VLSI) technology, a number of elementary processors with a large memory capacity can fit on a single chip [6-9]. As this trend progresses distributed-processing and pipelined processing become attractive solutions to overcome the single processor bottleneck [10,11]. For example, memory chips with a large number of stored bits, thought not to be realizable a few years ago, can now help simplify arithmetic operations (stored product techniques) [12]. This attractive solution for arithmetic operations has become particularly feasible with a recent market decline in the world memory price [13].

In chapter two of this thesis, we review the main characteristics of the discrete Fourier transform. Linear filtering computation of the DFT is then explored. Some basic algorithms like the Goertzel , the Curtis-Goertzel [22], the Ward-Goertzel [24], the chirp-z transform and Rader's algorithms are reviewed. Next, the most popular fast Fourier transform algorithms like the Cooley-Tuckey FFT [5,15], the prime factor algorithm (PFA) [30,32] and the Winograd Fourier transform algorithm (WFTA) [31] are reviewed. An extension of the DFT concept is also discussed, by means of the sliding and the delta rotation DFTs.

We are not trying to give an exhaustive tutorial on the DFT computation but rather to outline the major attributes of the algorithms in terms of the number of operations, hardware requirements and communication costs for data transfers. These characteristics will help in finding algorithms that are amenable to VLSI implementations.

To pursue this goal, we present in chapter three a summary of VLSI architectures. The chapter starts with a discussion of existing MSI/LSI implementations of DFT processors. Problems that could be found by translating these processors into VLSI designs are given. Then, opportunities created by VLSI technology are explored. The limitations and some basic guidelines in VLSI designs are presented. Multiprocessor systems which can benefit from VLSI technology are described. General distributed signal processing systems and pipeline processors are covered. Emphasis is placed on a special kind of pipeline processor that is the systolic array architecture. Systolic arrays are favored for a DFT processor implementation mainly because of their simplicity and their compatibility with VLSI design guidelines.

In chapter four, we further concentrate our study to those architectures intended to implement the discrete Fourier transform as a systolic array [38-39,49,51,53-54].

Subsequently, we introduce an improvement to the systolic implementation of the DFT by means of the modified first-order Goertzel algorithm [66]. Two types of systolic arrays are introduced a semi and a pure systolic array. Chapter four ends with a discussion concerning hardware implementations. A thorough design based on off-the-shelve integrated circuits built with VLSI down to SSI technologies is considered.

We give in chapter five, for the proposed arrays and the other DFT systolic arrays in general, a fixed-point error analysis. The analysis reveals both the possibilities and limitations of the use of DFT systolic arrays. Finally, chapter six concludes the topic of this thesis.

The original contributions of this thesis research are the development of a semi and a pure-systolic array for the computation of the discrete Fourier transform by means of the modified Goertzel algorithm, an improvement to the first order Goertzel algorithm called the modified Goertzel algorithm, and a statistical error analysis of the modified Goertzel algorithm.

Other contributions, partly original, include the use of stored product ROMs for the proposed systolic arrays, an error analysis for this realization and a note concerning overflow problems found in a fixed point realization of the second order Goertzel algorithm.

CHAPTER II

OVERVIEW OF DISCRETE FOURIER TRANSFORM COMPUTATIONAL ALGORITHMS

2.1) Introduction:

The object of this chapter is to present first the discrete Fourier transform and its properties. Subsequently computational algorithms for the DFT are reviewed. To permit a better understanding of their properties, the algorithms are grouped in four categories: linear filtering algorithms (section 2.3), fast transform-like algorithms (section 2.4), sliding DFT algorithm (section 2.5) and delta rotation algorithm (section 2.6). For each algorithm the equation is derived and an analysis of the number of operations for complex input data is given. Here we want to assess the performance of each method in terms of the number of arithmetic operations if computed on a general purpose computer, and the number of hardware multipliers and adders, together with the need for data exchange that would be required if the algorithms were mapped into fully parallel architectures. Finally, we end this chapter with a conclusion (section 2.7).

2.2) The discrete Fourier transform:

The discrete Fourier transform is the result of the adaptation of the Fourier integral to a form which is appropriate for computation on digital computers. Many text books give the derivation and properties of the DFT [2,14-19]. Here our primary objective is to state the properties of the DFT and explain the notation that will be used throughout this thesis.

The discrete Fourier transform of a finite duration discrete-time signal is given by

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp\{-j2\pi nk/N\} \quad , 0 \le k \le N-1 \qquad (2.1)$$

and where,

x(n) represents the discrete-time signal(small letter) after sampling a continuous-time signal at equally spaced time intervals (can be complex),

N is an integer number of samples and

X(k) represents the discrete-frequency transform function, the spectral lines.

This analysis transform (2.1) has its inverse function called the synthesis transform.

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) \exp\{j2\pi nk/N\} \quad , 0 \le n \le N-1 \qquad (2.2)$$

The complex exponential exp{-j2π/N} will in the following be

replaced by W_N for simplicity.

Mathematical properties of the DFT

Mathematically the DFT as an analysis and synthesis tool possesses several properties which are very useful when applied in digital signal processing. The notation used in the following is

$$\{x(n)\} \longleftrightarrow \{X(k)\} \tag{2.3}$$

This states that the discrete-time sequence x(n) and the discrete-frequency sequence X(k) are a DFT pair. We also introduce another discrete-time sequence, y(n), and its transform Y(k); a constant "a" that can be any number and an integer number "l".

Linearity

$$\{x(n)\} + \{y(n)\} \longleftrightarrow \{X(k)\} + \{Y(k)\} \tag{2.4}$$

$$\{a\,x(n)\} \longleftrightarrow \{a\,X(k)\} \tag{2.5}$$

Symmetry

$$\{x(-n)\} \longleftrightarrow \{X(-k)\} \tag{2.6}$$

Time shifting

$$\{x(n+l)\} \longleftrightarrow \{W_N^{-lk}\,X(k)\} \tag{2.7}$$

Frequency shifting

$$\{W_N^{ln}\,x(n)\} \longleftrightarrow \{X(k+l)\} \tag{2.8}$$

Convolution

$$\{\sum_{n=0}^{N-1} x(n)y(l-n)\} \longleftrightarrow \{X(k)Y(k)\} \tag{2.9}$$

DFT of a real sequence

The DFT of a real sequence is characterized by a real part that has even symmetry and an imaginary part that has odd symmetry. Therefore, when computing the DFT, it is not necessary to compute N frequency samples, only N/2 are required.

It is possible with one complex DFT to compute two real DFTs simultaneously. Let assume

$$z(n)=x(n)+jy(n) \qquad (2.10)$$

where x(n) and y(n) are both real sequences. Using the linearity property of the DFT (2.4), we find

$$x(n)+jy(n) \quad <----> \quad X(k)+jY(k) \qquad (2.11)$$

and

$$Z(k)=X(k)+jY(k) \qquad (2.12)$$

We need to express X(k) and Y(k) in Z(k); the following equations are found [15,17-18] :

$$X(k) = (Z(k) + Z(-k))/2 \qquad (2.13)$$

$$Y(k) = (Z(-k) - Z(k))/2 \qquad (2.14)$$

In an application context we must keep in mind some DFT characteristics, both functions x(n) and X(k) are periodic with period N. Therefore, (2.6) should be interpreted as a mapping of the functions indices "n" or "k" into N-n or N-k (except at zero). The zeroth sample is mapped into itself.

2.3) Linear filtering computation of the DFT:

2.3.1) Generalities:

In this section, the major algorithms for computing the DFT as a linear filtering process are summarized. Each algorithm is explained using a basic equation and a diagram for a sequence of length N. The number of arithmetic operations (add,multiply) required when the DFT is computed on a general purpose computer, and also the number of hardware adders and multipliers for a fully parallel system realization i.e. a system where the number of arithmetic units is proportional to N are given.

2.3.2) Goertzel algorithm:

2.3.2.1) First order recursive filter:

The Goertzel algorithm is aimed in reducing the number of operations on a general purpose computer and is a way to find the DFT of a sequence at a specific frequency sample [14,20-21].

Lets multiply the right hand side of (2.1) by the constant $W_N^{-kN}=1$,

$$X(k)= W_N^{-kN} \sum_{n=0}^{N-1} x(n)\, W_N^{kn} \qquad (2.15)$$

$$X(k)= \sum_{n=0}^{N-1} x(n)\, W_N^{-k(N-n)} \qquad k=0,1,\ldots,N-1 \qquad (2.16)$$

This is similar to a discrete convolution (2.9). The left

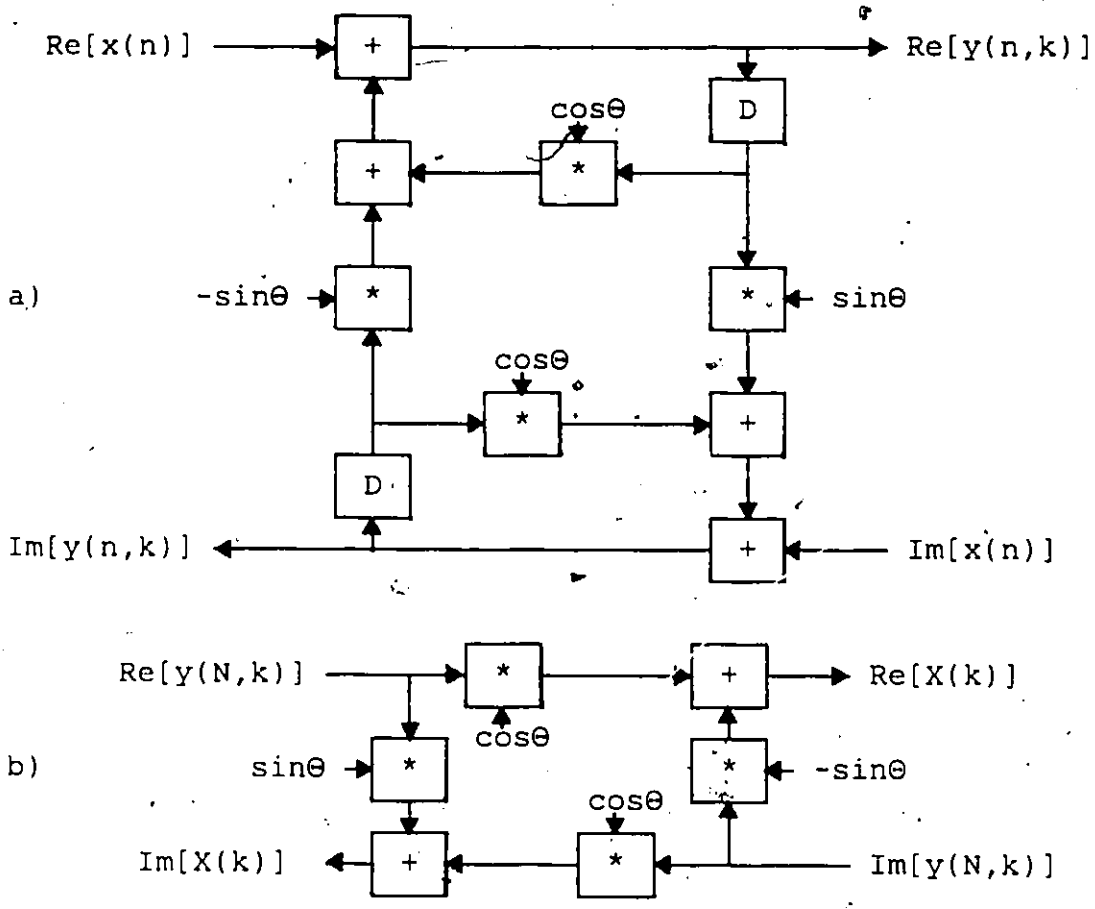
hand side of (2.16), X(k), can be replaced by the variable y(n,k), which indicates the n^{th} recursive step and the k^{th} frequency sample. Therefore, for one particular value of k, we have the following convolution equation

$$y(n,k) = \sum_{n=0}^{N-1} x(n)\ W_N^{-k(i-n)} \qquad (2.17)$$

and the frequency sample is available after the N^{th} iteration

$$X(k) = y(n,k)\big|_{i=N} \qquad (2.18)$$

The algorithm stated by (2.17) and (2.18) is known as the first order Goertzel algorithm. A system to compute (2.17) at a particular frequency sample "k" is depicted in figure 2.1. When we say X(k) is found after the N^{th} iteration, it really means that the partial result y(N-1,k) must be fed back in the network and a zero sample sent at the input. Therefore to satisfy (2.1) two frames must be separated with one data sample equal to zero. To get around this inconveniency, one could use a complex multiplier at the output thus allowing a continuous flow of data frames to be transformed as shown on figure 2.1.



where in the boxes, the symbol
"+" represents an arithmetic addition,
"*", an arithmetic multiplication,
"D", a time delay.

and

$\Theta = 2\pi k/N$

Figure 2.1. Arithmetic cell for the realization of the first order Goertzel algorithm, (a) recursive section and (b) phase correction section.

If complete transforms of length N in a continuous fashion are computed using N arithmetic cells as in figure 2.1, the resulting system would need a total of 6N adders and 8N multipliers. For example, such system can be

configured as follows: each arithmetic cell is connected to a data bus, every clock cycle an input sample is made available on the bus, this input sample is grabbed by each cell, each one computes a frequency sample and after N clock cycles the frequency samples are ready for retrieval.

On the other hand a software approach on a general purpose computer would require 4N real multiplications and 4N real additions for the computation of a DFT sample at a particular "k" plus two real additions and four real multiplications every N iterations. The total number of operations are (4N+2)N real additions and (4N+4)N real multiplications. This result is a bit more than a direct calculation (2.1) but the Goertzel algorithm doesn't require the computation of, nor the storage of, all the coefficients W_N^{kn} since these quantities are effectively computed by the recursion process implied in figure 2.1 [14].

2.3.2.2) Second-order recursive filter:

It is possible to transform the first-order system into a second-order recursive system [14,21]. To see this, first write the transfer function in complex form for the flow graph of figure 2.1.

$$H(z,k) = \frac{1}{1-W_N^{-k}z^{-1}} \qquad (2.19)$$

Secondly, multiplying both the numerator and the denominator of H(z,k) by the factor $(1-W_N^{k}z^{-1})$, we obtain

$$H(z,k) = \frac{1-W_N^k z^{-1}}{1-2\cos(2\pi k/N)\, z^{-1}+z^{-2}} \qquad (2.20)$$

Finally (2.20) can be represented by a flow graph as in figure 2.2.

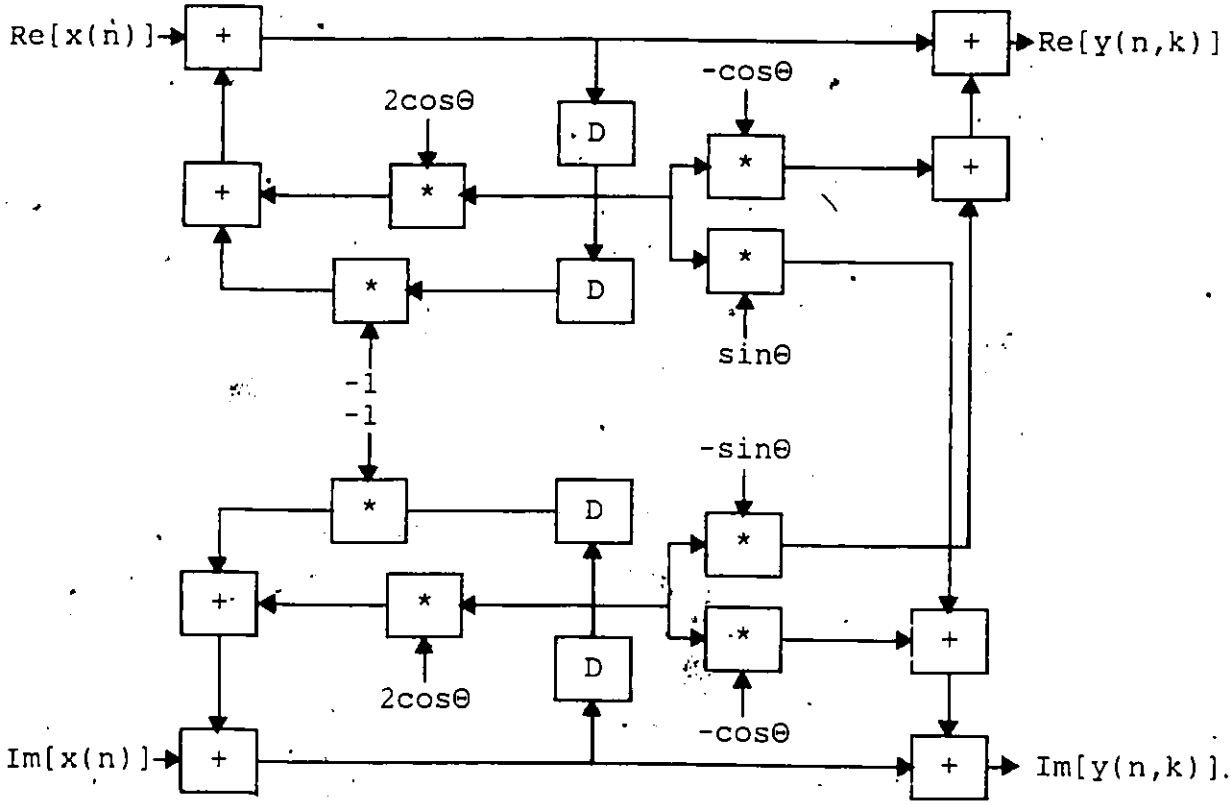


Figure 2.2.Flow graph of an arithmetic cell for the second order realization of the Goertzel algorithm [14].

The execution of the algorithm on a general purpose computer would require the computation of 2N(N+2) real multiplications and 4N(N+1) real additions. For a system implemented with N arithmetic cells like those of figure 2.2, a total of 6N multipliers and 8N adders are

required. This system would work as the one mentioned in section 2.3.2.1.

The major advantage of the second order approach over the first order is found in the increased accuracy i.e. only two multipliers ($2\cos\Theta$) are required in the recursive loop instead of four. This topic is covered in details in chapter 5.

2.3.3) Recursive prime-radix algorithm transforms:

2.3.3.1) Curtis-Goertzel algorithm:

The Goertzel algorithm uses N twiddle factors i.e. W_N^{-k} for k=0,1,...,N-1, in the first-order recursive realization (2.17) or N real coefficients, $2\cos(2\pi k/N)$, in the second-order recursive realization (2.20), for the computation of the DFT. A method is proposed in [22] to calculate the DFT using a recursive approach like the Goertzel algorithm. But this recursive "prime-radix algorithm transform" (PRAT) reduces the number of different coefficients to one, i.e. W_N.

Consider the N-point DFT, with N a prime number:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{kn} \qquad 0 \le k \le N-1 \qquad (2.1)$$

"Since N is prime, all N values of the complex exponent are used for each frequency cell calculation, i.e. for every value of k, except the first. This is not the case when N is composite. Consequently, it is possible to reorder "k" so that sequential values of the exponent increment in phase by $\{2\pi M/N\}$, with (M) MOD N $\neq 0$, rather than using sequential values of x(n). This does not change the process, simply the order in which it is done."[22]

For example, consider a five-point DFT given by

$$x(k) = \sum_{n=0}^{4} x(n)\, W_5^{kn} \qquad 0 \le k \le 4 \tag{2.21}$$

Expanding, removing the redundancy in the complex exponent and replacing W_5 by W gives

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 & W^4 \\ 1 & W^2 & W^4 & W^1 & W^3 \\ 1 & W^3 & W^1 & W^4 & W^2 \\ 1 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix} \tag{2.22}$$

Ignoring the d.c. term X(0), expanding and refactoring to use the complex coefficients in order gives

$$\begin{bmatrix} X(1) \\ X(2) \\ X(3) \\ X(4) \end{bmatrix} = \begin{bmatrix} x(0) & x(1) & x(2) & x(3) & x(4) \\ x(0) & x(3) & x(1) & x(4) & x(2) \\ x(0) & x(2) & x(4) & x(1) & x(3) \\ x(0) & x(4) & x(3) & x(2) & x(1) \end{bmatrix} \begin{bmatrix} 1 \\ W^1 \\ W^2 \\ W^3 \\ W^4 \end{bmatrix} \tag{2.23}$$

Referring to (2.22), it can be seen that all the coefficients, 1 through W^4, are used for each frequency cell calculation, except X(0) which is treated separately as a special case. Ignoring this term, the DFT once expanded and refactored to use the complex coefficients sequentially appears as in (2.23).

Equation (2.23) can be written in a more compact form as

$$X(k) = \sum_{n=0}^{4} W^n\, x[(nk^* (\text{MOD } 5))] \qquad 1 \le k \le 4 \tag{2.24}$$

where k^* is given by the congruence $kk^* \equiv 1 \text{ MOD } 5$ [18].

Or, for the N-point transform,

$$X(0) = \sum_{n=0}^{N-1} x(n) \qquad (2.25)$$

$$X(k) = \sum_{n=0}^{N-1} W^{Mn}\, x[(nk^*(\text{MOD } N))] \quad 1 \le k \le N-1 \qquad (2.26)$$

where k^* is given by the congruence $kk^* \equiv M \text{ MOD } N$. (2.26) can now be computed recursively, as can be seen by considering the general form of the k^{th} cell of the five-point DFT used above:

$$X(k) = \sum_{n=0}^{4} x'(n)\, W^n \qquad (2.27)$$

where $x'(n)$ is the shuffled version of $x(n)$ mapped as defined in (2.24) above. Expanding (2.27) and writing it in reverse order gives

$$X(k)=x'(4)W^4+x'(3)W^3+x'(2)W^2+x'(1)W^1+x'(0) \qquad (2.28)$$

Rewriting (2.28) as a polynomial evaluation using Horner's rule [21] results in

$$X(k)=(((x'(4)W+x'(3))W+x'(2))W+x'(1))W+x'(0) \qquad (2.29)$$

Consequently, the evaluation of a DFT using the algorithm described above uses a structure identical to that of the Goertzel algorithm of section 2.3.2. It can be represented either as a first-order recursive system (figure 2.3) or as a second-order recursive system (figure 2.4). The major differences being the fact that only one complex coefficient is required for any frequency sample calculation

and that for each of these frequency sample, a proper data reordering (data shuffling) is required.

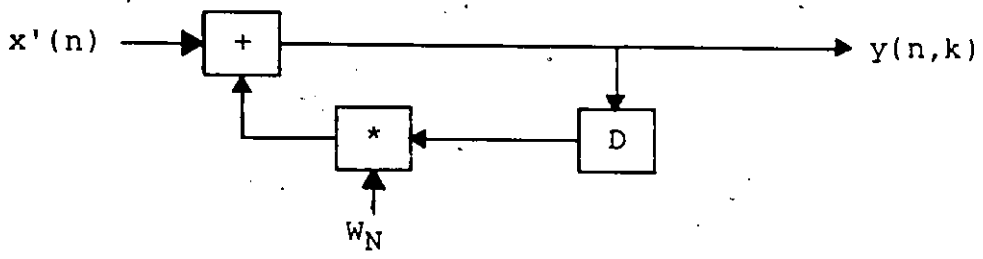


Figure 2.3 First-order recursive system for PRAT-Curtis algorithm [22].

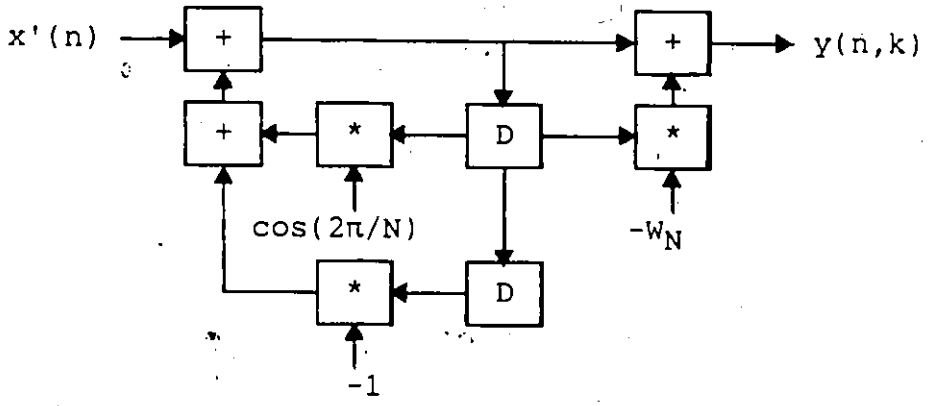


Figure 2.4 Second-order recursive system for PRAT-Curtis algorithm [22].

To go even further in reducing the complexity of the algorithm, one can see that X(k) could be calculated by loading the initial value of data,x'(4),into an accumulator, rotating the accumulator output through an angle of $-2\pi/5$ and adding the next data value,x'(3). The sum is again rotated by $-2\pi/5$ and x'(2) is added. This process iterates until all values of x'(n) have been processed. These rotations of the accumulator can be interpreted as shifting operations (i.e.multiplication by 2^{-n}). The value of M

(2.26),the phase iteration factor,for a given transform size N is chosen to provide scaling values that can be implemented easily. Convenient values of M, for some N (prime) in the range 3 to 47 to give cos(2πM/N) approaching 2^{-n} are listed in Table 2.1.

TABLE 2.1 Convenient values for M and scaling factors for primes in the range 3-47 [22].

N	M	cos(2πM/N)	Shift scaling
3	1	-0.5000	-2^{-1}
5	1	0.3090	$2^{-2}+0.9443*2^{-4}$
11	3	-0.1423	$-2^{-3}-0.5541*2^{-5}$
19	4	0.2455	$2^{-2}-0.5779*2^{-7}$
31	9	-0.2507	$-2^{-2}-0.6682*2^{-10}$
47	1	0.9911	$2^{0}-0.5710*2^{-6}$

Table 2.1 lists only two transform sizes with scaling values that can be implemented by simple binary shift. These are N=3 and N=31. The three-point transform is of limited practical application, unless highly composite data block sizes are used. The 31-point transform is useful in a number of applications, but using an approximate scaling value in the recursion introduces errors in the transform calculation [22].

An algorithm to calculate DFTs by recursive filtering has been outlined. The recursive PRAT algorithm uses a

special index mapping for shuffling the input data to reduce the total number of complex coefficients to only one. This results in an add/rotate operation which can be simplified to an add/shift operation with a proper choosing of N a prime. This technique has been employed as an alternate low-level primitive structure [22-23]. A comparison to the Goertzel algorithm will be presented in section 2.3.6.

2.3.3.2) Ward-Goertzel algorithm:

Shortly after the publication of [22],Ward published a letter that showed how the number of multiplications could be halved in a sequential calculation of complex DFTs while still retaining the recursive nature of the Curtis-Goertzel algorithm [24].

The new technique changes the basis of the complex plane to $(1,W_N^M)$, where W_N is the N^{th} root of unity; rather than the more normal (1,j). This results in the following number system applies again to N a prime.

$$R(W) = \{a+bW\} \tag{2.30}$$

where a,b are real numbers and $W=W_N^M$. Arithmetic in R(W) is performed using the following set of rules:

$$(a+bW) + (c+dW) = (a+c) + (b+d)W \tag{2.31}$$

$$(a+bW) * (c+dW) = (ac-bd) + (bc+ad+2bd\ Re\{W\})W \tag{2.32}$$

where $W=Re\{W\} + j\ Im\{W\}$.

Transformations between complex numbers, and numbers

represented in R(W),denoted by

$$\text{complex:}a+jb \text{ ----> } R(W):c+Wd \tag{2.33}$$

are given by

$$c=a-b(\mathrm{Re}\{W\}/\mathrm{Im}\{W\}) \text{ and } d=b/\mathrm{Im}\{W\} \tag{2.34}$$

Similarly the inverse transformation is

$$a=c+d\ \mathrm{Re}\{W\} \quad \text{and} \quad b=d\ \mathrm{Im}\{W\} \tag{2.35}$$

Suppose the arithmetic used to evaluate (2.29) is conducted in R(W), so that multiplication of W correspond to multiplying by (0+1W). Then using (2.32) each of the partial sums in (2.29) may be evaluated by one multiplication, even for complex input data.

$$(a+bW)W = (-b+(a+2b\ \mathrm{Re}\{W\})W) \tag{2.36}$$

$$2\mathrm{Re}\{W\} = 2\cos(2\pi M/N) \tag{2.37}$$

By making use of these results, the first-order recursive filter at particular frequency sample "k" is illustrated in figure 2.5 for complex input data. The same restrictions as in the previous section apply.

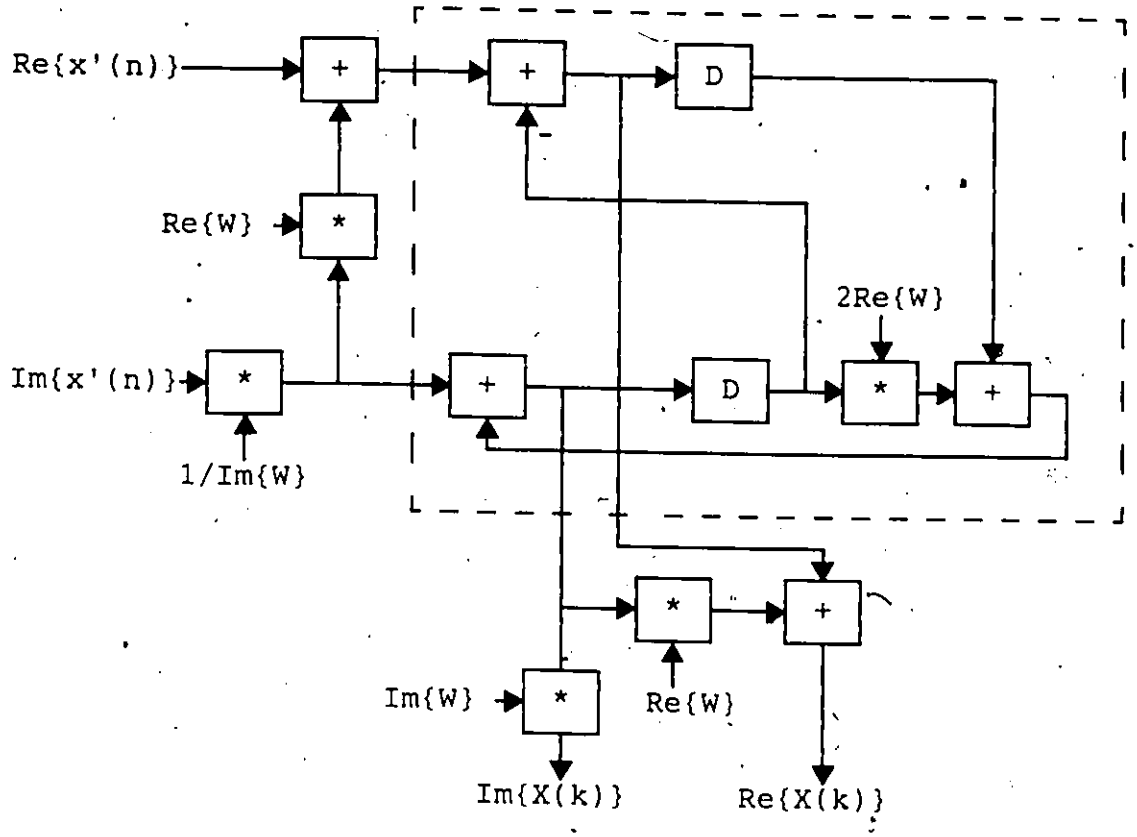


Figure 2.5 First-order recursive system for PRAT-Ward algorithm [24].

The region within the dotted lines indicate where the computation in R(W) are performed. The section at the left-hand side of the dotted rectangle does a conversion complex domain ----> R(W) using (2.34) and the section below the dotted rectangle, a conversion R(W) ----> complex domain using (2.35).

2.3.4) Chirp-z transform:

Consider the DFT of a sequence x(n)

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

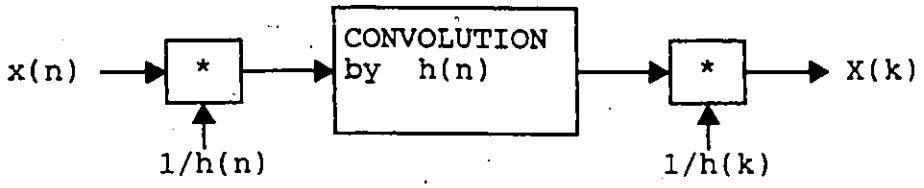
Rearrange the exponent "nk" to obtain:

$$nk = -(k-n)^2/2 + n^2/2 + k^2/2 \qquad (2.38)$$

Thus (2.1) becomes

$$X(k) = W_N^{k^2/2} \sum_{n=0}^{N-1} x(n)\, W_N^{n^2/2}\, W_N^{-(k-n)^2/2} \qquad (2.39)$$

This shows that X(k) may be computed by convolving the sequence x(n) premultiplied by a complex exponential function with that same function, and by postmultiplying by a similar function with exponent k^2. This process is depicted in figure 2.6. The major part of the algorithm is the complex finite impulse response (FIR) filter; its impulse response is that of a chirp filter, which is well known in radar signal processing; hence the name chirp-z transform. The method can also be applied to the evaluation of the z-transform [25-26]. The DFT is simply a special case of the algorithm when the radius is set to unity and the initial rotation angle to zero [14,18].



where $h(n) = W_N^{n^2/2}$

Figure 2.6 DFT computation using the chirp-z algorithm [26].

The computation of the algorithm on a general-purpose computer requires N(4N+8) multiplications and N(2N+4) additions. On the other hand, the implementation of a chirp-z transform using parallel arithmetic would need (4N+8) multipliers and (2N+4) adders. These last figures assume that the FIR filter is implemented with N arithmetic units i.e. one for each tap.

2.3.5) Rader's algorithm:

Another method for converting a DFT into a convolution was proposed by Rader [28]. This method is in many aspects computationally more efficient than the chirp-z transform algorithm. Rader's algorithm replaces the premultiplications and postmultiplications in the chirp-z transform algorithm by a rearrangement of the input and output data samples. We shall consider here the case of a DFT of size N where N is a prime. More information in [18,28] can be found on the polynomial formulation of Rader's algorithm.

We rewrite (2.1) into a different format.

For k=0, X(k) is given by

$$X(0) = \sum_{n=0}^{N-1} x(n) \qquad (2.43)$$

for k≠0,

$$X(k) = x(0) + \sum_{n=1}^{N-1} x(n)\, W_N^{nk} \qquad 1 \le k \le N-1 \qquad (2.44)$$

The indices "n" and "k" are defined modulo N. It can be shown that, if "u" is the set of integers 0,1,...,N-2 ,there are always primitive roots "g" defined modulo N such that g^u modulo N takes once and only once all the values 1,2,...,N-1 when "u" takes successively the values 0,1,...,N-2 [18]. Thus, for n, k≠0, we can replace "n" and "k" by "u" and "v" defined by

$$n \equiv g^u \text{ modulo } N$$

$$k \equiv g^v \text{ modulo } N \,, \qquad u,v = 0,1,\ldots,N-2 \qquad (2.45)$$

Under these conditions,(2.44) becomes

$$X(g^v) = x(0) + \sum_{u=0}^{N-2} x(g^u)\, W_N^{\,g^{u+v}} \qquad (2.46)$$

which shows that the summation term is computed as a circular correlation of the permuted data sequence $x(g^u)$ with the complex exponential W_N raised to the power g^u. Thus, for N an odd prime, most of the computation required to evaluate a DFT of N samples reduces to a circular correlation of N-1 points. The process is illustrated in figure 2.7.

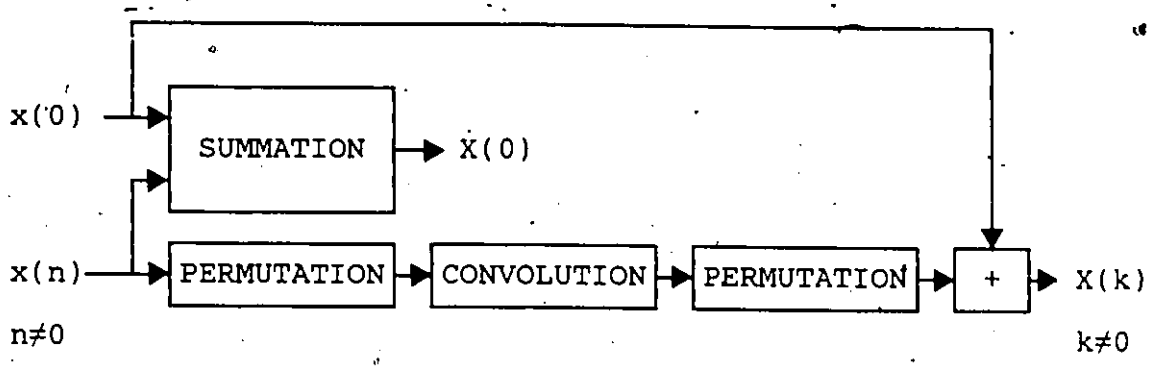


Figure 2.7 Computation of a N-point DFT by Rader's algorithm ,N a prime [18].

It is not easy for this algorithm to determine the number of components. This figure depends on how the circular correlation is implemented. Below, we reproduced a table found in [18] which gives the number of multiplications and additions for the sequential computation of the algorithm when N is small i.e. N < 16.

Table 2.2 Number of non-trivial real operations for short DFTs computed by Rader's [18].

DFT size	Number of multiplications	Number of additions
2	2	2
3	3	6
4	4	8
5	6	17
7	9	36
8	8	26
9	11	44
16	18	74

The usefulness of Rader's algorithm emerges when it is combined with other techniques like the prime factor algorithm (section 2.4.3), to compute large DFTs.

2.3.6) Discussion:

To summarize this section, it is worthwhile to gather all the characteristics of the linear filtering DFT algorithms covered in section 2.3. In table 2.3 for each algorithm ,the number of additions/multiplications required on a general-purpose computer and the number of adders/multipliers required for a parallel architecture are given. Furthermore, some important remarks are also given. The figures shown are valid only for a full length transform of size N and for complex input data.

For the general purpose computer approach, one can verify that, in general, the number of operations is proportional to N^2 or simply $O(N^2)$. It is easy to decide which of these algorithms is the most efficient. The criterion are the number of multiplications and additions weighted by the execution time for each one. A second criterion must be taken into account that is, the way in which the twiddle factors are generated or stored. This criterion refers to systems with limited storage facilities.

On the other hand, for a parallel system point of view, the number of arithmetic components is O(N). In this case, it is not a simple task to choose one algorithm among those that were studied. The number of arithmetic components is important as well as the way in which the

twiddle factors are generated and stored. Above all, the communication requirements for data exchanging and routing must be considered. This topic is crucial with parallel system implementations and is covered in section 3.3.

Table 2.3 Comparison of different linear filtering algorithms intended for the DFT, the input samples are complex.

Algorithm	General-purpose computer			Parallel system number of PE=O(N)			Limits on N	Equation
	mult.	add	Note	mpy's	adders	Note		
Direct method	$4N^2$	$4N^2$	1	4N	4N	1	Any N	(2.1)
Goertzel 1st order	$4N^2+4N$	$4N^2+2N$	2	8N	6N	3	Any N	(2.17) (2.18)
2nd order	$2N^2+4N$	$4N^2+4N$	2	6N	8N	3	Any N	(2.20)
PRAT Curtis	$2N^2+4N$	$4N^2+4N$	4	6N	8N	5	Prime number	(2.26)
Curtis-add/shift	2N $2N^2+2N$	$4N^2+4N$	4,8	2N 4N-SR	8N	5,8	Prime number	(2.26)
Ward	N^2+4N	$3N^2+2N$	4	5N	5N	5	Prime number	(2.26) (2.36-37)
Chirp-z Direct	$4N^2+8N$	$2N^2+2N$	6	4N+8	2N+4	6	Any N	(2.39)
Rader's Direct	see Table 2.2		7	N/A		7	Prime number	(2.46)

All the figures are given in terms of real operations

PE : Processing Element
mult.: Multiplication
mpy's : Multipliers
O(N) : Proportional to N
SR : Shift register
N/A : Not Applicable

Notes:

1 : Storage or generation of each coefficient W_N^{kn}

2 : Storage or generation of each coefficient W_N^{k}

3 : Need only one coefficient W_N^{k} per PE

4 : Only one coefficient needed, W_N

Requires data shuffling

5 : All the PE's are identical, except at k=0

Requires data shuffling

6 : Storage or generation of each coefficient $W_N^{-n^2/2}$

The number of operations and elements depend on how the convolution is implemented.

7 : Need data shuffling

The number of operations and elements depend on how the convolution is implemented.

8 : N=3 and 31 yield the best performance for an add/shift implementation.

2.4) Fast transform-like algorithms for the discrete Fourier transform:

2.4.1) Generalities:

In this section, we focus our attention on so-called fast transform-like algorithms for the DFT. These algorithms outperform any DFT algorithms when computed on a general purpose computer. Thus, the motivation for these algorithms is the number of operations (add/multiply) needed for a particular algorithm.

The FFT-like algorithms derive their computational advantage by transforming a DFT into a set of low-order ones. This transformation is achieved by a pair of one-dimensional to multidimensional index mappings on the input and output sequences. Two classes of index mappings have been developed for this purpose, and the corresponding algorithms have been referred to as the common factor and prime factor algorithms (CFA and PFA)[29]. The CFA, which is known as the Cooley-Tuckey FFT [5] is certainly the most popular FFT algorithm available. The prime-factor index mapping due to Good [30] and the low-order DFT algorithms due to Winograd, "Winograd Fourier transform algorithm (WFTA)" [31] have been combined to obtain a class of efficient PFAs [32]. The WFTA requires a minimal number of multiplications.

In the section that follows, we review three basic FFT

algorithms: CFA, PFA and WFTA. The mathematical derivations have been skipped in our study to emphasize the major characteristics of these algorithms.

2.4.2) The common factor algorithm:

2.4.2.1) Radix-2 algorithm:

To illustrate the CFA, we consider the special case of N an integer power of 2, i.e.$N=2^q$. Two approaches are available. The first one is equivalent to splitting the N-point input sequence x(n) into two (N/2)-point sequences x(2r) and x(2r+1) corresponding respectively, to even and odd samples of x(n),r=0,1,...,N/2-1. This process is called "decimation in time" [5,14-19] . A second form of the CFA is obtained by simply splitting the N-point input sequence x(n) into two (N/2)-point sequences x(r) and x(r+N/2) corresponding respectively to the N/2 first samples and to the N/2 last samples. This process is called "decimation in frequency",[14-19].

Both processes imply the separation of the sequence x(n) into two distinct sequences which is in fact equivalent to replacing an N-point DFT by two DFTs of length N/2. The sequence is split into a total of $q=\mathrm{Log}_2 N$ stages. This CFA is known as the Radix-2 algorithm, as derived in [18].

The process of the decimation in time radix-2 FFT algorithm is best illustrated by the flow graph of figure

2.8. The decimation in frequency flow graph is depicted by figure 2.9.

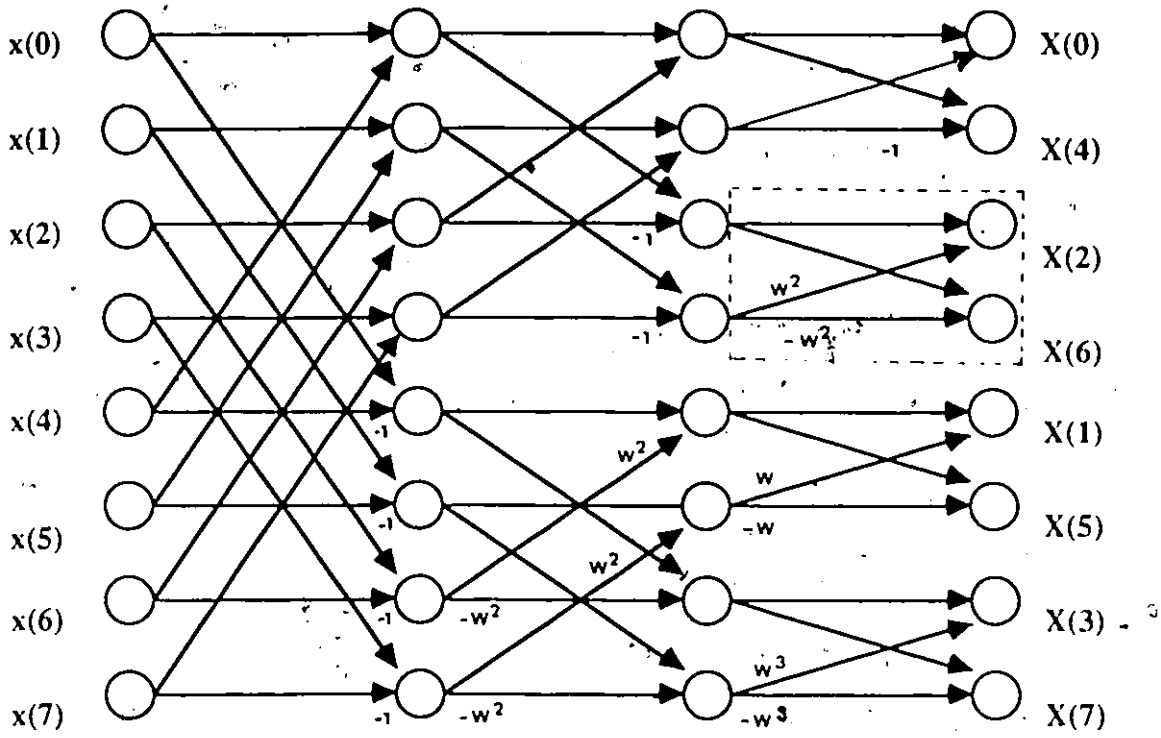


Figure 2.8 Decimation in time radix-2 FFT algorithm signal flow graph, N=8.[18].

A parallel system is formed by implementing every node of the network or similarly every butterfly (section of figure 2.8 enclosed in dotted lines). A software implementation on a general purpose computer would perform this butterfly operation N/2 Log_2N times.

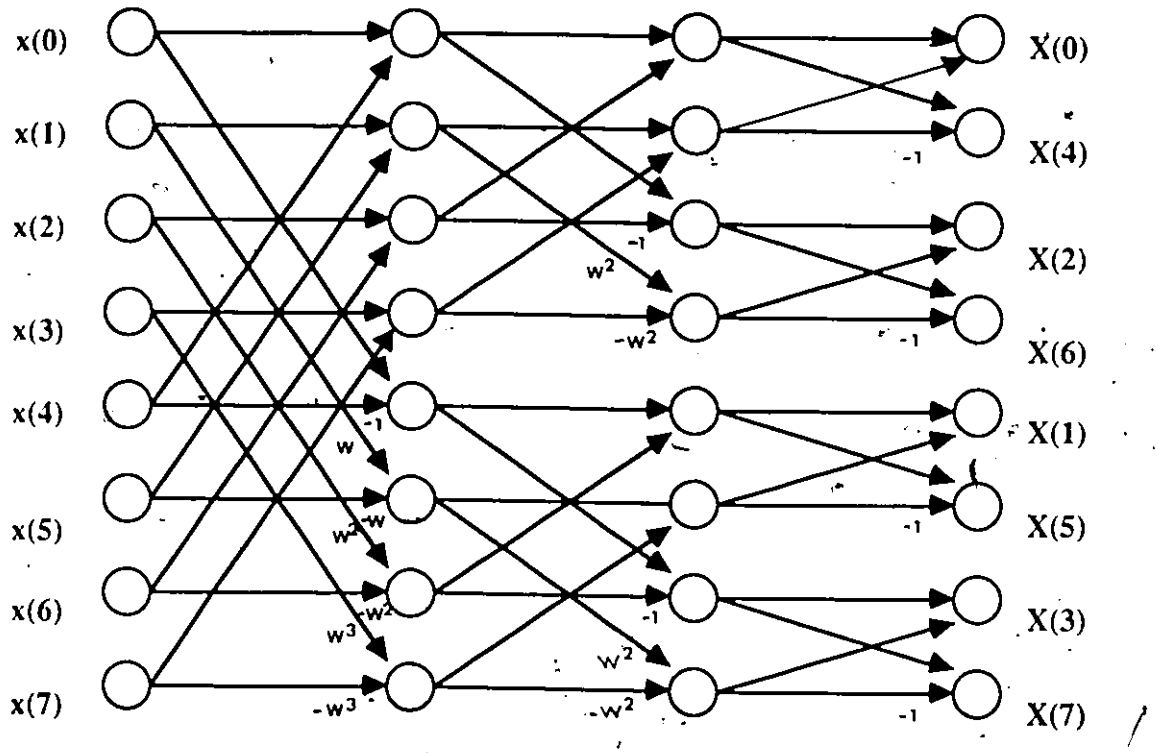


Figure 2.9 Decimation in frequency radix-2 FFT algorithm signal flow graph, N=8.[18].

In terms of the number of multiplications and additions required, reference [18] gives for both decimation techniques the following values,

Number of real multiplications: $(N/2)(-10 +3\text{Log}_2N)+8$ (2.47)

Number of real additions : $(N/2)(-10 +7\text{Log}_2N)+8$ (2.48)

These two equations apply when the complex multiplications are implemented with 3 real multiplications and 3 real additions and when the symmetries of the trigonometric

functions are fully used [18]. For a parallel implementation usually referred to as "parallel pipeline realization" [33], the number of multipliers and adders are also given by (2.47) and (2.48).

2.4.2.2) Radix-4 algorithm:

We saw that for the radix-2 algorithm, x(n) was divided in two groups. It is also possible to divide x(n) in four groups. This can be accomplished only when $N=2^q$ with q even. Therefore the N-point sequence x(n) is split into four sequences of N/4 points corresponding to x(4r), x(4r+1), x(4r+2) and x(4r+3) for r=0,1,...,N/4-1. This process is done Log_4N times. This radix-4 decimation in time algorithm converts in fact an N-point DFT into 4 DFTs of length N/4. The butterfly operation, here,recombines a set of four nodes into another set of four nodes.

The number of nontrivial real operations for the radix-4 DFTs, when a complex multiplication is implemented with 3 real multiplications and 3 real additions, and the symmetries of the trigonometric functions are fully used [18], which apply for both the parallel implementation and the implementation on a general purpose computer, are

$$\text{Number of real multiplications} : (9N/8)Log_2N - (43N/12) + 16/3 \qquad (2.49)$$

$$\text{Number of real additions} : (25N/8)Log_2N - (43N/12) + 16/3 \qquad (2.50)$$

For N greater than 16 the radix-4 algorithm significantly reduces the number of operations/components in comparison with the radix-2 algorithm. The process of splitting the sequence into a larger number of sequences e.g. radix-8, radix-16,etc... improves the figures slightly. This small decrease in operations/components is achieved at the expense of a more complex implementation.

2.4.3) The prime factor algorithm:

We have seen in section 2.3.5 that Rader's algorithm is mainly used for small DFTs (N<16). For large DFTs, the derivation of the algorithm becomes cumbersome and computationally inefficient. Another technique which is known as the prime factor FFT allows one to compute a large DFT of size N by combining several small DFTs of sizes N_1, N_2,..., N_Q which are relative prime factors of N [18]. This technique was proposed by Good [30]. The approach was shown to be of practical interest when it is combined with Rader's algorithm. Good's algorithm provides one of the foundations on which the very efficient Winograd Fourier transform algorithm is based (next section, 2.4.4).

Lets consider the case of a DFT, X(k) of size N, where N is the product of two mutually prime factors P and R

$$X(k,l) = \sum_{r=0}^{R-1} W_R^{rl} \sum_{p=0}^{P-1} x(p,r)\, W_P^{pk}$$

$$W_P = e^{-j2\pi/P}\ ,\ W_R = e^{-j2\pi/R}\ ,\ \begin{matrix} k=0,..,P-1 \\ l=0,...,R-1 \end{matrix} \qquad (2.51)$$

This illustrates that X(k,l) can be evaluated by first computing one DFT of P terms for each value r. This gives P sets of R points X(r,k) which are the input sequences to P DFTs of R points. Thus, with this method, X(k,l) is calculated with R DFTs of length P plus P DFTs of length R.

In order to evaluate the number of multiplications, M, and additions, A, along with the number of components multiplier/adder, which are necessary to perform a DFT by the PFA, we assume that M_P, M_R and A_P, A_R are the number of multiplications (multipliers) and additions (adders) required to the DFTs of lengths P and R, respectively. The results are given by:

$$M = P\, M_R + R\, M_P \qquad (2.52)$$

$$A = P\, A_R + R\, A_P \qquad (2.53)$$

In general for the case where N can be decomposed into multiple mutually prime factors N_1, N_2,..., N_q,..., we have

$$N = \pi N_q\ , \qquad (2.54)$$

where the N_q's are again mutually exclusive,

and

$$M = \Sigma\, N\, M_q/N_q \qquad (2.55)$$

$$A = \Sigma\, N\, A_q/N_q \qquad (2.56)$$

Thus, the computation of a large DFT is reduced to

that of a set of small DFTs of lengths $N_1, N_2, \ldots, N_Q$. These small DFTs can be computed with any algorithm, but it is extremely attractive to use Rader's algorithm for this application because this particular algorithm is very efficient for small DFTs.

Table 2.4 lists the number of nontrivial real arithmetic operations for various DFTs computed by the PFA and Rader's algorithm (table 2.2). A formula in this case is not easy to find; a table will do the job.

Table 2.4 Number of nontrivial real operations for DFTs computed by the PFA and Rader's algorithm [18].

DFT size N	Number of real multiplications	Number of real additions
30	100	384
48	124	636
60	200	888
120	460	2076
168	692	3492
240	1100	4812
504	2524	13388
840	5140	23172
1008	5804	29548

2.4.4) The Winograd Fourier transform algorithm:

In the preceding section, it has been shown how a composite DFT of size N, where N is the product of Q factors $N_1,\ldots,N_Q$, could be mapped, by simple index permutations, into a multidimensional DFT of size $N_1N_2\ldots N_Q$. When this multidimensional DFT is evaluated by the conventional row-column method, the algorithm becomes the prime factor algorithm. In the following, we overview another way of evaluating the multidimensional DFT. This technique is based on a nesting algorithm introduced by Winograd [31]. It is also particularly effective in reducing the number of multiplications when it is combined with Rader's algorithm [18].

We limit the study of the WFTA to the case where N is the product of two factors. One can note that the two-dimensional DFT defined by (2.51) can be regarded as a one-dimensional DFT of length R where the summation term performed over "p" is replaced by a vector times matrix product. In other words, the DFT defined by (2.51) can be expressed as

$$X(k,l) = \sum_{r=0}^{R-1} W_R^{rl}\ Y\ X_r \qquad (2.57)$$

where X_r is a P element column vector of the input data x(p,r) and Y is a P*P matrix of the complex exponential W_p^{pk}

$$X_r = \begin{bmatrix} x(0,r) \\ x(1,r) \\ \vdots \\ x(P-1,r) \end{bmatrix} \tag{2.58}$$

$$Y = \begin{bmatrix} W_P^0 & W_P^0 & W_P^0 & \cdots & W_P^0 \\ W_P^0 & W_P^1 & W_P^2 & \cdots & W_P^{(P-1)} \\ W_P^0 & W_P^2 & W_P^4 & \cdots & W_P^{2(P-1)} \\ \vdots & & & & \vdots \\ W_P^0 & W_P^{(P-1)} & & \cdots & W_P^{(P-1)^2} \end{bmatrix} \tag{2.59}$$

Thus, the polynomial DFT defined by (2.57) is a DFT of length R where each multiplication by W_R^{rl} is replaced with a multiplication by $W_R^{rl}Y$. This last operation itself is equivalent to a DFT of length P in which each multiplication by W_P^{pk} is replaced with a multiplication by $W_R^{rl}W_P^{pk}$.

The WFTA is particularly attractive when the small DFTs are evaluated via Rader's algorithm. In this case, the small DFTs are calculated with A input additions, M complex multiplications, and A^2 output additions. Therefore, the WFTA for a DFT of size P*R can be represented as shown in figure 2.10.

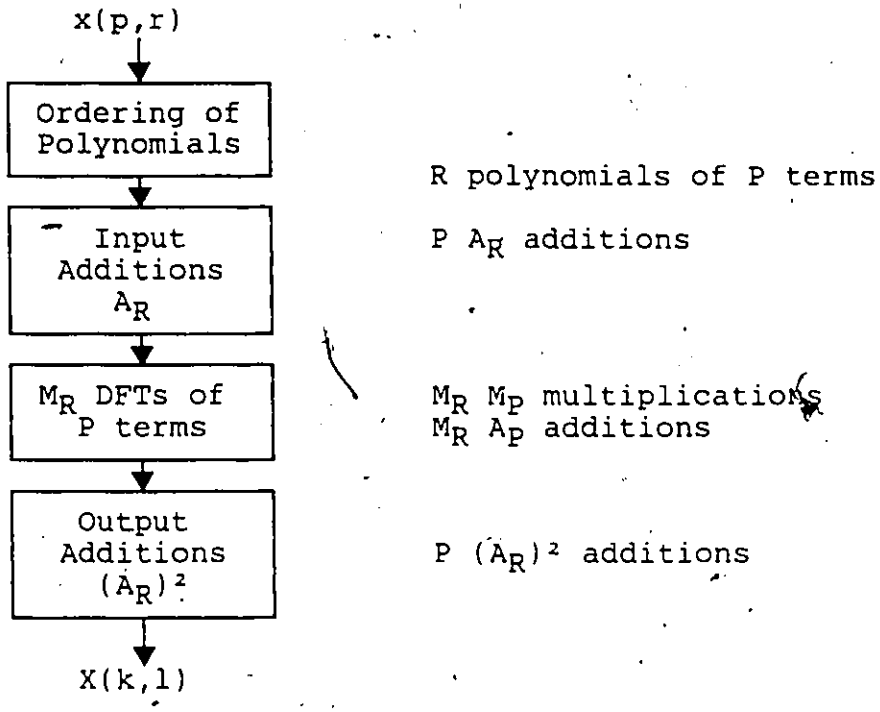


Figure 2.10 Block diagram of the number of operations for the two-factor WFTA [18].

If M_R, A_R and $(A_R)^2$ are the number of complex multiplications and input, output additions for the DFT of length R, the total number of multiplications M and additions A, becomes

$$M = M_P\, M_R \tag{2.60}$$

$$A = P\, A_R + M_R\, A_P \tag{2.61}$$

where M_P and A_P are, respectively, the number of multiplications and additions for the P-point DFT. Note that the number of additions depends upon the order in which the operations are executed. Table 2.5 gives the number of nontrivial real operations for the one-dimensional DFTs computed by the WFTA. Those results apply also to the number

of multipliers and adders in a parallel system.

Table 2.5 Number of nontrivial real operations for DFTs computed by WFTA and Rader's algorithm [18].

DFT size N	Number of real multiplications	Number of real additions
30	68	384
48	92	636
60	136	888
120	276	2076
168	420	3492
240	632	5016
420	1288	11352
504	1572	14540
840	2580	24804
1008	3548	34668

2.4.5) Discussion:

In this section, the common factor, the prime factor and the Winograd Fourier transform algorithms were briefly reviewed. Reference [18] upon which most of the section is inspired on gives a detailed derivation of these algorithms.

It is impossible to say which of these algorithms is optimum for VLSI hardware implementations. Chapter 3 will introduce a number of VLSI design guidelines which will help in making a choice.

The main characteristics of the three FFT algorithms can nevertheless be highlighted. The common factor algorithm applied to a sequence of length N, where N is a power of two is best characterized by the fact that a regular butterfly calculation is required throughout the computation of the DFT. The prime factor algorithm which calculates a DFT as a series of small DFTs reduces the number of operations when compared to the CFA but requires different types of butterflies to accommodate the partition of a DFT of size N into relatively prime smaller DFTs. The principal contribution of the Winograd Fourier transform is a reduction in the number of multiplications over the CFA and the PFA.

The algorithms above have one thing in common that is the input data samples and the intermediate results require a special routing or shuffling from one node to another.

2.5) Sliding Fourier transform:

In the mid-seventies with the progress made on charge coupled devices (CCD's), an attractive technique of computing the DFT gained in importance. This technique, called the discrete sliding Fourier transform (DSFT), calculates the Fourier coefficients through a window sliding over the data sequence. Its major application is in short-time spectral analysis of speech signals [34]. A similar algorithm can be traced back to 1966 when J.H.Halberstein published [35] a method that updates the Fourier coefficients through a window like in the DSFT. This method of calculation is devised to be implemented by a computer program and has been used [36] to compute a continuous DFT. Here we discuss only the algorithm reported in [34].

The DSFT is defined as

$$X_S(k) = \sum_{n=k}^{k+N-1} x(n) \cdot h(k-n)\, W_N^{-kn} \qquad (2.62)$$

where the index "S" denotes the word "sliding". Figure 2.11 depicts a comparison of the DFT and the DSFT for N=3.

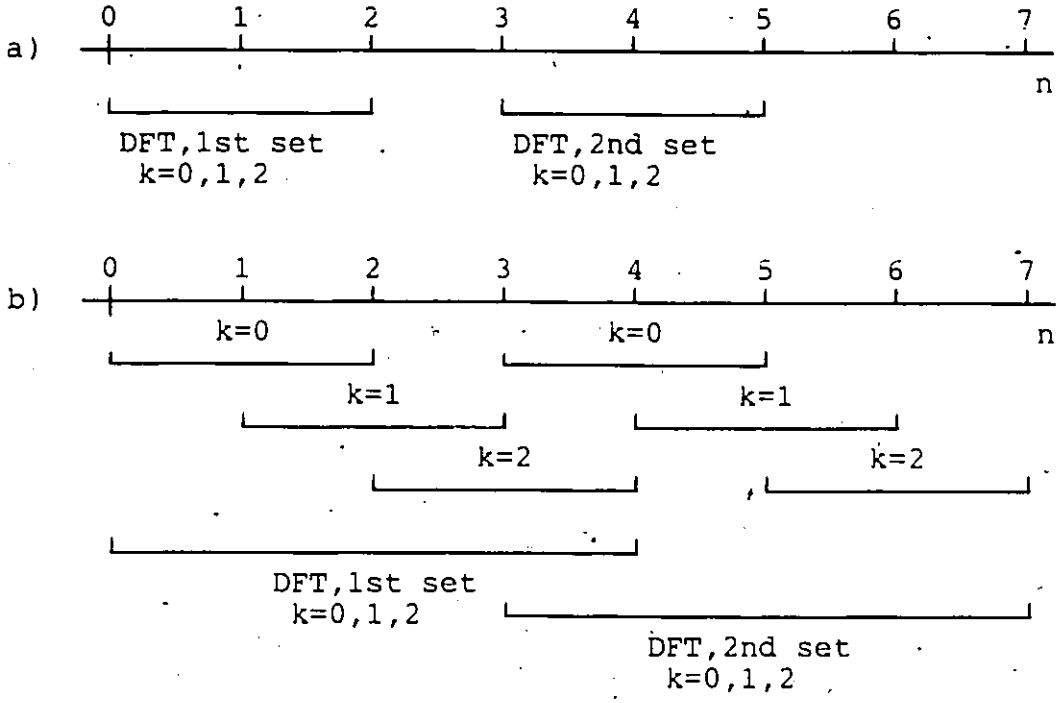


Figure 2.11 Comparison between (a) a DFT and (b) a DSFT for N=3 [34].

One can consider each nonoverlapping frame of N points of $X_S(k)$ to be an approximation to the true DFT of each respective nonoverlapping frame of N points of x(n). Another viewpoint is to consider each value of the DSFT itself as a spectral estimate. That is, since the goal is to estimate an underlying spectrum, one can argue that each of the N spectral values may as well be based on different short-time segments, as on a single short-time segment. The validity of this claim strongly depends on the stationarity of the input samples. For more information concerning this topic, the reader can consult reference [34].

In general, the DSFT doesn't yield results which are

exactly equal to the DFT unless the signal is periodic. For example in speech processing, averaging periodigrams obtained from the DSFT can results in power spectral estimates which approach those obtained with the DFT.

Concerning the implementation, the number of operations depend strongly on how the algorithm is implemented. Chapter 4 will clarify the situation.

2.6) Computation of DFTs via a delta rotation algorithm (DRA):

This algorithm evaluates the DFT of samples of a continuous time signal by multiplying the DFT of the difference signal at the output of a linear delta modulator (LDM) by a rotation factor [37]. It is in fact the combination of an encoding technique and an FFT algorithm. The details of this method of computing the DFT won't be given here. We limit our study to the characteristics of the method.

The evaluation of an N-point DFT using the DRA requires N complex multiplications and N^2 complex additions. However due to the fact that the output of the LDM has a magnitude either plus one or minus one, there is no need for full complex multiplications. The algorithm does not need a scrambling (shuffling) of the input data.

The major drawback to this DRA algorithm, stems from the fact that DM encoding a signal at a given signal-to-

noise ratio necessitates a sampling rate some M-times that of an equivalent pulse-code-modulation encoder. For this reason, we won't try to estimate the number of operations or components because the M fold increase in sampling depends on too many factors. Comparison with other DFT algorithms would be almost impossible.

2.7) Summary:

This chapter has reviewed in some details the discrete Fourier transform and its properties. Two different approaches have been covered. The first one groups all the algorithms that compute the DFT as a linear filtering process. The second category includes all the algorithms which are called fast Fourier transform algorithms. They differ considerably in concept.

The direct computation of the DFT, Goertzel and the chirp-z transform algorithms are characterized by the regularity in the data routing. No special rearrangement of the samples is necessary. Their implementation on a general purpose computer requires a number of basic operations $O(N^2)$. With a one-dimensional array that uses a maximum of parallelism (table 2.3) the number of components is $O(N)$. Chapter 3 and 4 will give further details of the arrays.

In the same category were included the Curtis-Goertzel, Ward-Goertzel and Rader's algorithms even though

they don't share exactly the same characteristics. The common property that is shared by all is the number of operations and components. These last three algorithms need a special data rearrangement known as data shuffling. They can be seen as a transition between the first and second category which contains the FFT-like algorithms and is covered below. The closest to this second category would be Rader's algorithm.

The second category composed by the common factor, the prime factor and the Winograd Fourier transform algorithms is best described by the number of operations/components to be $O(NLog_2N)$ and by the fact that data shuffling is required. As mentioned in section 2.4, the major concern at the time was the minimization of the number of multiplications as these are the most time consuming in a general purpose computer. Data routing (shuffling) was not a major problem, as all the algorithms were aimed to a general purpose computer implementation. In a computer program, addressing a data sample is done by a simple symbol specifying the row and the column in an array. One can easily conclude that the Winograd Fourier transform algorithm is the most efficient method above all.

Two other methods for computing the DFT have been mentioned the discrete sliding Fourier transform and the delta rotation algorithm. The DSFT resulted from the

progress in charge-coupled-devices (CCD) technology. The delta rotation algorithm computes the DFT but uses a new idea that is,the samples are encoded with a delta modulator. This implies a sampling frequency much higher then that of a PCM modulator for the same signal-to-noise ratio.

The chapter that follows, chapter 3, is concerned with the ways to map a signal processing algorithm into a hardware system using VLSI technology. The results of chapters 2 and 3 will be used in chapter 4 to design efficient DFT processors.

CHAPTER III

VLSI ARCHITECTURES

3.1) Introduction:

A wide variety of DFT processors have been proposed and implemented to meet the increasing performance requirements of signal processing applications in digital communications and radar. This chapter examines various existing DFT processors along with potential new VLSI architectures.

Section 3.2 discusses the performance achieved and the limitations brought by different MSI/LSI implementations of DFT processors that have been built. The recently available high performance special purpose chips are also discussed. Section 3.3 introduces the subject of VLSI technology. Designs guidelines required for the implementation of a signal processor in VLSI are exposed. Section 3.4 presents a description of multiprocessor systems. The classification discusses architectures such as distributed signal processor systems and pipeline processors. Particularly in section 3.5, the accent is put on a special type of pipeline processor that is the systolic array approach. This approach has been made possible due to rapid progress in VLSI design.

3.2) MSI/LSI implementations of DFT processors:

3.2.1) Generalities:

In the past few years, two approaches have been used to implement signal processing algorithms. One is to built dedicated DFT processors with standard off-the-shelve MSI and LSI component chips. A second is to design a system using programmable signal processor chips. This latter class includes single chips and multiple chips for bit-slice processors. Their programmable nature make them more flexible than the first approach. But on the other hand, for high speed signal processing applications e.g. radar, the first approach is preferred over the second due to speed requirements. Both methods are now discussed in a DFT context.

3.2.2) Dedicated DFT processors:

Dedicated DFT processors have been mainly used in applications where throughput rates in the range of 1-100 MHz are essential. Most of these special purpose hardware designs exploit the parallelism inherent in the FFT algorithm (figures 2.8-9) to achieve high throughputs. The realization of an eight point pipeline FFT was reported [41]. The throughput rate per sample of the processor was 128 kHz. RCA developed a pipeline FFT matched filter to process wideband signals in excess of 10 MHz [43]. A DFT

processor based on a prime radix algorithm (section 2.4.3) was designed [29]. The processor used the WFTA (section 2.4.4) to implement the low-order DFT modules. Serial arithmetic, parallel data streams and pipelined computations were used in these WFTA modules to give high throughput rate per sample in the order of 25 MHz. A radix-4 pipeline FFT processor was implemented [44]. A throughput rate in excess of 40 MHz was achieved for sequence lengths of 16, 64, 256, 1024. Recently, a fully parallel low-order FFT processor have been designed [33]. The system uses stored-product ROMs in place of hardware multipliers to achieve an equivalent throughput rate per sample of about 160 MHz for N=16.

3.2.3) High performance special processor chips:

Special processor chip realizations of the DFT usually require less hardware than dedicated DFT processors but can hardly meet the speed requirements of many radar signal processing applications. They are best used in applications like speech and sonar signal processing where throughput rates per sample ranging in the tens of kHz are sufficient.

A Fourier transform processor [59] based on the Intel 3000 bit-slice microprocessor, together with a TRW multiplier and some TTL circuits performed a 256-point FFT in 5.4 msec yielding a throughput rate per sample equal to

47.5 kHz. A TMS 320 signal processor based FFT system [61] performed a 1024-point FFT in 78 msec at an effective throughput rate of 12.8 kHz per sample. A 32-point radix-2 FFT monolithic chip was fabricated by TRW [62]. The time required to compute a 32-point FFT was 47 μsec for a throughput rate per sample of 680 kHz.

3.2.4) Discussion:

From the examples given above, one concludes that for high throughput rates per sample, in the order of several MHz, DFT processors have been constructed mainly around an FFT-type algorithm and dedicated hardware. For low rates, under 1 MHz, signal processor chips have been preferred over dedicated hardware.

Today, with new processor requirements including increased throughput rates as well as reduced power consumption and system size [64], VLSI technology will have a major significance in system designs. New architectures to implement DFT processors using VLSI have to be considered. Direct translation of existing DFT processors into VLSI becomes impossible because of the lack of compatibility of present designs with VLSI guidelines. The following section (3.3) introduces these VLSI guidelines.

3.3) VLSI technology:

3.3.1) Generalities:

It is not clearly defined where large scale integration (LSI) ceases and VLSI begins. It can be stated that VLSI chips contain more that 100 000 devices [38]. This large chip complexity presents increasing demands on design, fabrication and testing.

The chief aim of this section is to state the design rules implied by VLSI technology. Fabrication and testing are not explicitly reviewed but a particular choice of architecture for computing the discrete Fourier transform (chapter 4) will make these operations simpler. References [10-11,38-40,45-46] contain the design guidelines for VLSI architectures which are summarized below using [39] as the major reference.

3.3.2) VLSI architectural design guidelines:

"VLSI architectures should fully exploit the potential of VLSI technology and take into account (i) the layout constraints and the resultant interconnection costs in terms of area and propagation time, and (ii) the cost of VLSI processors as measured by silicon area and pin count.",[39]. Therefore,when choosing an architecture, one must stress the importance of modularity, regularity of data flow and of

control paths, local communication, and massive parallelism. The design guidelines are given below.

1) Homogeneity:

In VLSI, the designer must attempt to keep the architecture as regular and modular as possible. Identical processing cells will reduce the overall complexity. Memory and processing power will be relatively cheap as a result of high regularity and modularity. In the communication between cells, a careful algorithm study may help create some form of regularity in the layout of the system.

2) Pipelining:

In many digital signal processing (DSP) applications, throughput rate per sample represents the overriding factor dictating system performance. Pipeline techniques fit naturally in the prospect of improving throughput rate. Suitable pipelining techniques are well established, particularly for most DSP algorithms.

Promising pipelined architectures like systolic arrays are discussed in section 3.5.

3) Communication:

In VLSI technology, using multiple cells, computations per se are becoming easily affordable.

The fundamental limitation is found in the cost of communication, relative to the cost of cells, logic and storage. Communication is expensive in chip area; indeed, most of the area of a chip is taken up by wires on several levels. Communication is also expensive in sending signals externally between chips. This applies when (i) package pin limitations, the area used for bonding pads and pad drivers, and (ii) the cost of packages, must be considered in multi-chip system designs. For VLSI CMOS technology, communication accounts also for most of the power consumed and dissipated on a chip.

Thus, both the cost and performance factors of VLSI favor architectures in which communication is localized. Those architecture will play a dominating role in VLSI technology. The following section will try to identify VLSI architectures for DFT realizations.

3.4) Special multiprocessor systems:

3.4.1) Generalities:

This section surveys different classes of architectures that have been proposed for signal processing applications. All of these use the concept of parallel processing. Parallel processing systems are based upon an interconnection of a variety of processors that achieve a higher throughput rate than possible with a single processor based system [64]. Two classes of architectures are presented, that is distributed processor systems and pipeline processors. The following will help in identifying a proper VLSI architecture for a DFT processor.

3.4.2) Distributed signal processing systems:

The continuing increase in system integration and decline of hardware costs have made possible the construction of large distributed computing networks in the context of general data processing. These systems are based on a decentralization of hardware, controls, and databases. Characteristics of distributed processing systems include fast response, high availability, resource sharing and good expandability. These networks are being used widely by banking institutions and airline companies [63].

Reference [64] expand the concept directly to signal

processing systems. A collection of signal processors can be interconnected to implement a large computing task in a coordinated fashion. The network topology determines the particular distribution of computational load among the processors. Networks topologies including the star, ring, shuffle exchange, fully connected, cube, crossbar and bus networks can be used as data connection structures for communications between the various processors.

3.4.3) Pipeline processors:

Computations using a pipeline processor are done by splitting the algorithm into smaller parts and allocating separate hardware sections to each part [65]. Instructions and data flow through the stages of the processors at a rate that is independent of the length of the pipeline (number of separate stages) and dependent only on the rate at which new entries can be fed to the input of the pipeline and on the clock rate of the digital hardware. The concept behind a pipeline processor is that if an algorithm can be executed on a single processor based system in T seconds, and is partitioned into N parts or steps, then a pipeline designed to execute the same algorithm can yield rates up to T/N seconds i.e. an N-fold increase in performance. Figure 3.1 depicts the process.

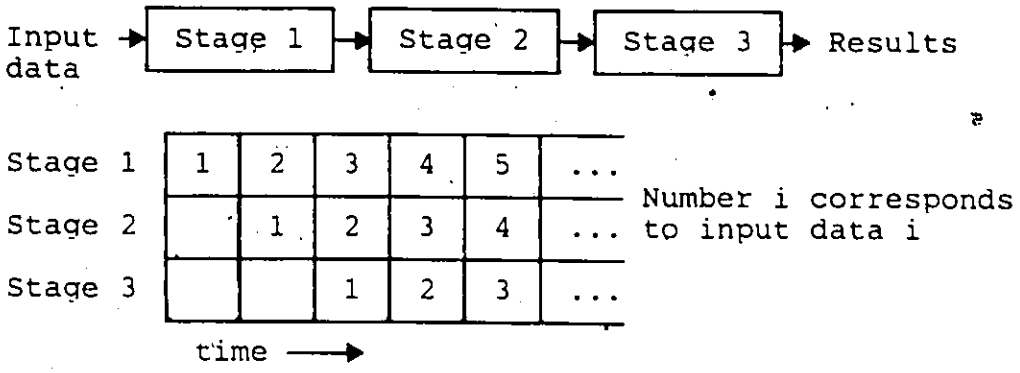


Figure 3.1 Pipeline processor for an algorithm partitioned into 3 steps.

3.4.4) Discussion:

In most multiprocessing systems, to achieve high throughput rates, it is essential to spread a task (algorithm) over many processing elements. However, the general-purpose nature of multiprocessor systems i.e. which can be reconfigured for different computing algorithm has led to a complicated system organization and severe system overheads. Such systems are not suited for real-time signal processing where a high throughput rate is absolutely essential [39]. Moreover, if a VLSI implementation is foreseen, a successful system will be determined by the proper use of the design guidelines mentioned in section 3.3.

A solution to real-time signal processing requirements with minimum system overheads is the use of special-purpose pipeline processors known as systolic arrays. Systolic arrays fall naturally in the class of special-purpose VLSI architectures. They have structural properties that are suitable for VLSI implementation. The key attributes of systolic arrays are described in the following section.

3.5) Systolic array architecture:

3.5.1) Generalities:

The systolic array architecture, introduced by H.T.Kung and C.E.Leiserson [38], is a powerful approach for applying VLSI to signal processing. This section is concerned in bringing out the main characteristics of the systolic array architecture. General diagrams showing cell distributions and communication paths are depicted. Cell complexity could range from a simple combinatorial logic [40] to a multiplier-accumulator [46]. Thus a cell is seen as a black box where data streams are delayed and transformed. Communication paths represented here by straight lines accommodate multi-bit data travelling either in parallel or in serial form.

Computational tasks can be classified into two families, compute-bound algorithms and I/O-bound algorithms. When computing a particular algorithm, if the total number of operations is larger than the total number of input and output elements, then the algorithm is compute-bound, otherwise it is I/O-bound [45]. For example, the ordinary matrix-matrix multiplication represents a compute-bound algorithm. Adding two matrices, on the other hand is an I/O-bound algorithm.

Any attempt to speed up the calculation of an I/O-

bound algorithm must rely on an increase in memory bandwidth (access time). Memory bandwidth can be increased by using faster components [63]. Speeding up a compute-bound algorithm, however, may often be accomplished in a relatively simple and inexpensive manner, that is, by the systolic approach [45].

A systolic system is a network of processors which rhythmically computes and passes data through the system. It is mainly composed of few different kinds of processing elements (PE), ideally only one kind. This is best illustrated by figure 3.2. The figure shows an example of a one-dimensional systolic array. The concept can be extended to a 2-D network of cells, figure 3.3. For the latter example, the interaction between the main memory and the network are much more demanding than for the one-dimensional systolic array.

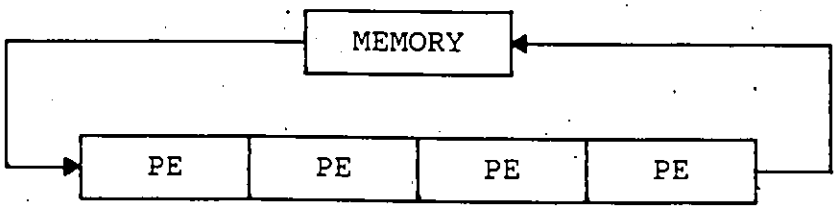


Figure 3.2 Basic principle of a one-dimensional systolic array [45].

The array above achieves a four-fold improvement in speed over a single PE system. The improvement depends strongly in a successful mapping of an algorithm into a

recurrence equation i.e. separation into parts.

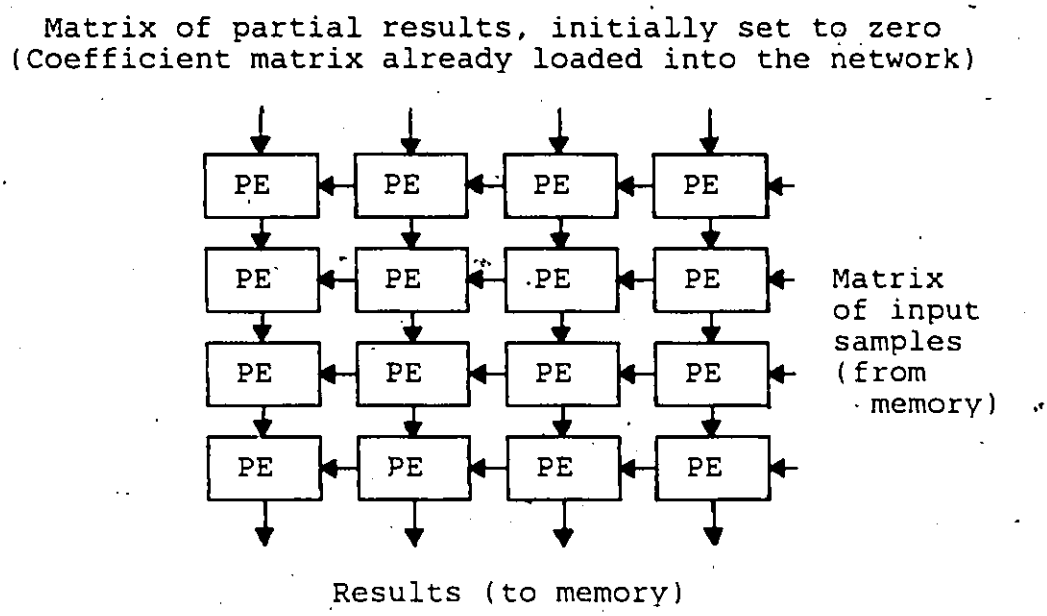


Figure 3.3 Two-dimensional systolic array [46].

The simplest configuration is certainly the one-dimensional array. We shall now focus our interests on that architecture, which may realize many DSP algorithms. With some effort, it is possible to extend the study to the 2-D case. Here again, it is mentioned that the networks given below are not aimed at any special application. Only general one-dimensional systolic arrays are discussed. Possible applications will be enumerated in section 3.6.

3.5.2) One-dimensional semi-systolic array:

This arrangement of cells is characterized by the fact that input data or output results require to be broadcasted. Figure 3.4 shows two examples of such systems. The first one ,(a), depicts a structure where a data input is applied (broadcasted) to all cells of the systolic array during each cycle via a bus. The results flow one after the other in time. The second figure (b), illustrates a structure where the input data are fed one after the other through the systolic array. The results which may appear skewed in time or all at once are retrieved by a bus or a tree-like network [38,45].

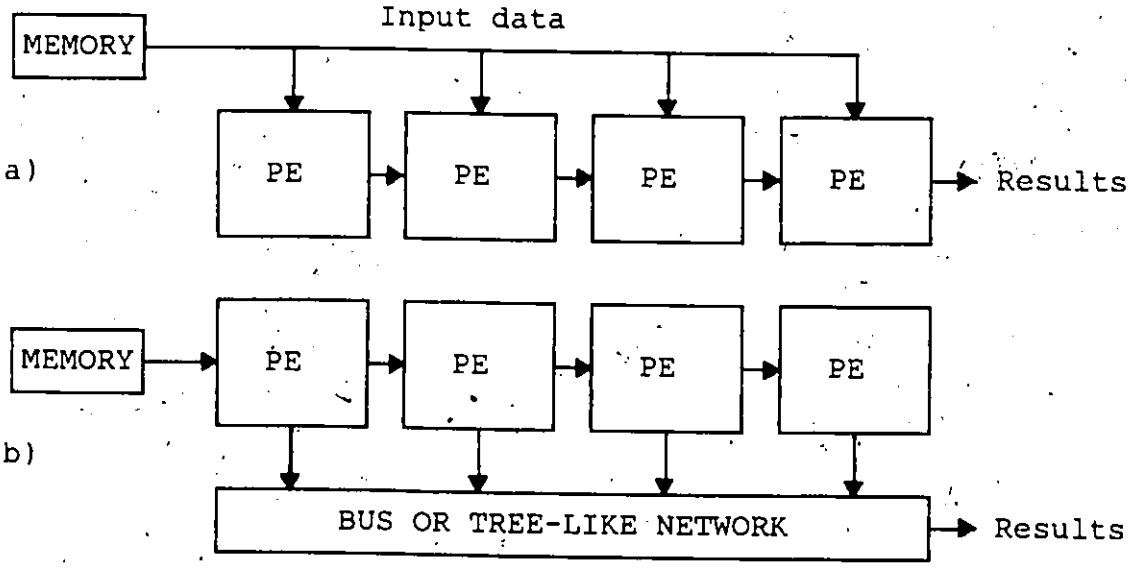


Figure 3.4 Semi-systolic arrays,a) broadcasted inputs, b) results retrieved by bus or a tree-like network.

3.5.3) One-dimensional pure-systolic array:

This network uses only local communications. Figure 3.5 gives two architectural arrangements. Part a) shows an architecture where the input data move through the array of cells in a pipeline fashion and the results reach the output one after the other through a sequence of registers. On the other hand, the network of part b) has true pipeline data input and output.

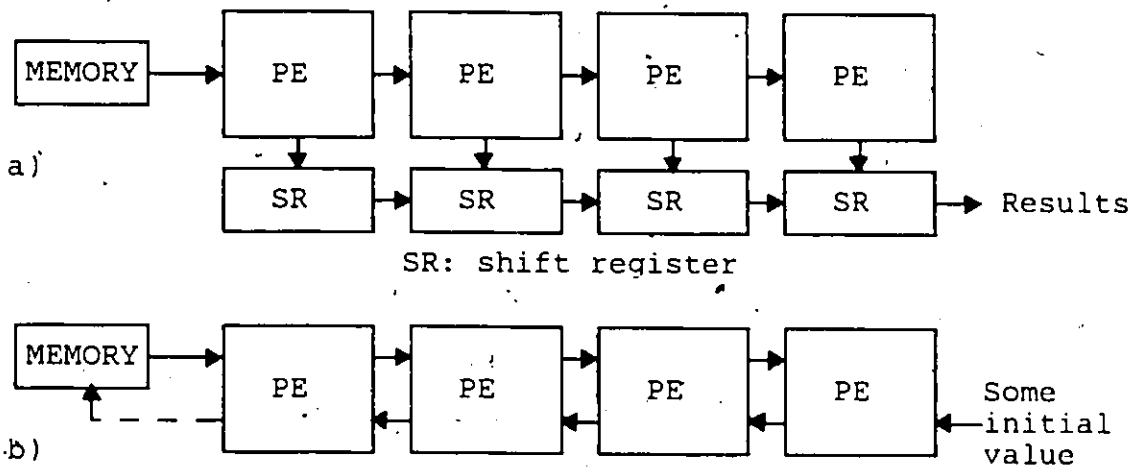


Figure 3.5 Pure-systolic arrays, a) "systolized" output path, b) fully systolic array; with four PEs.

Variants of these architectures exist, but the examples shown describe semi and pure-systolic arrays best.

3.6) Systolic arrays applied to digital signal processing:

Digital signal algorithms that can be mapped onto one-dimensional and two-dimensional systolic array architectures are summarized below. [45-46,49-51].

COMMUNICATION GEOMETRY	EXAMPLE
One-dimensional systolic arrays	1-D median filtering, " FIR filters, " IIR filters, " convolution, " correlation, discrete Fourier transform, general matrix-vector products.
Two-dimensional systolic arrays	2-D filtering as above, discrete Fourier transform, adaptive noise cancellation, general matrix-matrix operations.

Some of these DSP algorithms fall naturally in one of the two communication geometries, i.e. one-dimensional DSP algorithms are mapped onto one-dimensional arrays and the two-dimensional ones onto 2-D arrays. Others, like the DFT can be mapped onto both geometries according to the computational algorithm used (chapter 2).

3.7) Discussion:

The chief concern of this chapter was the identification of a suitable VLSI architecture for the design of a special purpose DFT processor.

Section 3.2 presented some existing DFT processors built with standard MSI/LSI components and programmable processor chips. The processors built for high speed signal processing with throughput rates per sample in excess of one MHz were found to lack expandability and to become too cumbersome for transform lengths greater than 16 samples. These processors are usually expanded by relatively complicated hardware techniques. Processors based on programmable chips are used mainly for low throughput rate applications e.g. speech and sonar. They are easily expanded via software techniques but with present day technology cannot reach the throughput rates required in areas such as digital communications and radar.

Subsequently, section 3.3 reviewed the major VLSI designs guidelines. As the concentration of many processing elements becomes more and more economical, the use of extensive parallelism in future VLSI designs will require the use of proper architectures. These architectures are composed of multiple cells preferably of few different types which are interconnected by a local network of control and

data paths. Thereby, with these VLSI architectural guidelines, direct translation of the DFT processors mentioned in section 3.2 into VLSI architectures becomes incompatible with such guidelines; the use of an FFT computational algorithm being the cause. As was explained in chapter two, FFT-like algorithms require a complex data exchanging scheme like the shuffle exchange network. In this case, partial results are sent to and received from distant processing elements. These non-local communications must be avoided in true VLSI designs.

A simple classification of special multiprocessing systems was then presented. Two topics were discussed distributed signal processing systems and pipeline processors. The first class can be used in a general purpose signal processing environment where multiple DSP algorithms are required in several application areas. This kind of system necessitates a careful partition of an algorithm, and also an operating system and an application software package. These requirements combined may account for the relatively high price tag [50].

The second class, the pipeline processor, was introduced as a remedy to the rather demanding distributed signal processing systems. One sub-class named systolic arrays gained interest recently in signal processing applications [38-40,45-46,48-49]. The architecture makes

multiple use of each data input, therefore high throughput rates per sample can be achieved with modest I/O bandwidths for outside communications. Extensive parallelism through the use of simple basic cells rather than the sequential use of a few complex processors as in computer architectures leads to powerful special-purpose systems. Indeed, cells must be simple and of a few types so that design, implementation and testing costs can be minimized. The communications between neighboring cells are simple and local, and the control unit is very simple. All these characteristics imply simple, area-efficient layouts of systolic arrays implemented using VLSI technology. Section 3.6 gave some digital signal processing algorithms which can be mapped onto systolic arrays.

The following chapter will be concerned exclusively with the mapping of the discrete Fourier transform onto systolic array architectures.

CHAPTER IV

SYSTOLIC ARRAY IMPLEMENTATION OF THE DFT

4.1) Introduction:

One DSP algorithm that deserves special attention for systolic array implementation is the discrete Fourier transform. In chapter 2, a group of DFT computational algorithms were presented. For the past 20 years, a sub-class, the FFT algorithms, has prevailed over the others for hardware implementations (see section 3.2). But now, VLSI technology has entailed a major review of the criteria for evaluating the implementation of digital signal processing algorithms. For example, the efficiency of an algorithm is based more on the degree of the communication complexity required between the hardware elements rather than on the number of arithmetic operations needed (refer to section 3.3). Due to the complex global communication requirements, DFT implementations using an FFT type algorithm are now questioned [51,55]. Therefore, linear filtering computation of the DFT as outlined in section 2.3 which can be mapped onto VLSI systolic arrays ought to be considered.

The different architectures proposed in the current literature to meet the constraints for efficient VLSI

systolic implementations are surveyed in section 4.2. Existing two-dimensional, and one-dimensional semi and pure-systolic arrays are described in terms of hardware requirements and throughput rate. Then, in section 4.3, two improved systolic array implementations of the DFT are introduced. Based on a modification of the Goertzel algorithm, these two systolic arrays yield systems with minimal complexity and therefore increased throughput rates. Finally hardware implementations for these two new DFT systolic arrays are presented and discussed in section 4.4.

4.2) Survey of existing systolic arrays for the DFT:

4.2.1) Generalities:

Several architectures have been proposed for mapping the DFT algorithm onto systolic arrays. They can be classified into two categories: one-dimensional and two-dimensional. These architectures are first presented, and finally, their characteristics are summarized and analysed.

A frame is used to refer to a set of N data samples or simply a vector. Moreover, the time to output a frame is the time taken by the network to compute a DFT from the first to the last DFT sample to leave the array.

4.2.2) Two-dimensional systolic array for matrix-matrix multiplication:

The DFT of a finite data segment (frame) can be expressed as a matrix times vector product (2.22). To transform a sequence of frames, this operation can be represented as a matrix times matrix multiplication. Urquhart and Wood [49] described a two-dimensional array for such a matrix operation. Figure 4.1 shows the array for the DFT of two sequences, $x_1(n)$ and $x_2(n)$ of length N=3. This two-dimensional array was seen in section 3.5.1, figure 3.3.

The parallelogram shapes represent the order of arrival of (i) the input data (right hand side) (ii) the initial conditions (on top). The parallelogram at the bottom of the figure shows instead the order of appearance of the DFT samples, X(0) being the first one. The N^2 twiddle factors are static in the array, one in each cell. The $x_i(n)$ are fed as rows of a matrix. The resulting frequency samples $X_i(k)$ are computed in the array and exit in the order shown.

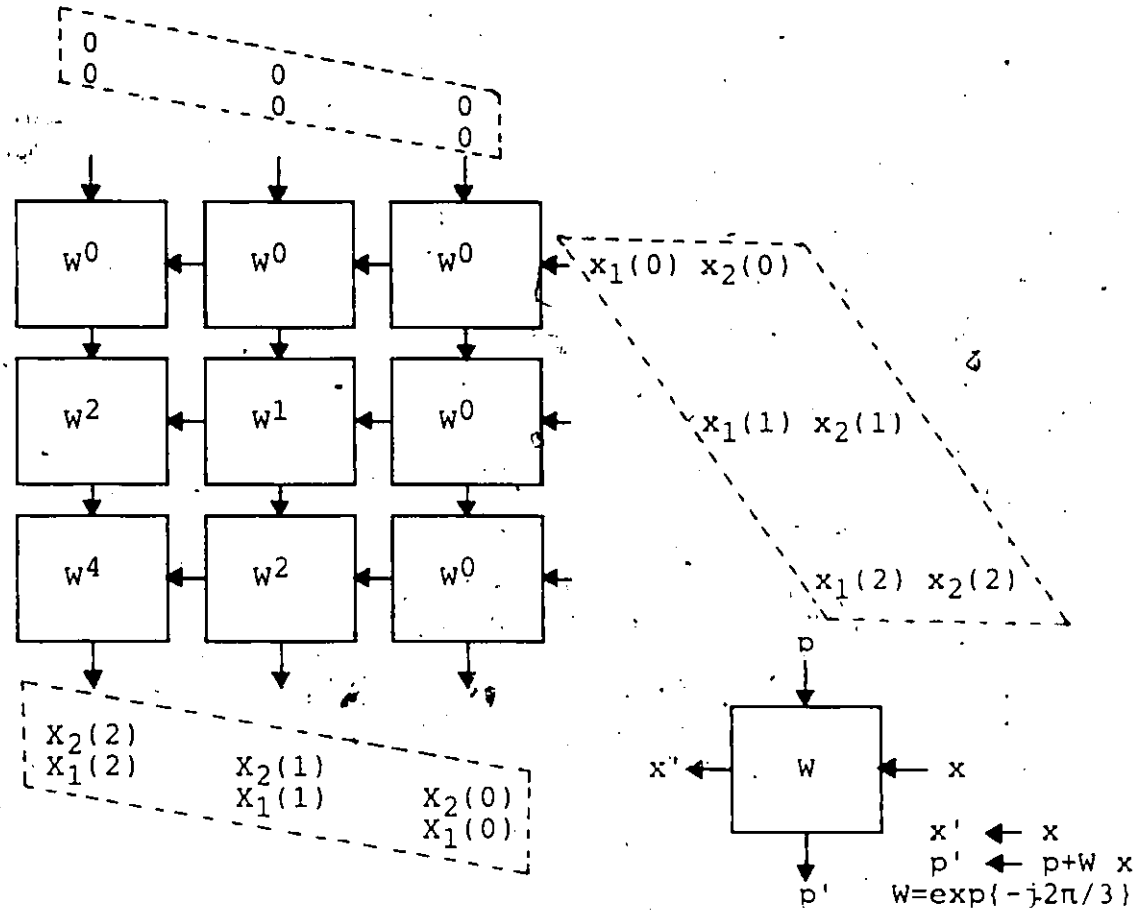


Figure 4.1 Two-dimensional systolic array for the computation of the DFT, for N=3 and 2 frames x_1 and x_2, [49].

Such an architecture requires a total of N^2 basic cells. It also needs an extensive communication network for exchanging the input samples and the output results typically N input samples are fed in and N DFT samples leave at each clock cycle. Therefore, the problem of packaging multicell array chips must be considered. Other two-dimensional systolic arrays can be applied to the DFT. Due to the low utilization efficiency of the processing elements, which is 25% to 50%, they were omitted in this thesis [38,49].

4.2.3) One-dimensional systolic array for matrix-vector multiplications:

A one-dimensional systolic array that implements the DFT algorithm as a matrix-vector multiplication is shown in figure 4.2 for N=4. The computation of the DFT starts by first storing the 4 elements of x(n); one in each cell. Then the matrix composed of the twiddle factors, W_N^{nk} is passed across the array along with a vector of initial values set to zero (on top). The results exit at the bottom of the array ,X(0) first and finally X(3).

This architecture represents an improvement in terms of hardware complexity; only N basic cells are needed and only one parallel data path is required (for the twiddle factors). But due to the initial pre-loading requirement of the data x(n), this one-dimensional systolic array is obviously not suited for processing a continuous stream of frames. A complete vector (frame) is computed in 3N clock cycles. This result includes the initial delay of N clock cycles.

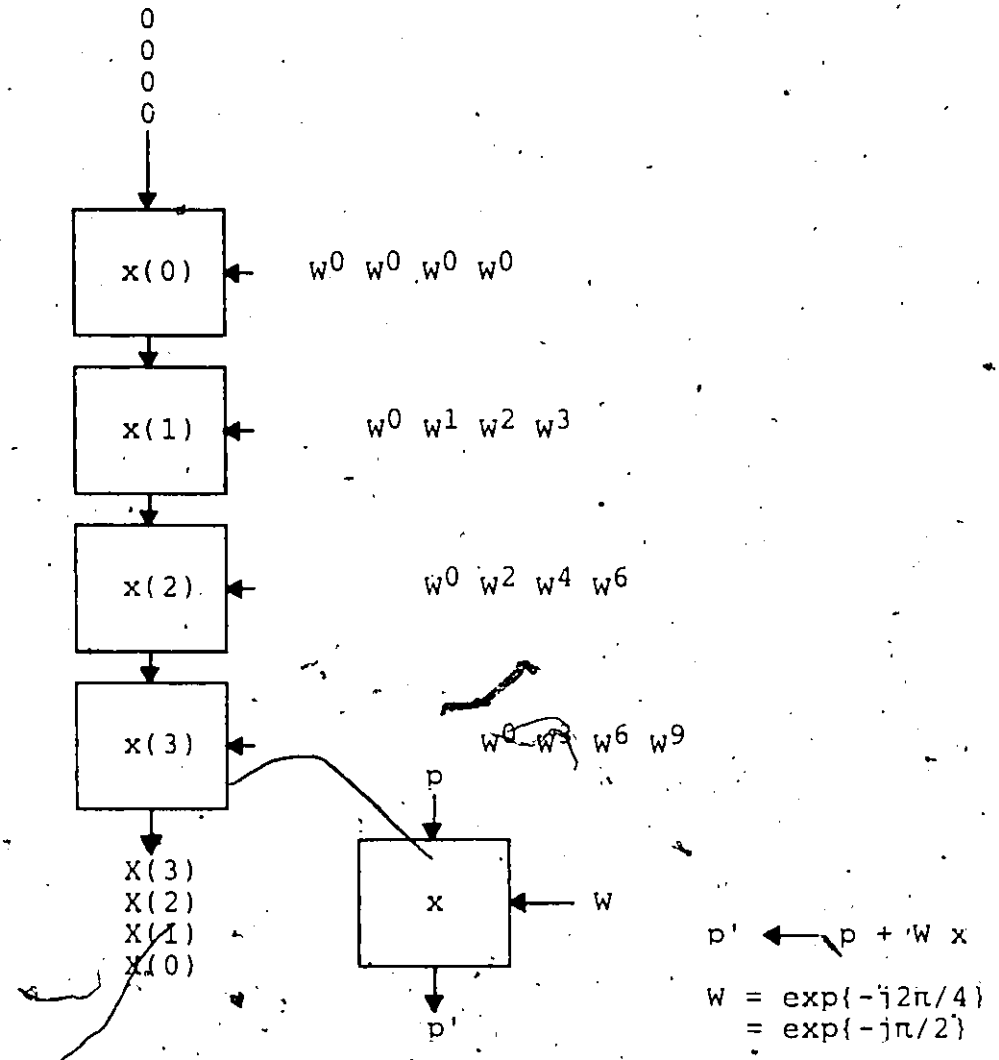


Figure 4.2 Matrix-vector one-dimensional systolic array applied to the DFT, N=4, [49].

4.2.4) One-dimensional systolic array using pipelined data inputs:

The basic equation that defines the DFT (see 2.1) can be mapped directly onto a systolic array. Each one of these cells receives one data sample every clock cycle and after N clock cycles outputs one DFT sample. They each contain N pre-loaded twiddle factors i.e. one row of the twiddle factor matrix. Partial results are retained in the cell but data samples are fed through the array in a pipelined fashion. Figure 4.3 illustrates this process.

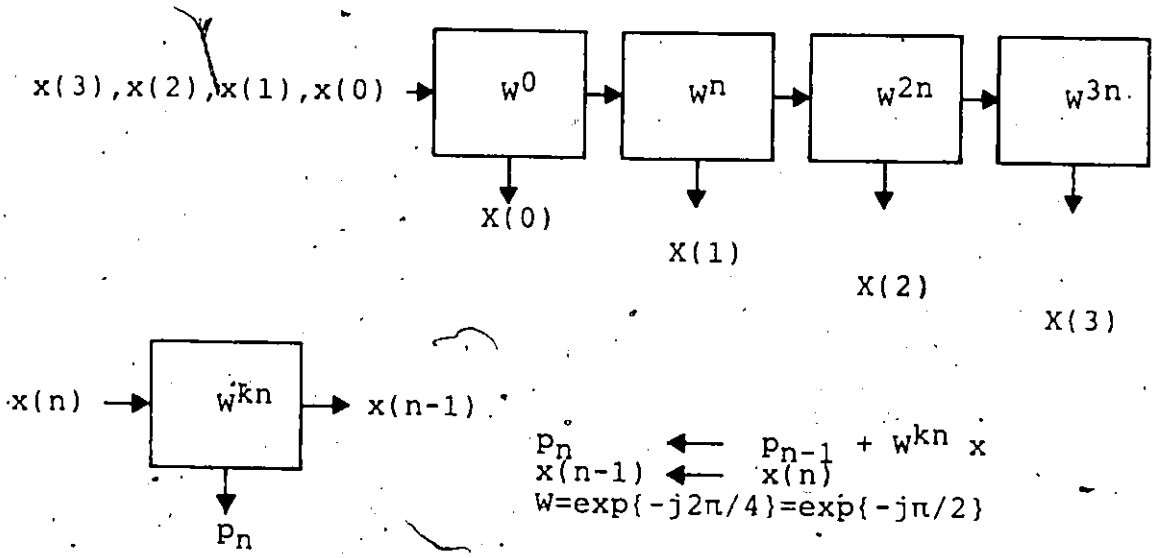


Figure 4.3 One-dimensional systolic array using pipelined data inputs, N=4, [39].

Given that the results appear skewed in time, they can be retrieved by either a data bus or a tree network (see figure 3.4 b)). This systolic architecture allows the processing of a continuous flow of frames. After N data

samples passed through a cell, the partial result at that particular cell is reinitialized to zero therefore making the cell ready for the computation of the spectral line corresponding to the next frame. The throughput rate is proportional to the time delay introduced by a basic cell. A complete DFT is performed in N clock cycles.

4.2.5) Kung-Leiserson's one-dimensional systolic array for the DFT:

One of the first systolic array for the computation of the DFT was described by Kung and Leiserson [52] ,and appears in the book by Mead and Conway [38]. The architecture is illustrated in figure 4.4 for a 6-point DFT.

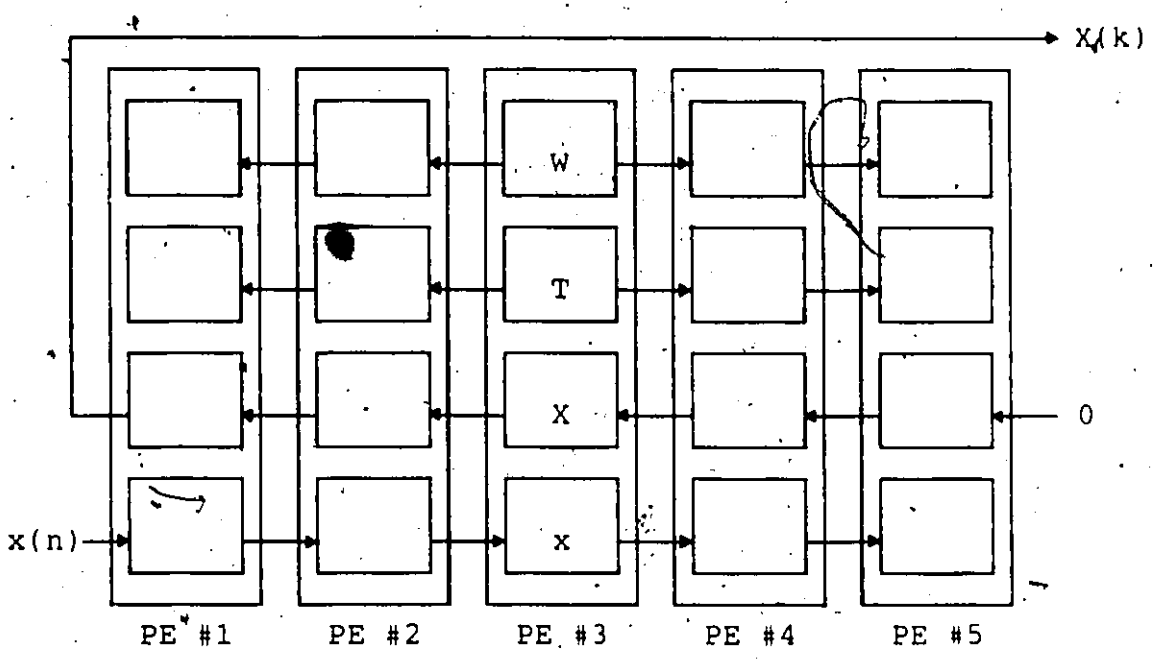


Figure 4.4 Kung-Leiserson's one-dimensional systolic array for the DFT, N=6, [53].

Each PE contains four registers (W,T,X,x) and performs operations according to the following rules [53]:

(a) For PEs #1-2 (respectively,#4-5),registers W and T receive the contents of the corresponding registers on the right (respectively,on the left) on each pulsation (clock cycle),

(b) For PEs #1-2 and #4-5, register X shifts its content to the left on each pulsation and register x ,to the right,

(c) All odd-numbered, except #3, (respectively, even-numbered) PEs perform the following operations on odd-numbered (respectively, even-numbered) pulsation:

$$X \leftarrow X + W\,x$$

$$W \leftarrow W + T$$

(d) PE #3 (at the center) performs the following operations on even-numbered pulsations:

$$W \leftarrow W + T^2\,w$$

$$T \leftarrow T\,w$$

where $w=e^{-j2\pi/N}$

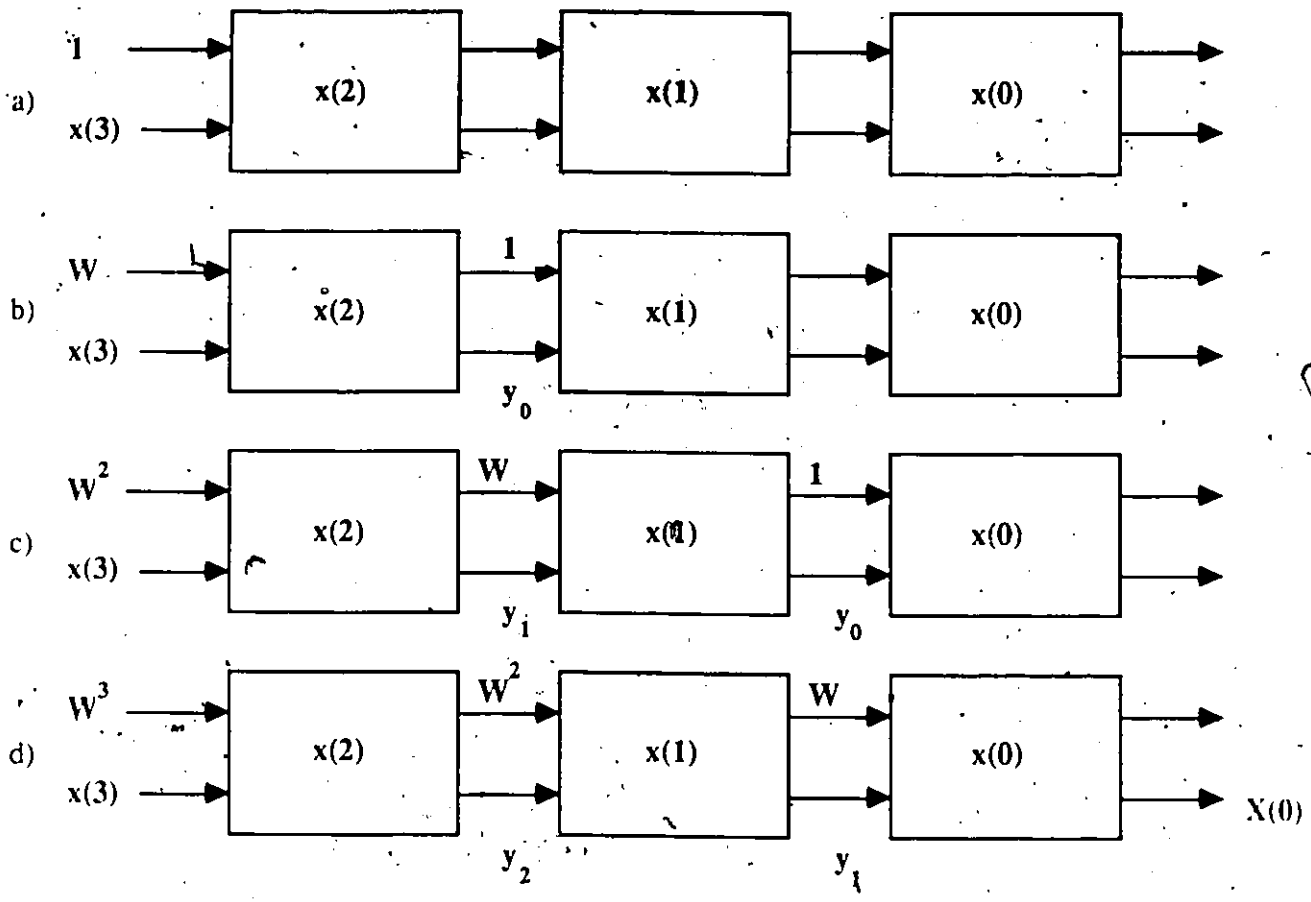
PEs #2 is initialized with the first data sample of x(n).

For an N-point DFT, the linear array described in figure 4.4 will take N clock cycles to perform the

computation of the first point of the required DFT vector. Subsequently, because of the array's serial pipelining, it will take another N-1 clock cycles for the remaining N-1 results to be piped to the output, therefore completing the full DFT vector in O(2N) pulsations. These values are referenced from the end of the pre-loading phase i.e. after PE #2 has been loaded with x(0).

4.2.6) Kung's one-dimensional pure-systolic array for the DFT:

Kung in [51] proposes a pure-systolic array that uses Horner's rule for the computation of the DFT (2.29). Figure 4.5 illustrates the scheme for N=4. The data sequence is first loaded in the array,(a), then the calculation of the DFT starts by moving the first coefficient $W_4^0=1$ into the array, (b), and by shifting the following 3 coefficients, (c)-(d). The first frequency sample X(0) appears at the output of the right most cell when the fourth twiddle factor is loaded,(d). Then, the other 3 frequency samples follow in order at each clock cycle.



Basic cell:

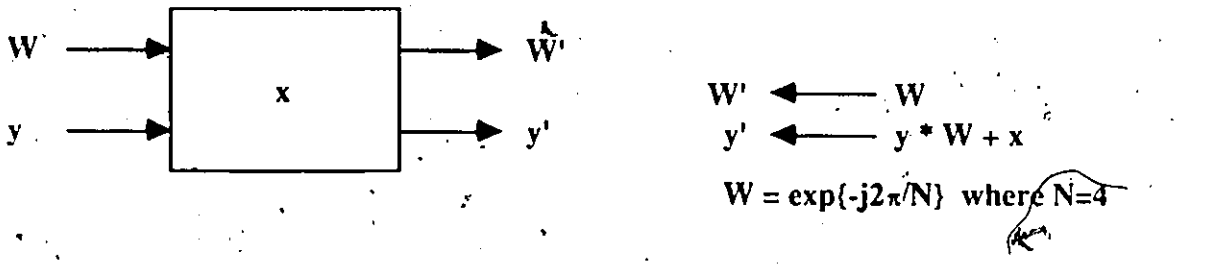


Figure 4.5 Kung's one-dimensional systolic array at successive clock cycles , N=4 [51].

For each N-point DFT, it takes a total of 2N-2 clock cycles to get the first DFT sample. For a full length DFT, a total of 3N-3 clock cycles are required from the first sample entry x(0) to the last DFT coefficient X(N-1).

This arrangement can be best used when a single DFT vector is needed (this case is seldom to occur [14]) or when a series of DFTs are to be computed but not in a continuous fashion. To accommodate a continuous flow of data sequences, this one-dimensional systolic array would require buffers for data management. This solution will in fact create a control overhead not really desirable with VLSI implementations (see section 3.3).

4.2.7) Sliding DFT systolic array implementation:

This sub-section examines a sliding DFT systolic array implementation [54]. As mentioned in section 2.5, the sliding DFT gives the same results as the DFT only when the waveforms under study are periodic. Using this DFT computational algorithm (see section 2.5), it is possible to design a one-dimensional pure-systolic array similar to Kung's which doesn't need separate loading and computational phases. Figure 4.6 depicts such an array for N=3.

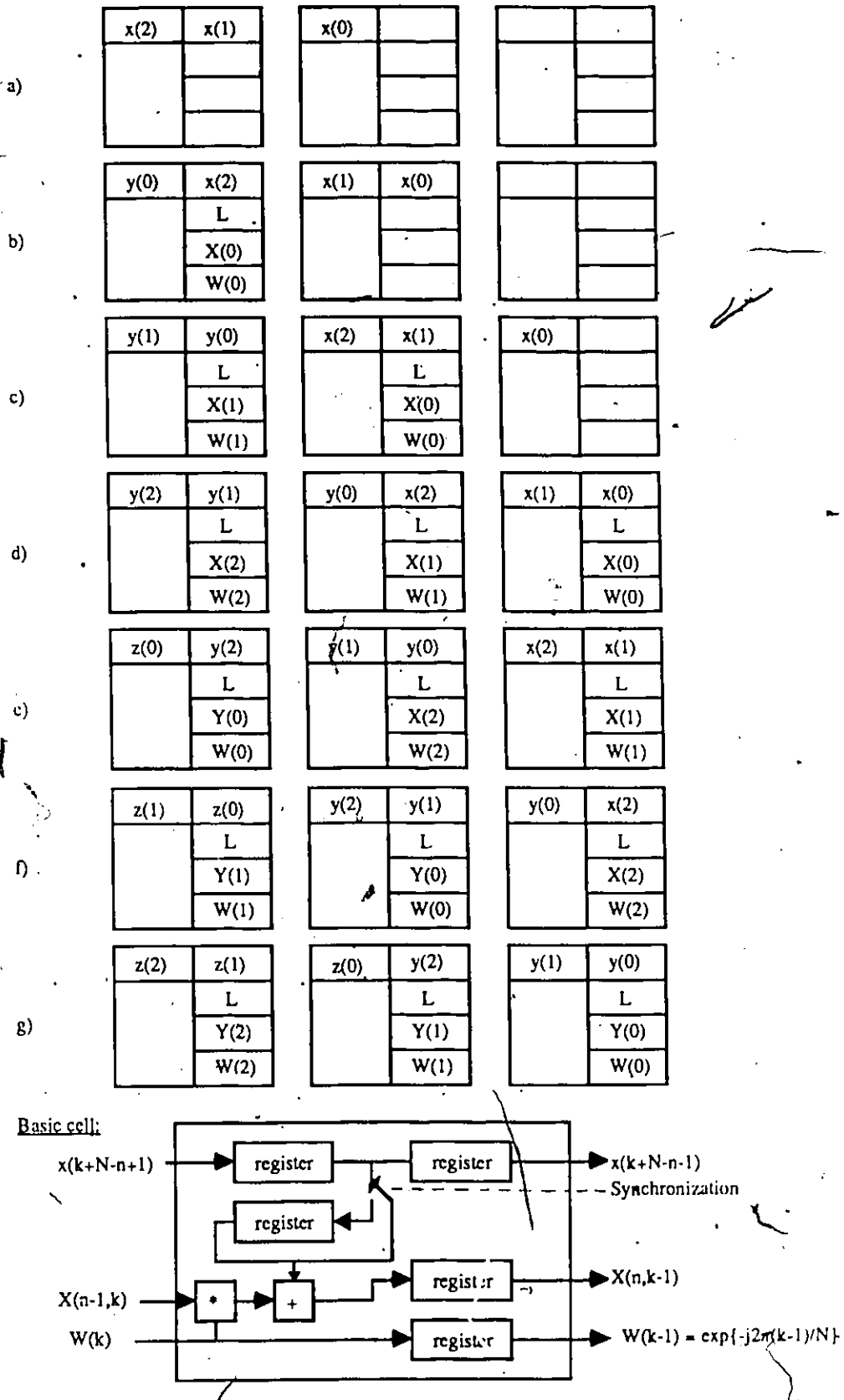


Figure 4.6 Pure-systolic array implementing the sliding DFT algorithm, N=3,[54].

Figure 4.6 is the timing diagram of the flow of data samples and results through the array. The data samples, the twiddle factors and partial results contained in the registers in each of the modules (3 basic cells) are shown at successive times (down the diagram). The instant of latching of the data samples into the appropriate register in the modules is indicated by a capital "L". Data samples are shown moving through the array while the previous frame of DFT coefficients are being computed. Following the arrival of the last sample of a frame, x(N-1), into the first module, the computation of the DFT for that frame proceeds while data samples of the next frame move through the array, y(n). The first output for a frame, X(0), is obtained N clock cycles after the arrival of the last data sample for that frame, x(N-1), and this is 2N clock cycles after the arrival of the first data sample x(0). The last output for the frame, X(N-1), is available N-1 clock cycles after X(0). The same procedure applies for y(n) and z(n) and any subsequent frame.

4.2.8) Comparison:

The characteristics and performance of the five DFT systolic arrays are given in table 4.1. These architectures are also compared to a fully implemented FFT (section 3.2.2) [55] and to a perfect shuffle [39,53] realization of the DFT using a radix-2 FFT algorithm (section 2.4.2). The parallel FFT architecture refers to the implementation of all the nodes shown on figure 2.8 or 2.9. The perfect shuffle is in fact the implementation of one column of nodes combined with a rearrangement of the data paths so that results at each iteration are transferred to the right node. The seven networks are described in the table using seven entries, mainly:

(1) Number of PEs: the exact number of PEs used in the given architecture. A PE is defined as an arithmetic unit composed of multipliers, adders and registers.

(2) Complexity of a PE: the number of multipliers and adders per cell for complex data input.

(3) Time to output the initial vector: the number of clock cycles required to calculate a frame from the first data sample,e.g. x(0),to the last DFT sample to exit the array,e.g. x(N-1), where a clock cycle is the time delay introduced by one basic cell (PE).

(4) Time to output subsequent vectors: the number of clock cycles required to calculate successive frames (sometimes referred to throughput rate).

(5) Area: this is a rough figure. Values given are adapted from [55] to [51]'s results. Here what is important is the relative area between different architectures.

(6) External memory requirements: the memory required off-array, usually to provide or collect the data samples and results.

(7) Communications complexity: type of links needed for exchanging information between cells.

Part of these results can be found in [51,53,55] others were mentioned in the previous sections. The sliding DFT systolic array implementation was excluded from this table, because it doesn't compute a real DFT (only for periodic signals). More about the algorithm is said in section 2.5. The same argument applies to possible DFT systolic arrays implementing a delta rotation algorithm, section 2.6. On the other hand, the prime-radix algorithm transform (PRAT) wasn't considered for any systolic array implementations in spite of its similarity with the Goertzel algorithm. This is due to the fact that such an implementation would violate

the VLSI guideline that asks for simple and local communications. PRAT requires data shuffling like an FFT.

For the case where a continuous flow of vectors are to be computed e.g.in real-time signal processing, the one-dimensional matrix-vector and Kung's systolic arrays must be discarded because of rather lengthy processing time. The others are very good candidates. The fastest are the two-dimensional arrays namely the two-dimensional systolic array and the parallel FFT architecture. They both yield N DFT samples every clock cycle. But despite this high throughput, they are by far the largest designs available. The two-dimensional systolic array uses N^2 basic cells and requires an extensive I/O management. Internal communications are, on the other hand, of the local type. The parallel realization of the FFT algorithm uses instead fewer basic cells: $(N/2)\cdot LogN$. But this reduction in arithmetic components is outweighed by the complex inter-cell communications. This is due to the way the FFT algorithms are formulated (section 2.4).

Table 4.1 Comparison of proposed VLSI systolic arrays for the DFT, including some FFT solving circuits.

	2-D	Matrix-v	pipeline	K-L	Kung	FFT para	FFT P-S
#PE	N^2	N	N	#1:N-2 #2: 1 (center)	N-1	N/2 LogN	N/2
PE	adders:4 mult. :4	adders:4 mult. :4	adders:4 mult. :4	#1:A=6 M=8 #2:8,12	adders:4 mult. :4	adders:6 mult. :4	adders:6 mult. :4
Lat.	2N	3N	2N	2N-1	3N-3	LogN	LogN
Time	1	3N	N	N	3N-3	1	LogN
Area	$O(N^2)$	O(N)	O(N)	O(N)	O(N)	$O(N^2)$	$O(\frac{N^2}{LogN})$
Mem.	Note 1	Note 1	Note 2	Note 2	Note 2	Note 1	Note 3
Comm	Note 4	Note 4	Note 5	Note 5	Note 5	Note 6	Note 7

$$LogN = Log_2 N$$

Notes:

1 : Memory large enough to store/retrieve data/results.
2 : Very low requirements.
3 : Able to generate or supply the twiddle factors.
4 : Local type communications between cells.
5 : Very local communications between cells and main memory.
6 : Highly complex internal communications, requires special data shuffling.
7 : Complex data routing (data shuffling).

Using current technology, one might place ten multiply-add cells on a chip. This means that one hundred thousand chips would be needed for a thousand-element DFT for the N^2-cell design compared to five thousand chips for the parallel FFT. The two-dimensional DFT system cannot be considered feasible for large N with present day technology. The parallel FFT is more attractive because it requires fewer cells that the two-dimensional systolic array. As mentioned above, the drawback of this architecture is that the wires routing on chips is very area-consuming and some are very long, typically O(N/2). The other DFT architectures use only nearest-neighbor connections.

The one-dimensional pipelined-data array and the Kung-Leiserson's systolic array also are capable of real-time signal processing. They require almost the same number of basic cells, though the second network uses more multipliers and adders per cell than the first one. For expandable systems, the Kung-Leiserson's systolic array is definitely the architecture to choose. Its local I/Os permit expansion not achievable by the one-dimensional pipelined-data systolic array. This is because loading capacitance on a bus or delay found in a multi-level tree network can significantly slow down the whole system [38]. In terms of hardware complexity, a thousand-point DFT would need for

both architectures approximately 100 chips of 10 multiply-add cells each.

The perfect shuffle network uses N/2 basic cells and can be used in real-time signal processing (is being used already [33]). However, it suffers from the same drawback found in all DFT networks realized via an FFT algorithm that is long inter-cell (or inter-chip) wires. Most of the chip area is covered with wires. It is not easy to expand the perfect shuffle network to let say N=256 starting with networks built for N=16, this is because the off-chip connections that would be required to get that larger perfect shuffle network won't necessarily be of local type [55].

In the following section, we address the problem of designing cost-effective systolic arrays for the computation of the DFT applied to the processing of a continuous flow of data vectors.

4.3) Improved DFT systolic arrays by means of the modified first order Goertzel algorithm:

4.3.1) Generalities:

This section introduces an improved DFT computational algorithm. Based on a modification of the first order Goertzel algorithm, the method is easily mapped onto efficient semi and pure-systolic arrays.

The Goertzel algorithm is first derived using Horner's rule for polynomial evaluation. The resulting recurrence equation is then modified so that the DFT of a continuous flow of data sequences can be supported i.e. not like in the one-dimensional systolic array for matrix-vector product and for Kung's systolic array where a pre-loading phase is required. Finally the proposed recurrence equation is mapped onto two different systolic arrays: a semi-systolic array and a pure-systolic array.

4.3.2) Goertzel algorithm via Horner's rule:

Horner's rule for polynomial evaluation, mentioned in section 2.3.3 and 4.2.6, will be reviewed here. It is also called a nested evaluation of a polynomial and is given by

$$y(p)=((.((a_{N-1}p+a_{N-2})p+\ldots+a_2)p+a_1)p+a_0 \qquad (4.1)$$

where a_n is the n^{th} coefficient of the polynomial $y(p)$, of order N-1, and where p is any point in the complex plane at

which y(p) is to be evaluated. This algorithm avoids the need to compute the powers of p as these are in effect computed recursively.

The discrete Fourier transform can be rewritten using Horner's rule. The DFT (2.1) can be expanded and the terms rearranged as in (4.1). It follows that

$$X(k)=(\ldots((x(N-1)W_N^k+x(N-2))W_N^k+\ldots +x(2))W_N^k+x(1))W_N^k+x(0) \qquad (4.2)$$

where x(n) replaces a_n and W_N^k replaces p. This sequence of operations can be described as a linear difference equation in the form of

$$y(n,k) = W_N^k\, y(n-1,k) + x(N-n) \qquad 0\le k\le N-1 \qquad (4.3)$$

with initial condition y(0,k)=0 and with the index n going from 1 to N, the desired result being the solution at n=N. This is

$$X(k) = y(N,k) \qquad 0\le k\le N-1 \qquad (4.4)$$

In order to process a continuous flow of data sequences, the polynomial (4.2) should be evaluated with the data samples in the proper order, x(0) first and x(N-1) last. This is illustrated by rewriting (4.2) with the samples in the proper order and by replacing X(k) by A(k).

$$A(k)=(\ldots((x(0)W_N^k+x(1))W_N^k+\ldots+x(N-2))W_N^k+x(N-1) \qquad (4.5)$$

It is possible to find the output samples X(k) from A(k), (4.5) can be expressed in closed form as

$$A(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{(N-1-n)k} \qquad \text{where } 0 \le k \le N-1 \qquad (4.6)$$

By expanding and simplifying the complex exponential, we get

$$A(k) = W_N^{-k} \sum_{n=0}^{N-1} x(n)\, W_N^{-nk} \qquad \text{where } 0 \le k \le N-1 \qquad (4.7)$$

The symmetry property of the DFT (section 2.2) is applied so that (4.7) results in

$$A(k) = W_N^{-k}\, X(-k) \qquad (4.8)$$

The DFT of the sequence x(n) is retrieved from (4.8) by a change in variable and by rearranging the terms. The output samples of the DFT are now given by

$$X(k) = W_N^{-k}\, A(-k) \qquad (4.9)$$

or as a recursive equation where A(-k) becomes

$$y(n,k) = W_N^{-k}\, y(n-1,k) + x(n) \qquad \text{where } 0 \le n \le N-1,\ 0 \le k \le N-1 \text{ and } y(-1,k)=0 \qquad (4.10)$$

and (4.9)

$$X(k) = W_N^{-k}\, y(N-1,k) \qquad 0 \le k \le N-1 \qquad (4.11)$$

(4.10) expresses a different way of deriving the first order recursive realization of the Goertzel algorithm (section 2.3.2.1). As noted in that section and also in [57] the direct application of the Goertzel algorithm leads to a correct amplitude, but the phase requires correction, as shown in (4.11).

4.3.3) Modified first order Goertzel algorithm:

(4.11) can be simplified by eliminating the term W_N^{-k}, and by combining (4.10) and (4.11) to form (4.12):

$$X(k)=W_N^{-k}[(.((x(0)W_N^{-k}+x(1))W_N^{-k}+\ldots +x(N-2))W_N^{-k}+x(N-1)] \quad (4.12)$$

Now, the isolated term W_N^{-k} on the right hand side of (4.12) can be moved inside the squared brackets such that it multiplies one term of the polynomial at a time. By doing so, we end up with the following identity,

$$X(k)=[((.((x'(0)W_N^{-k}+x'(1))W_N^{-k}+\ldots +x'(N-2))W_N^{-k}+x'(N-1)] \quad \text{where } x'(n)=x(n)\ W_N^{-k} \quad (4.13)$$

The polynomial in (4.13) looks like that in (4.2), it uses Horner's rule but the order in which the data samples are used is changed, the phase correction is applied to each term and finally ,the exponent of the twiddle factors is now positive $W_N^{-k}=e^{j2\pi k/N}$.

The sequence of operations in (4.13) is best described as a linear 1st order recursive equation in the form of

$$y(n,k)=W_N^{-k}\ \{x(n)+y(n-1,k)\} \quad \text{where } 0\le k\le N-1, \ 0\le n\le N-1 \text{ and } y(-1,k)=0 \quad (4.14)$$

The DFT samples are obtained after N iterations or simply

$$X(k) = y(N-1,k) \qquad 0\le k\le N-1 \quad (4.15)$$

We shall now see how the proposed DFT algorithm in mapped onto systolic arrays.

4.3.4) Systolic array implementations:

The modified first order Goertzel algorithm avoids the use of a phase correction which was required for the first order Goertzel algorithm (sections 2.3.2.1 and 4.3.3). The task of mapping the DFT onto systolic arrays using the modified Goertzel algorithm is simplified. The resulting two arrays incorporate all properties that make the design compatible with VLSI, such as modularity (homogeneity), pipelining, simple and local communications (section 3.3). A semi-systolic array (figure 4.7) and a pure-systolic array (figure 4.9) are now presented.

4.3.4.1) Semi-systolic array:

Figure 4.7 depicts the semi-systolic array for a 3-point DFT. The data samples flow through the array in a pipeline fashion. Each data sample is used by a cell for the computation of one DFT sample. The data samples are then shifted out to their nearest neighbor. A new data sample is therefore available at each cell for the computation of a DFT sample. Two frames composed of 3 samples each are used to show the process of computing a DFT: {a(0), a(1), a(2)} and {b(0), b(1), b(2)}. The resulting DFT vectors are given by {A(0), A(1), A(2)} and {B(0), B(1), B(2)}.

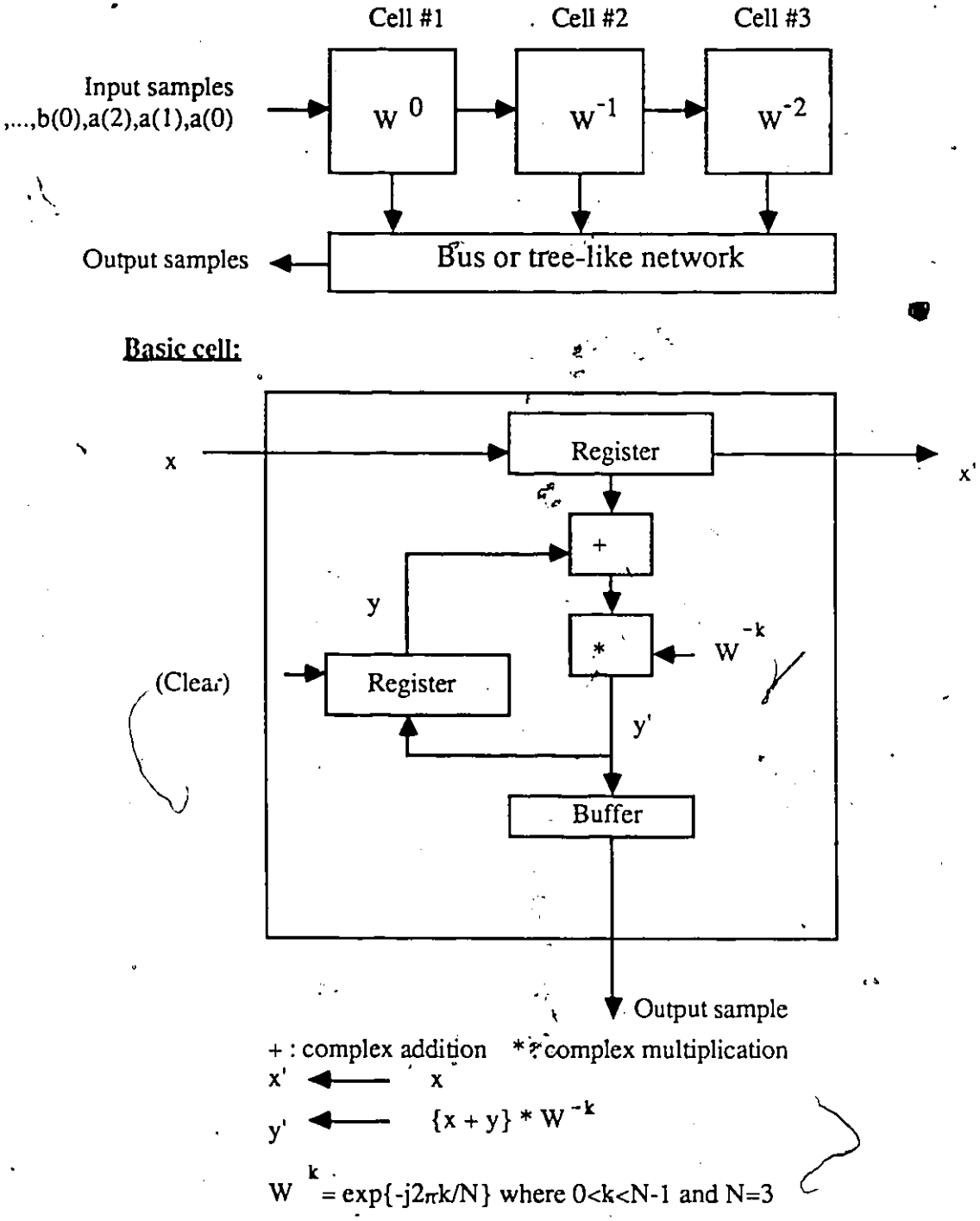


Figure 4.7 Semi-systolic array for computing the DFT using the modified first order Goertzel algorithm, N=3.

All cells accomplish the same operation, mainly (4.14). The basic cell shown on figure 4.7 presents the different operations required for the computation of a frequency sample in complex notation. Section 4.4 will give more details on the actual realization in terms of real operations. Only one twiddle factor is stored per cell in the array. The cells contain one fixed W_N^{-k} each.

Figure 4.8 gives the state of the systolic array at successive clock cycles. At each cell, the recursive equation (4.14) is performed three times. The result is then available at the cell's output through a buffer. The DFT coefficients appear skewed in time. In the mean time, right after the frequency sample has been sent to the cell's buffer, the cell's register in the recursive path is cleared thus making the cell ready for the computation of a new frequency sample of the subsequent 3-point vector.

As mentioned in section 3.5, a bus or a tree like network can be used to collect the results. If parallel arithmetic and parallel data exchange are assumed then, the latency time for an N-point DFT is equal to N clock cycles. The first complete vector is made available after 2N clock cycles and any subsequent vector, after N clock cycles.

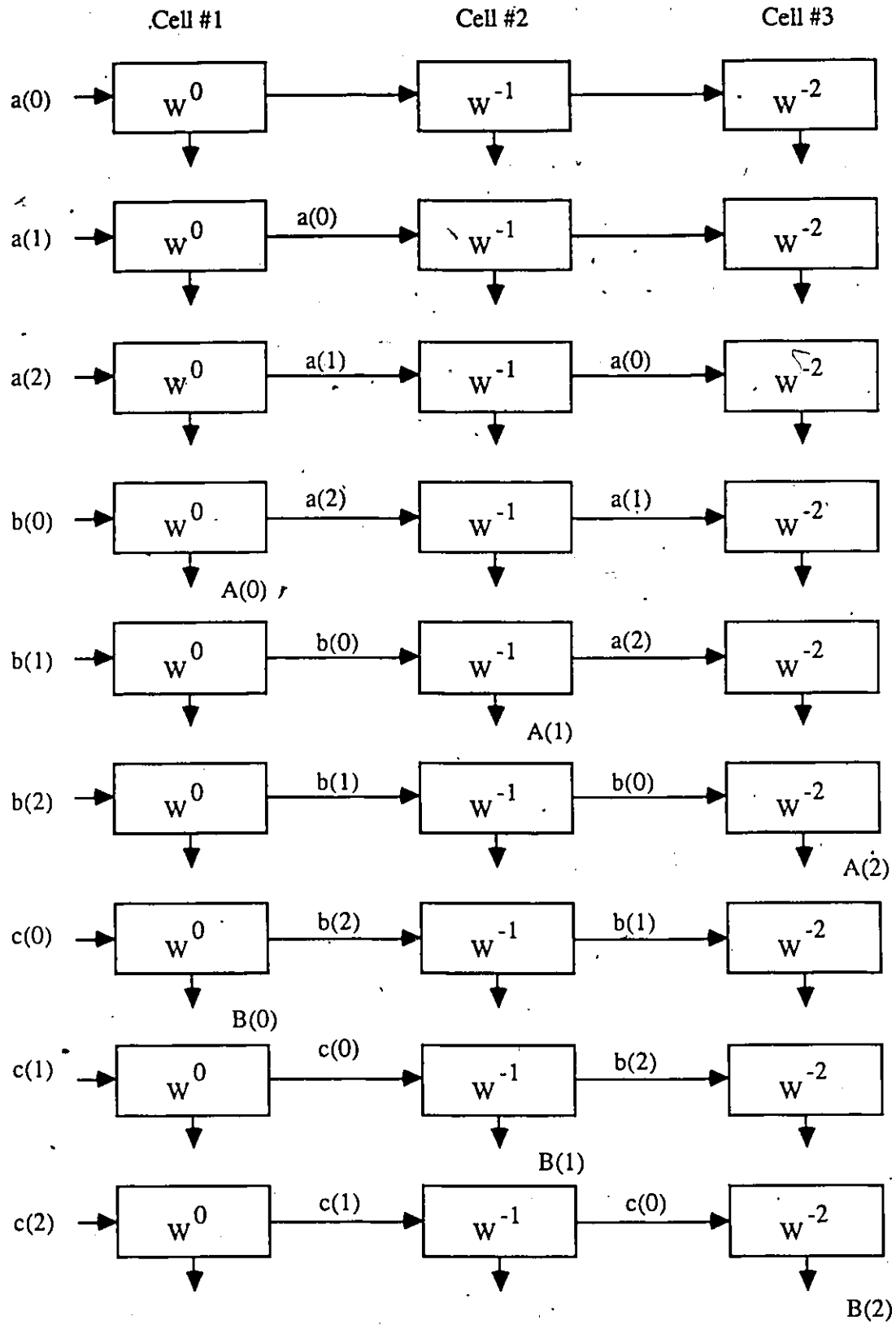


Figure 4.8 Semi-systolic array at successive clock cycles, N=3.

4.3.4.2) Pure-systolic array:

It is possible to avoid the bus or tree-like network found in the semi-systolic array by adding two registers and some combinatorial logic in each cell, except in the last cell. Figure 4.9 illustrates the systolic array resulting from this modification when N=3. It mainly shows the input and output paths of the array. Each cell contains one twiddle factor, W_N^{-k}. The input samples enter the array, one every clock cycle. On the diagram they are given by a(n) and b(n).

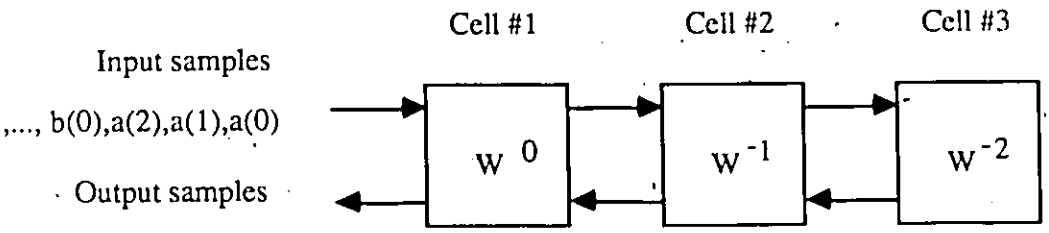
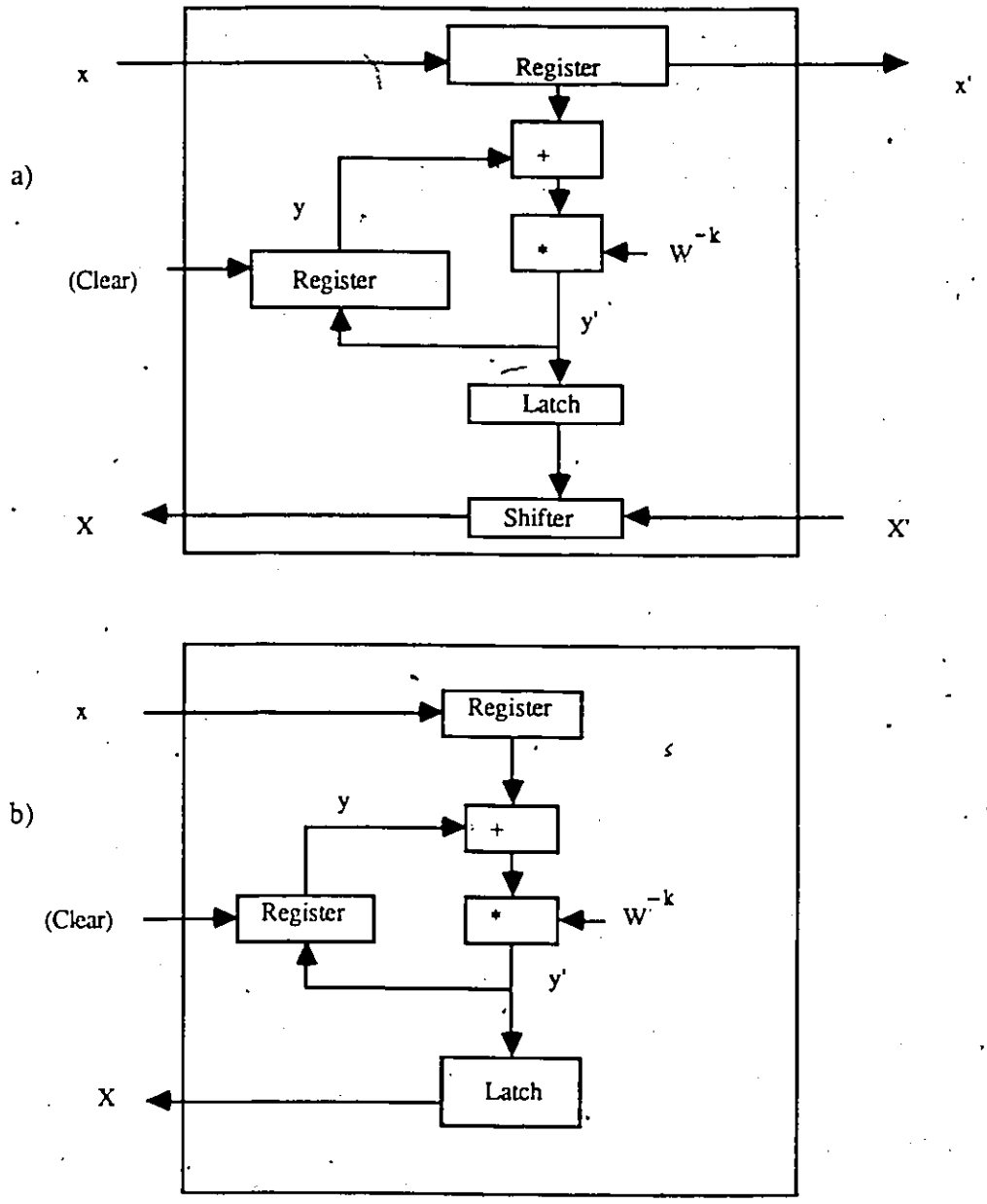


Figure 4.9 Pure-systolic array for computing the DFT using the modified first order Goertzel algorithm, N=3.



+ : complex addition * : complex multiplication

$x' \leftarrow x$

$y' \leftarrow \{x + y\} * W^{-k}$

$W^{k} = \exp\{-j2\pi k/N\}$ where $0 \leqslant k \leqslant N-1$ and $N=3$.

Figure 4.10 Cell details for the pure-systolic array , a) cells #1 to 2, b) cell #3.

Figure 4.10 shows the two types of cells. The array requires two identical cells; the last one has only one latch with no shift register (shifter).

The data samples flow through the array in a way similar to that of figure 4.7. The difference with the other systolic array is found in the manner in which the DFT samples are evacuated. The stage of latches keeps the DFT samples from interfering with those of the preceding transform which are being shifted out of the array through the stage of shift registers. Global clocks control the registers, latches and shift registers (not shown on the diagram, but will be discussed in section 4.4). Figure 4.11 depicts the process using a timing diagram similar to the one seen on figure 4.8.

Assuming parallel arithmetic and parallel data exchange, the DFT samples leave the array in sequence every clock cycle but after an initial delay (latency) of five clock cycles. For an N-point DFT transform, N basic cells containing one twiddle factor each are required. The latency is equal to 2N-1 clock cycles. Successive vectors are computed every N clock cycles.

Further design details e.g. throughput rate per sample are given in the following section for this particular DFT systolic array.

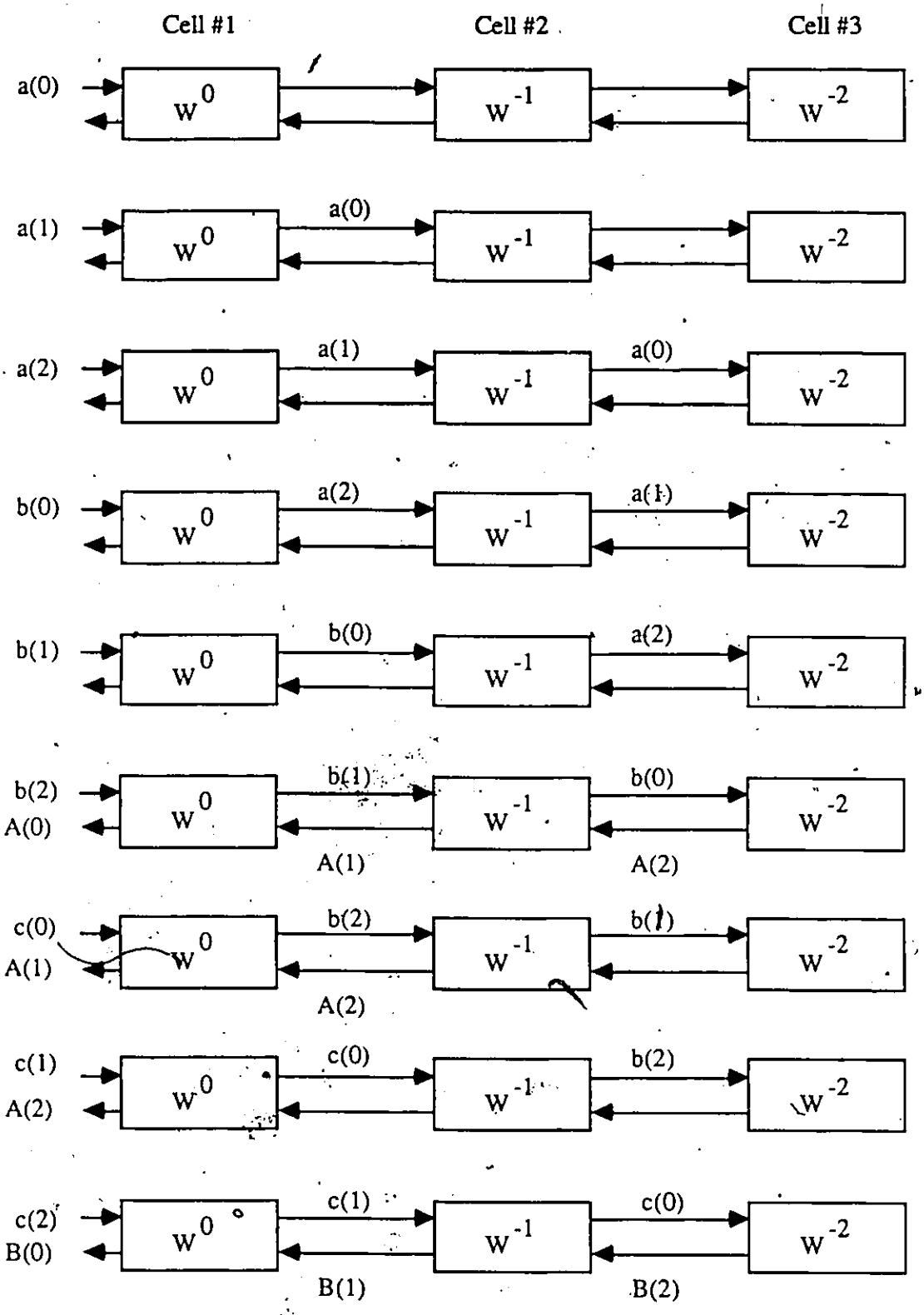


Figure 4.11 Pure-systolic array at successive clock cycles, N=3.

4.4) Semi-custom designs:

4.4.1) Generalities:

It is difficult to put a price tag on the VLSI systolic arrays that have been presented in section 4.3, figures 4.7 and 4.9 . The actual VLSI implementation is not within the scope of this thesis. But in the available literature on VLSI, one finds that cost-effectiveness of special-purpose arrays such as the DFT algorithm is mostly determined by the application of the design guidelines summarized in section 3.3. The cost of these arrays must be kept low enough to justify their limited range of application.

Nevertheless, semi-custom implementations of a pure-systolic array (section 4.3.4.2 and figure 4.9) based on the modified Goertzel algorithm with emphasis on the design of the basic cells will be discussed in this section. The types of arithmetic operations to be implemented and the formats for information exchange are presented. For the semi-custom designs presented below, we assume that the library of basic building blocks is composed of components available on the market. Details like throughput rate per sample and power consumption are given.

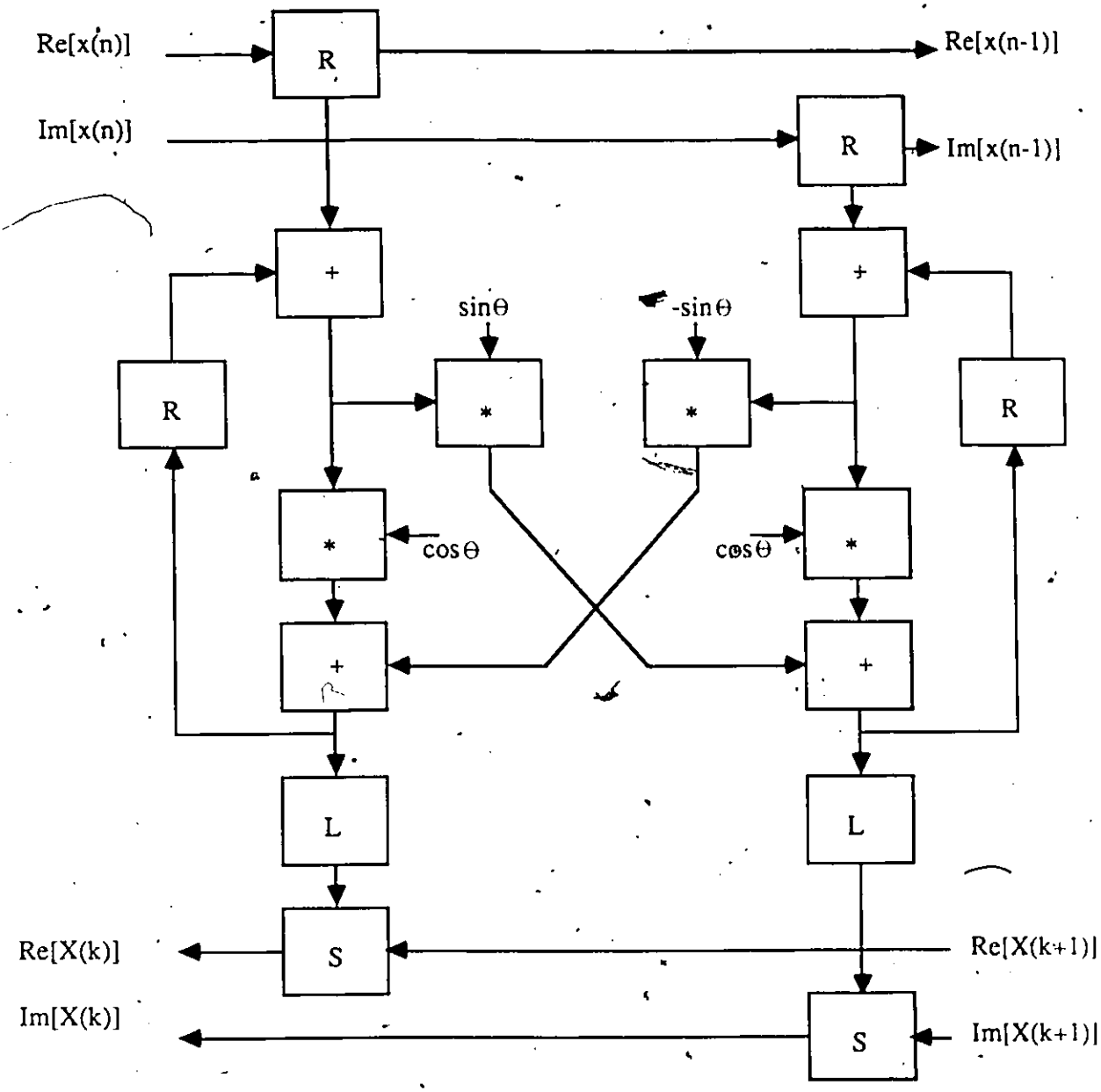
4.4.2) Structure of a basic cell:

The cells presented in figures 4.7 and 4.10 perform (4.14). We now show two possible hardware implementations for (4.14) incorporating complex signals, and consider either hardware multiplier or stored product ROM usage. Expand (4.14) into its real and imaginary parts:

$$\mathrm{Re}[y(n,k)] = \{\mathrm{Re}[x(n)]+\mathrm{Re}[y(n-1,k)]\}\cos\theta + \{\mathrm{Im}[x(n)]+\mathrm{Im}[y(n-1,k)]\}\sin\theta \quad (4.16)$$

$$\mathrm{Im}[y(n,k)] = \{\mathrm{Re}[x(n)]+\mathrm{Re}[y(n-1,k)]\}\sin\theta + \{\mathrm{Im}[x(n)]+\mathrm{Im}[y(n-1,k)]\}\cos\theta \quad (4.17)$$

It can be seen that four adders and four multipliers are required per cell. Figure 4.12 depicts a basic cell. Figure 4.13 shows the complete pure-systolic array of figure 4.9 for N=3. Cell #3 has the same arithmetic section as the other cell but the stage of shift registers indicated on the diagram by "S" is not required.



R: register , L: latch register., S: shift register

$\theta = 2\pi k/N$

FIGURE 4.12 Real arithmetic operations involved in a basic cell.

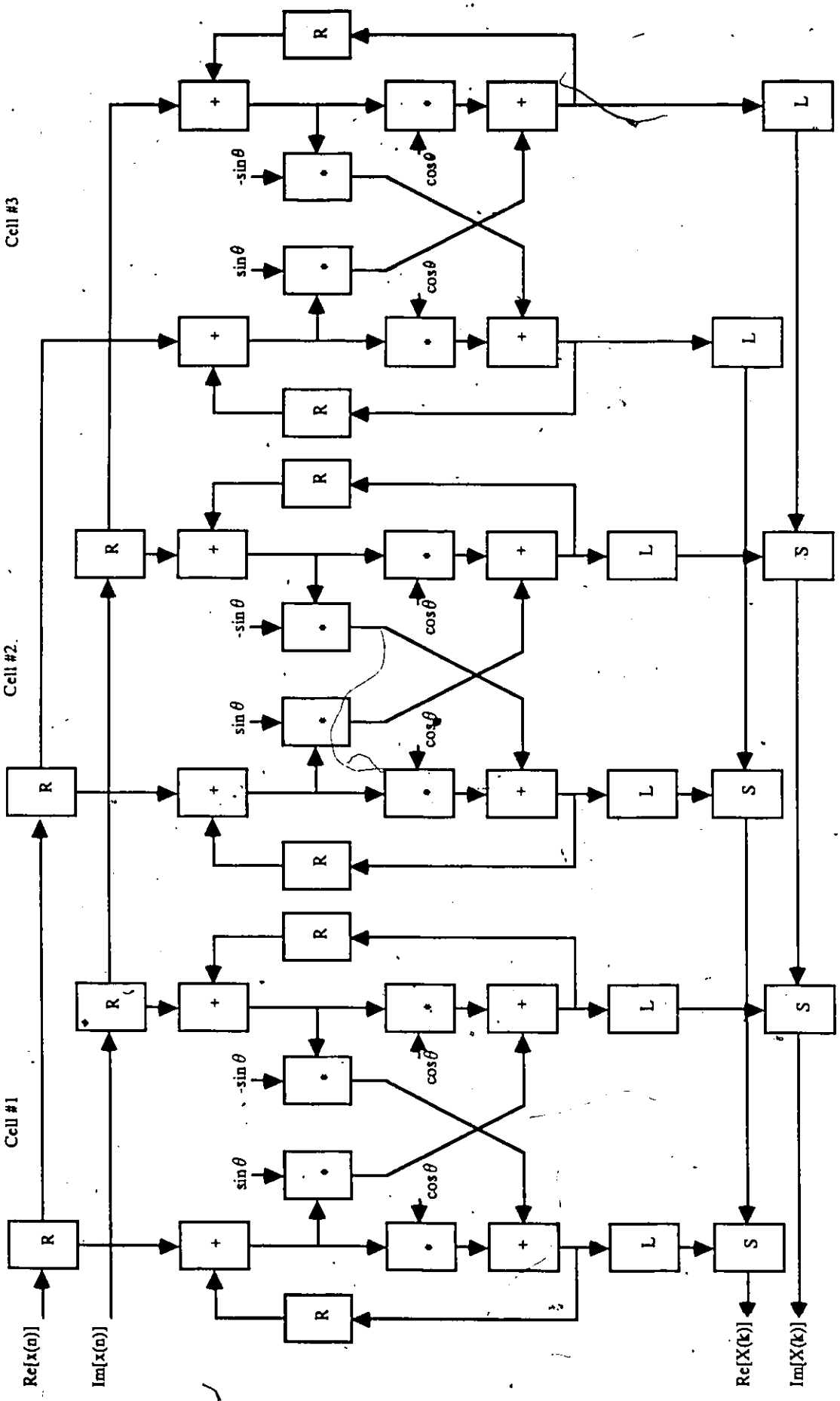


Figure 4.13 Details of the DFT pure-systolic array for N=3.

4.4.3) Hardware multipliers:

Hardware multipliers are among the first IC's to benefit from recent advances in VLSI CMOS technology. CMOS multipliers have now replaced the first multipliers fabricated by TRW using bipolar technology. They are in fact pin for pin equivalent to the standard established by the TRW popular line of multipliers. Furthermore, for the same wordlength and speed, they achieve power consumptions which are 1/25 of those found for bipolar equivalents and 1/15 of their NMOS counterparts.

Appendix I gives some hardware multipliers now available on the market. They handle product sizes from 8 by 8 bits up to 24 by 24 bits in fixed point, 32 and 64 bits in floating point with speeds starting at 235 ns down to 65ns and power consumptions ranging from 75 mW up to 500 mW per IC. The arrival of new companies in the business has caused a decline in pricing. The normal price per IC for quantities in 100's used to be around $150 CDN. But it has recently gone down to $62 for comparable performance.

With a multiplier, both the coefficient and partial results are limited by the input word length of the IC. The coefficients cosθ and sinθ have therefore to be rounded to a specific word length. This will introduce some errors in the calculation of a DFT, which can be avoided by the use of

stored product ROM's.

4.4.4) Stored product ROM's:

An interesting way to implement a multiplier when coefficients in the array are static, is by means of the stored product technique [12,33]. With this technique, all possible products of the input signal and a fixed coefficient are rounded and stored in a real-only-memory (ROM). These results are computed off-line with the accuracy available on a main frame, usually double-precision (64 bits). This method eliminates the error introduced when coefficients are rounded prior to multiplication.

Appendix II summarizes the characteristics of the ROM stored product technique for different word lengths. It can be seen from the tables that CMOS has not yet proved its full potential for high speed ROMs. For small memory sizes (4K*4) TTL offers the best speed figures but the highest power dissipation. For higher densities (32K*8), (64K*8) ,(64K*16) CMOS technology appears to be the only route. Its power dissipation is in the range 150 to 1150 mW and access times from 100 to 250 ns. On the average , prices of ROM stored product implementations are lower than those of hardware multipliers. On the other hand, speed-power products of ROMs are still higher than those of hardware multipliers either for TTL or CMOS technologies.

The newest ROM on the market is the (64K*16) on one IC. This memory IC will eventually attain its full maturity in a year or so. One should therefore watch carefully the market of big ROM ICs (including PROMs and EPROMs), as access time, power dissipation and prices decline rapidly [9-13].

4.4.5) Design examples:

This section gives an insight into the throughput rate per sample and power dissipation achievable with a semi-custom implementation of the DFT pure-systolic array. The analysis is focused on the two possible implementations of a basic cell (figure 4.12) described earlier i.e.hardware multipliers and stored product ROM's which realize (4.16-17).

Complex input samples were chosen to be 8-bit words and output samples either 12-bit or 16-bit words. High speed CMOS registers and adders are common to both implementations, figures 4.14 and 4.15.

Figure 4.16 shows the timing diagram for cell #1 which applies to both realizations. In this diagram the elements of figures 4.14 and 4.15 are assumed to be synchronized to the positive edge of the clock signals. The timing diagram for cell #2 can be derived from figure 4.16 by simply introducing a lag of one clock cycle; for cell #3, 2 clock cycles are required.

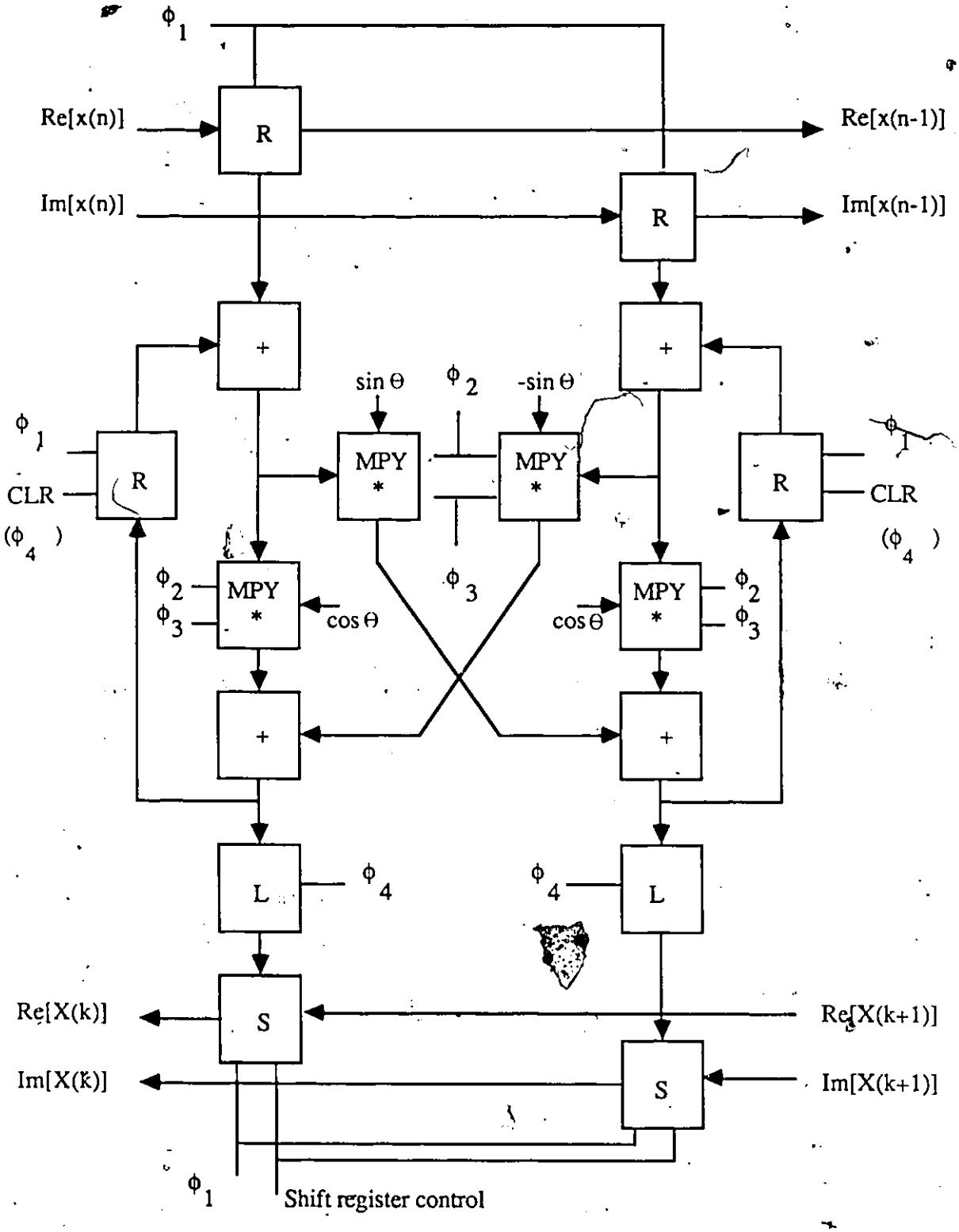


Figure 4.14 Details of cell #1 and #2 for the DFT pure-systolic array implemented with hardware multipliers, N=3.

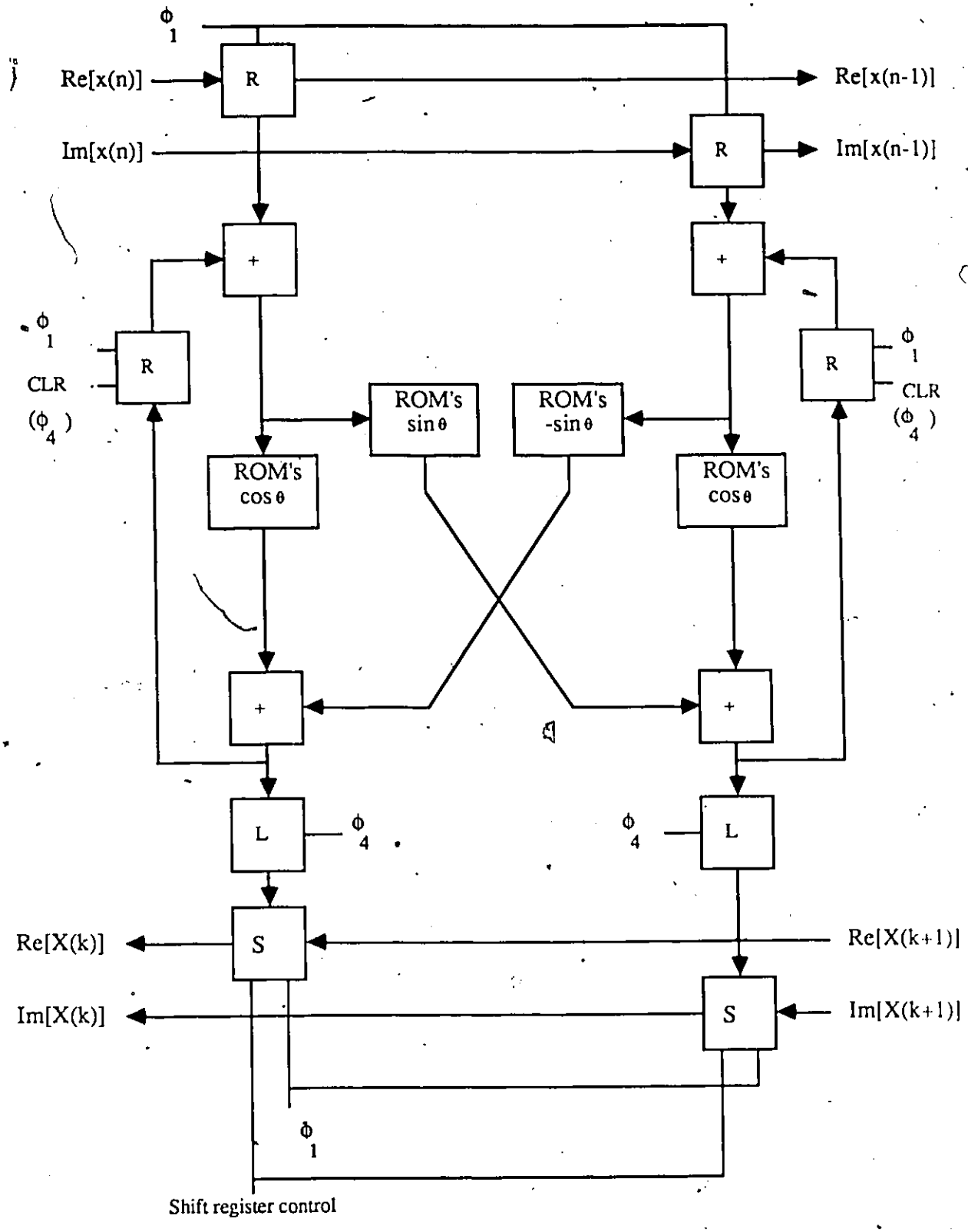
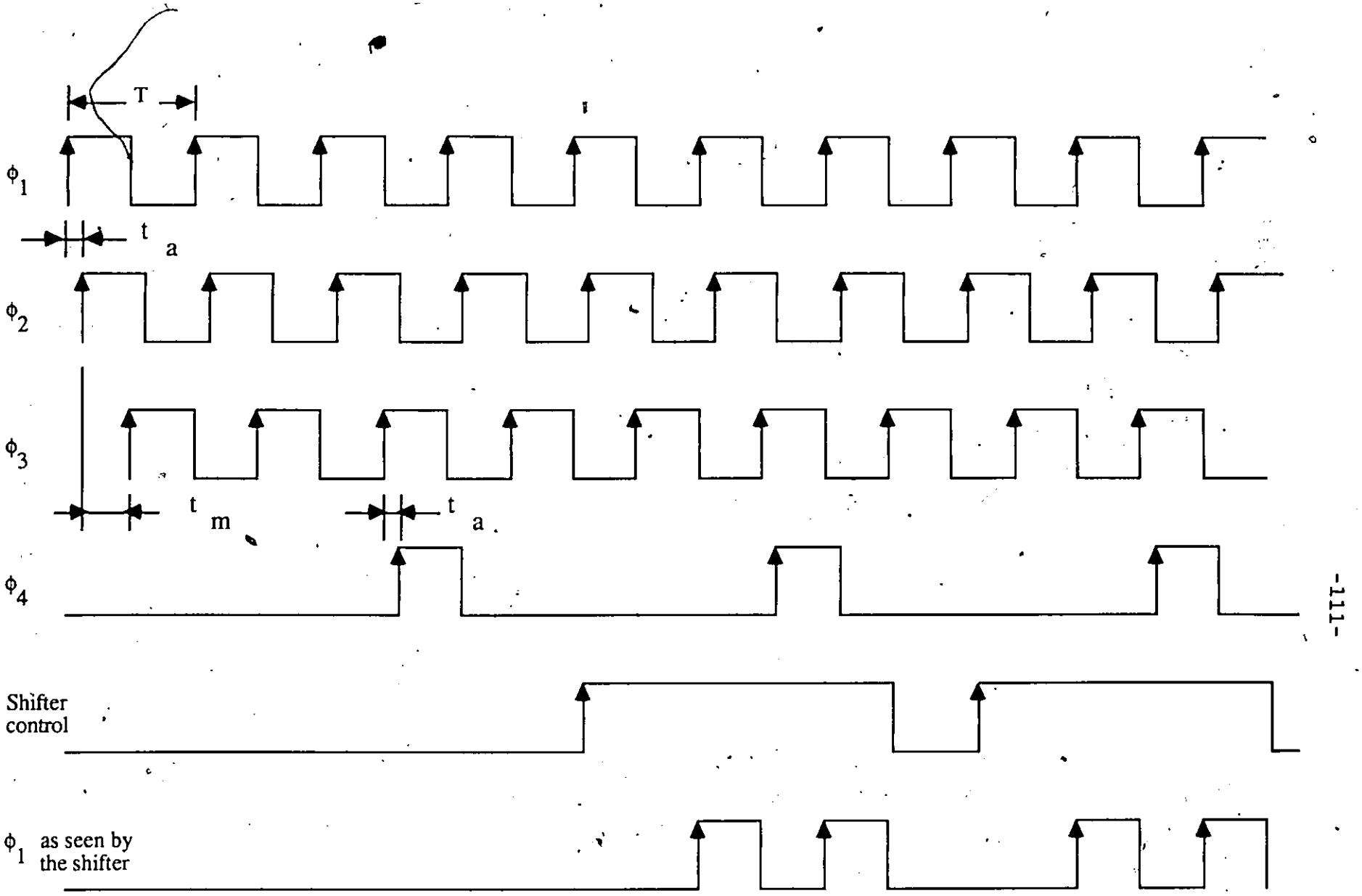


Figure 4.15 Details of cell #1 and #2 for the DFT pure-systolic array implemented with stored product ROMs, N=3.



t_a : time for an addition, t_m : time for a multiplication

Figure 4.16 Details of the timing diagram for the DFT pure-systolic array, cell #1 and N=3.

Table 4.4 IC specifications for the two possible implementations of the DFT pure-systolic array.

Symbol	Function	Part number	Word-length (bits)	Delay (ns)	Power (mW)
R L S	Register Latch Shift-register	one+half 74HC377	12	15	0.04
		two-74HC377	16	15	0.04
+	Adder	three 74HC283	12	35	1.2
		four 74HC283	16	45	1.6
*	Stored product ROM	three-16K*4	12	t_M	P_M
		two-64K*8 or one-64K*16	16	t_M	P_M
	Hardware multiplier	one 12-bit	12	t_M	P_M
		one 16-bit	16	t_M	P_M

Note: For t_M and P_M, appendices I and II must be used.

Using the specifications given in table 4.2 and figures 4.14-16, one can write the following equations for the total delay and total power dissipation per cell,

1) 12 bit arithmetic,
t_d =delay for R+2(delay of adder)+t_M=85+t_M (4.18)
P =8(diss. for R)+4(diss. of adder)+4P_M=5.12+4P_M (4.19)

2) 16 bit arithmetic,
t_d =delay for R+2(delay of adder)+t_M=105+t_M (4.20)
P =8(diss. for R)+4(diss. of adder)+4P_M=6.72+4P_M (4.21)

Table 4.3 gives the figures of some designs according to the following two criteria (i) minimum delay (ii) minimum power dissipation.

Table 4.3 Performance of optimized designs based on the ICs given in table 4.2 and appendices I and II.

		12 bits		16 bits	
CRITERIA		Hardware	ROM	Hardware	ROM
Min. Delay	Part #	LMU12PC-1	82HS195	MPY-016K	Amd27C10 24
	Delay (ns)	150	110	130	255
	Throughput rate per sample (MHz)	6.67	9.1	7.7	3.92
	Operations per second per cell (4 add + 4 mult.)	53.30MOPS	72.7	61.5	31.4
	Power dissipation per cell (mW)	400	8700	16000	1000
Min. Power	Part #	same as above	MB7134	MP1010	same as above
	Delay (ns)		135	160	
	Throughput rate per sample (MHz)		7.4	6.25	
	Operations per second per cell (4 add + 4 mult.)		59.3MOPS	50	
	Power dissipation per cell (mW)		3600	500	

Table 4.4 summarizes some designs performance for N=16 and N=256. These sequence lengths were chosen so that the design performance can be compared with other types of design e.g. other systolic arrays and FFT based systems. The systolic arrays of figures 4.7 and 4.9 can in fact compute DFTs of any size.

Table 4.4 Design performance for N=16 and N=256.

	N=16		N=256		
Design based on	Throughput per vector	Power diss.	Throughput per vector	Power diss.	Remarks
LMU 12PC-1	2.25 µs	4.8 W	38.4 µs	100 W	12 bits
82 HS195	1.76 µs	104 W	28.0 µs	---	12 bits
MB 7134	2.16 µs	43 W	34.6 µs	---	12 bits
MPY 016K	2.08 µs	192 W	33.3 µs	---	16 bits
Amd 27C1024	4.08 µs	12 W	65.3 µs	252 W	16 bits
MP 1010	2.56 µs	6 W	41.0 µs	126 W	16 bits

4.5) Summary:

This chapter has been concerned with systolic array implementation of the discrete Fourier transform. A survey of existing systolic arrays has been presented. A two-dimensional systolic array for the DFT has shown a considerable I/O overhead when compared to one-dimensional DFT arrays. One-dimensional DFT systolic arrays present basically two types of I/O interconnections: semi-systolic arrays and pure-systolic arrays. The simplest are the pure-systolic arrays; one input sample and one output result per clock cycle.

Subsequently, the existing DFT systolic arrays have been compared to FFT-based systems using either a perfect shuffle network or a fully parallel approach. The comparison has been made upon the following criteria: (i) number of PEs, (ii) complexity of a PE, (iii) latency time, (iv) throughput rate per vector, (v) upper bound on the chip area, (vi) external memory requirements and (vii) communication complexity between the processing elements. FFT-based systems were found to yield the highest throughput rates but on the other hand overwhelming external and internal communication requirements implied the highest system area figures. Instead, DFT systolic arrays present the best area-efficient design. But, due to the need of a

pre-loading phase or because of low processor utilization, all of them are not fully compatible with real-time signal processing requirements.

Keeping in mind the requirements for minimum system area and for real-time signal processing, two new systolic arrays for the computation of the DFT have been outlined. They both require N basic cells. The first one is a semi-systolic array in which a data sample is entered every clock cycle and successive results appear skewed in time on a bus or tree-like network after an initial delay of N clock cycles where N is the sequence length. A second array that uses the same cell arrangement but is configured as a pure-systolic array has then been introduced. This array makes use advantageously of two levels of registers which permit the results to be outputted from the same set of lines.

Based on a new algorithm called the modified first order Goertzel algorithm, the systolic arrays do not require a pre-loading phase at each DFT calculation. Furthermore, utilization of the incoming data samples in the natural order as obtained from an analog to digital converter (A/D) or the combination of an A/D and a window function make the designs suitable for processing a continuous flow of vectors.

It as been demonstrated that a basic cell can be implemented either by hardware multipliers or by a ROM

stored product technique. The latter implementation has been shown to be attractive because of the new improvements made to ROM's access time and power dissipation, and also to the decline of prices which is usually more abrupt than that of hardware multipliers. Different semi-custom designs have therefore been developed and their performance assessed in terms of throughput rate per sample and power dissipation.

Moreover, the proposed systolic arrays based on the modified first order Goertzel algorithm can compute a DFT of any length on any circular contour or any segment of a particular contour. Inverse Fourier transforms can also be implemented using the same systolic array but with the twiddle factor's sign changed (permutation of $\sin\theta$ by $-\sin\theta$ and vice-versa) and with the introduction of a scaling factor $1/N$ (2.2).

All the design examples used fixed point arithmetic. This approach simplifies the system (including the VLSI realization). The errors introduced by such an approach and in a general sense by a digital architecture are to be discussed in the next chapter.

CHAPTER V

PERFORMANCE ANALYSIS OF ONE-DIMENSIONAL DFT SYSTOLIC ARRAYS

5.1) Introduction:

In this chapter, the analysis and the simulation of the effects of rounding for fixed point arithmetic is presented for the first and the second order realization of the Goertzel algorithm along with an analysis of overflow conditions (section 5.2). The Goertzel algorithm is chosen because it is representative of most systolic arrays useful for DFT analysis mentioned in section 4.2 e.g. Kung's pure-systolic array [51] (section 4.2.6). Subsequently, a similar analysis is presented in section 5.3 for the modified first-order Goertzel algorithm.

The effect of signal quantization is not considered here because its effects are independent of the structure of the array. The error analysis for coefficient quantization is omitted because of high complexity and because for stored product ROM realization this error is not present. The study is limited to giving simulation results and finding the influence of coefficient quantization on the DFT output samples computed using fixed point arithmetic.

5.2) Fixed point error analysis of existing DFT systolic arrays by means of the Goertzel algorithm:

5.2.1) Generalities:

As seen in chapter 2, the Goertzel algorithm implements the DFT as a finite impulse response recursive filter. Here we propose to study the signal-to-noise ratio (SNR) for the first and second order realization of the Goertzel algorithm. The analysis assumes that the N frequency samples are computed by N basic cells; one frequency sample per cell. Partial results are therefore rounded to B bits after every recursion and kept in the basic cell.

The first-order Goertzel algorithm requires basically a sequence of multiply and accumulate operations. This same operation is found in almost all the existing systolic arrays discussed in section 4.2. Therefore, there is no need to repeat the error analysis for each systolic array. The error analysis of the first-order algorithm is sufficient. The error analysis of the second-order algorithm is introduced because of its possible use in some systolic arrays.

Appendix III gives the details of the analysis for a linear shift invariant system. The same methodology will be used here to find the SNR.

5.2.2) First order Goertzel algorithm:

The first order Goertzel algorithm was seen in sections 2.3.2.1 and 4.3.2.

The equations describing a cell as in (2.17)-(2.18) and in (4.10)-(4.11) can be expanded into the real and imaginary parts. Assuming complex input samples, we write the following equations,

$$Re[y(n,k)] = Re[y(n-1,k)]\cos\theta - Im[y(n-1,k)]\sin\theta + Re[x(n)] \quad (5.1a)$$

$$Im[y(n,k)] = Re[y(n-1,k)]\sin\theta + Im[y(n-1,k)]\cos\theta + Im[x(n)] \quad (5.1b)$$

where $0 \le n \le N-1$ and $0 \le k \le N-1$

and,

$$Re[X(k)] = Re[y(N,k)]\cos\theta - Im[y(N,k)]\sin\theta \quad (5.2a)$$

$$Im[X(k)] = Re[y(N,k)]\sin\theta + Im[y(N,k)]\cos\theta \quad (5.2b)$$

where $0 \le k \le N-1$ and $0 \le k \le N-1$.

Figure 5.1 depicts the model for one basic cell. The complex output signal y(n,k) indicates that the k^{th} frequency sample is being considered.

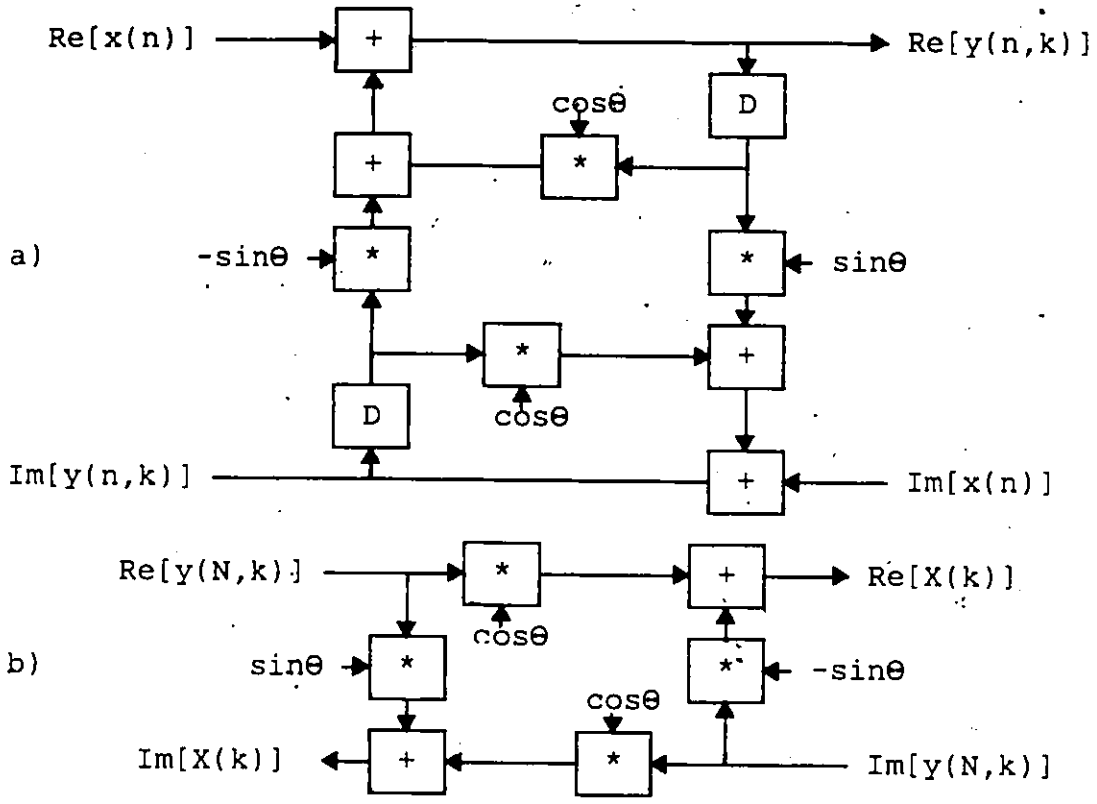


Figure 5.1. Basic cell for the realization of the first order Goertzel algorithm, a)recursive section implementing 5.1 and b) FIR section implementing 5.2.

Using the roundoff noise model of a multiplier, figure III.1 (Appendix III), the linear system above is transformed into an equivalent model that includes the noise sources as in figure 5.2. Because the FIR network shown in figure 5.1 (b) is performed only once, it has been omitted from figure 5.2. But it is accounted for in the error variance calculation. In fact, four additional error sources e_5, e_6, e_7 and e_8, one after each multiplier, are considered.

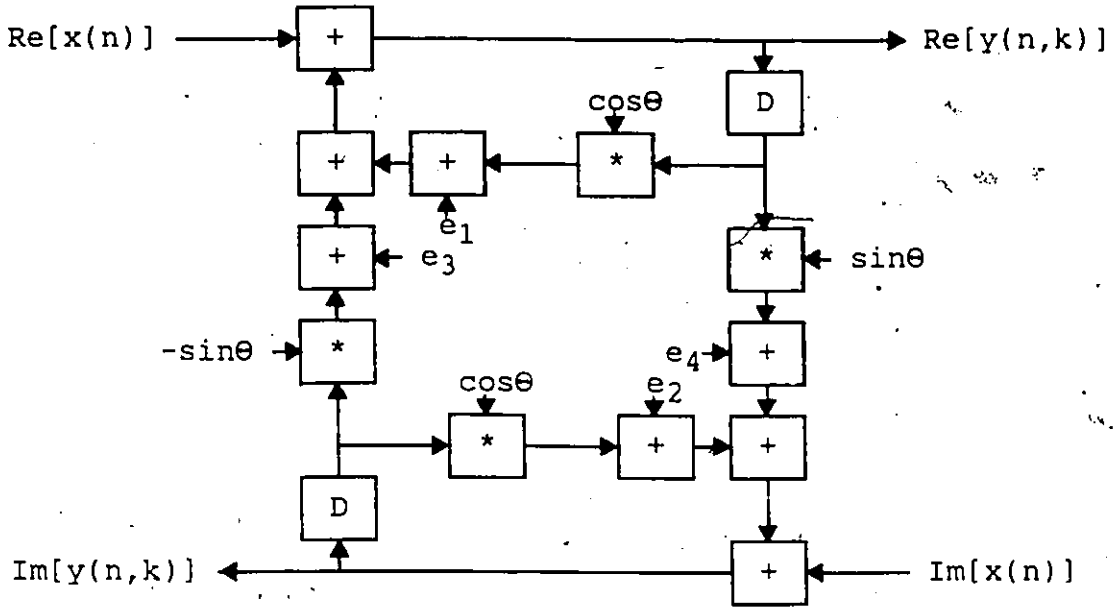


Figure 5.2. Statistical model of the 1st order Goertzel algorithm cell.

In figure 5.2, for clarity, only the exact output signal, y(n,k), is shown. But as explained in Appendix III, a second term is present, that is the output error signal due to the noise sources.

Mason's rule [14] can help us in finding the unit-sample responses necessary to the analysis. It can be verified that these are given by (5.3)-(5.8). The subscript "ek" indicates the error source "k" and Re[x(n)], Im[x(n)] are respectively the real part and the imaginary part of the input signal. The $h_{ek}(n)$ are the complex unit-sample responses from the noise sources to the output of a cell (figure 5.2). For example, $Re[h_{e1}(n)]$ is the unit-sample response from noise source $e_1(n)$ to the output Re[y(n)].

$$Re[h_{e1}(n)]=Re[h_{e3}(n)]=Im[h_{e2}(n)] = Im[h_{e4}(n)]= \cos(\Theta n) \quad (5.3)$$

$$Re[h_{e2}(n)]=Re[h_{e4}(n)]= -\sin(\Theta n) \quad (5.4)$$

$$Im[h_{e1}(n)]=Im[h_{e3}(n)]= \sin(\Theta n) \quad (5.5)$$

The complex unit-sample responses from the input to the output of a cell are named "g". For example, $Re[g_{xr}(n)]$ is the unit-sample response from the input signal Re[x(n)] to the real output.

$$Re[g_{xr}(n)]=Im[g_{xi}(n)]= \cos(\Theta n) \quad (5.6)$$

$$Re[g_{xi}(n)]= -\sin(\Theta n) \quad (5.7)$$

$$Im[g_{xr}(n)]= \sin(\Theta n) \quad (5.8)$$

In the equations above , $\Theta=2\pi k/N$ and, k and n vary between 0 to N-1. Substituting (5.3)-(5.5) in (III.2), the following expression is found for the variance of the output noise signal:

$$\sigma^2_f = (N+1)\ 2^{-2B}\ /6 \quad (5.9)$$

This equation was found by adding the effects of the error sources in the recursive structure, figure 5.1 (a), and of the error sources from the last stage of the basic cell, figure 5.1 (b).

To determine σ^2_y, the output signal variance, we first have to evaluate the scaling factor that must be applied to the input signal in order to prevent dynamic range problem (overflows). From figure 5.2, we see that there are four critical nodes and the conditions for no overflow resume to

the following expression,

$$x_{max} < \frac{1}{\sum_{n=0}^{N-1} |\cos(\theta n)| + |\sin(\theta n)|} \qquad (5.10)$$

The right hand side term in equation (5.10) varies between the following extremes, $1/N$ for $\theta=0,\pi/2$ and $1/(\sqrt{2}\ N)$ for $\theta=\pi/4,3\pi/4$. These bounds are associated with worst case sequences.

In the case of DFTs performed on real time signals, additional care must be taken in preventing the occurrence of overflows in digital systems using fixed point arithmetic. This is so, because most of the real time signals that convey information are not totally random. There is always some sort of coherence in the signals. A study of the nature of the signals involved is required in that situation [22].

Using (III.7), the expression for the output signal variance, and the results above, it follows that

$$\sigma^2_y = \sigma^2_x\ N \qquad (5.11a)$$

with

$$\sigma^2_x = (\text{scaling factor})^2/3 \qquad (5.11b)$$

By combining (5.9), (5.11) and choosing a scaling factor of $1/N$, it follows that the SNR (III.8) is equal to

$$SNR = 2^{2B+1}/(N(N+1)) \qquad (5.12)$$

Due to rounding, the output signal mean is zero.

5.2.3) Second order Goertzel algorithm:

The Goertzel algorithm as demonstrated in section 2.3.2 can also be realized in terms of a second order recursive filter. Figure 5.3 shows the equivalent model of a basic cell when the input data are complex and with roundoff error sources.

To calculate the SNR, we again need the output noise variance and the output signal variance. One can notice that the error sources "e_1" and "e_2" intervene N times and "e_3","e_4","e_4","e_5" and "e_6" only once in the computation of σ^2_f. So, a better SNR is expected compared to the first order realization of the Goertzel algorithm. But on the other hand, we foresee overflow problems in branches where the coefficient equals $2\cos\theta$. The multiplicative factor "2" implies an automatic shift of one bit to the left. Therefore, when the result is added to an incoming branch, the odds for an overflow are increased compared to the previous scheme.

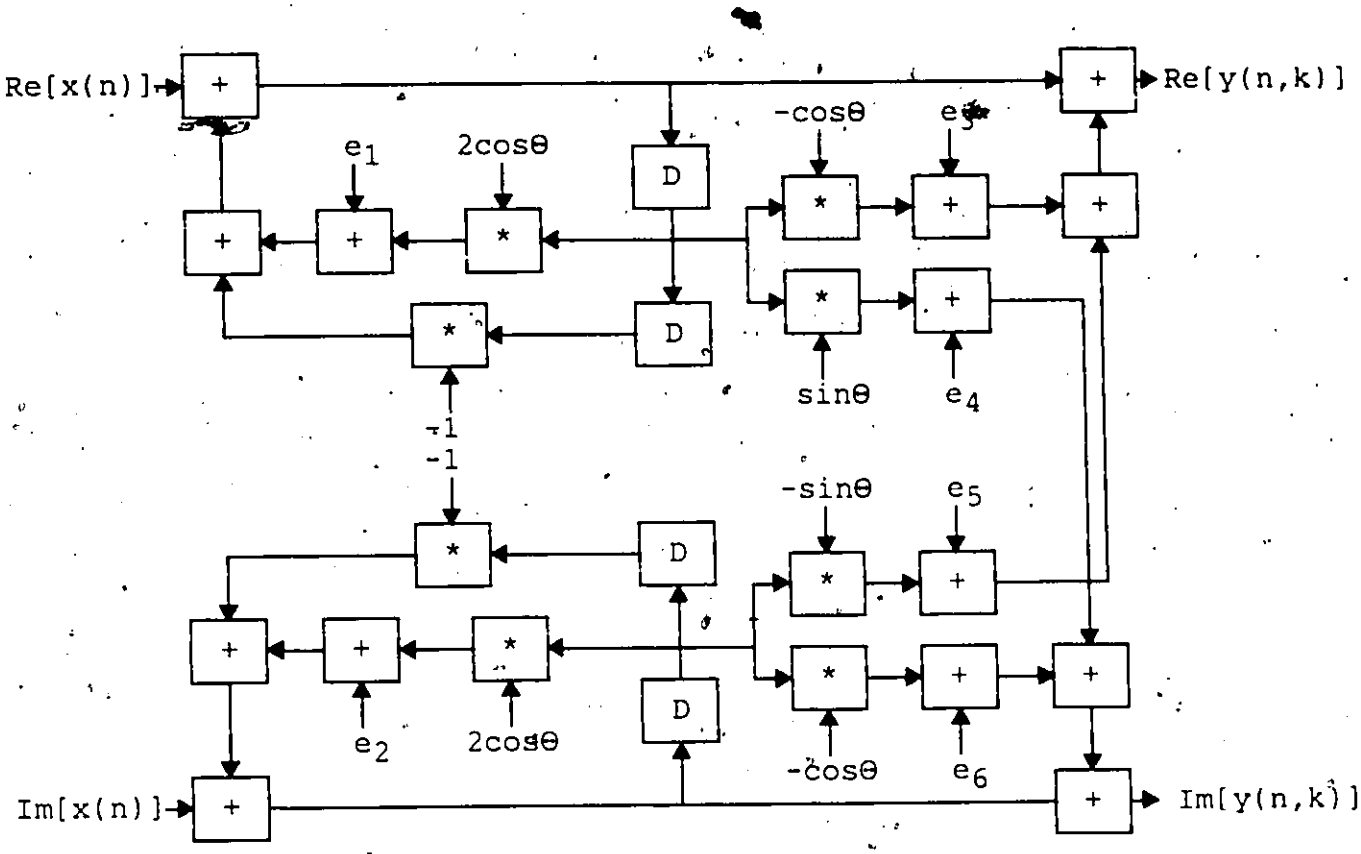


Figure 5.3. Flow graph of a basic cell for the second order realization of the Goertzel algorithm.

The complex unit-sample responses necessary for the error analysis are given by equations (5.3) through (5.8) of the previous section. One can verify these results using Mason's rule.

The output error signal variance is given by

$$\sigma^2_f = (N+2)\, 2^{-2B} /12 \qquad (5.13)$$

To prevent the possibility of an overflow, one must calculate x_{max} using the unit-sample impulse responses at all the critical nodes. Here, there are only two critical nodes. Both are located right after the adder that sums an

incoming input sample with the result of the second order recursive filter. Their unit sample response is given by

$$g(n) = \frac{\sin(\Theta(n+1))}{\sin(\Theta)} \tag{5.14}$$

The corresponding maximum value at these two nodes is

$$x_{max} < \frac{|\sin(\Theta)|}{\sum_{n=0}^{N} |\sin(\Theta(n+1))|} \tag{5.15}$$

For values of Θ around 0 and π, x_{max} tends to

$$x_{max} < 2/(N(N+1)) \tag{5.16}$$

and for Θ around $\pi/2$ and $3\pi/2$,

$$x_{max} < 2/N \tag{5.17}$$

For all the other Θs, a scaling factor between those two extremes is required. The signal-to-noise ratio of this second order configuration can't be compared with the first-order algorithm of the previous section. The scaling factor required at each Θ or "k" varies too much. If one chooses $1/N^2$ as a scaling factor, the SNR will be much lower than that for the first-order system. On the other hand, a scaling factor of 1/N will prompt overflows in the cells computing DFT samples around 0 and N/2. The second-order realization of the Goertzel algorithm is therefore not suited for a fixed point implementation. Floating point arithmetic is thus the only remedy for this scheme.

5.2.4) SNR of DFT systolic arrays:

For fixed point arithmetic with products rounded to B bits and additions performed on B bits numbers (excluding sign bit) yielding results with an accuracy of B bits, the SNR of the following DFT systolic arrays,

1) Two-dimensional systolic array for matrix-matrix multiplication (figure 4.1)

2) One-dimensional systolic array for matrix-vector multiplication (figure 4.2)

3) Kung's one-dimensional pure-systolic array (figure 4.5)

is given by

$$SNR = 2^{2B+1} / N^2 \qquad (5.18)$$

This result is different from the one for the first-order Goertzel algorithm because an FIR section like in figure 5.1 (b) isn't required and therefore the error for that section is not present in the output noise variance.

5.3) Fixed point error analysis of the modified first-order Goertzel algorithm:

The figure below depicts the flow graph for one basic cell. Two systolic arrays implementing the modified first-order Goertzel algorithm were derived in section 4.3, figures 4.7 and 4.9. The major difference with the 1st order realization of the Goertzel algorithm (figure 5.2) is that the multiplicative factor cos(θ) is now found in the feedforward paths instead of in the recursive paths. Using the same method of analysis (Appendix III), the SNR is derived and the results of the overflow analysis are shown below.

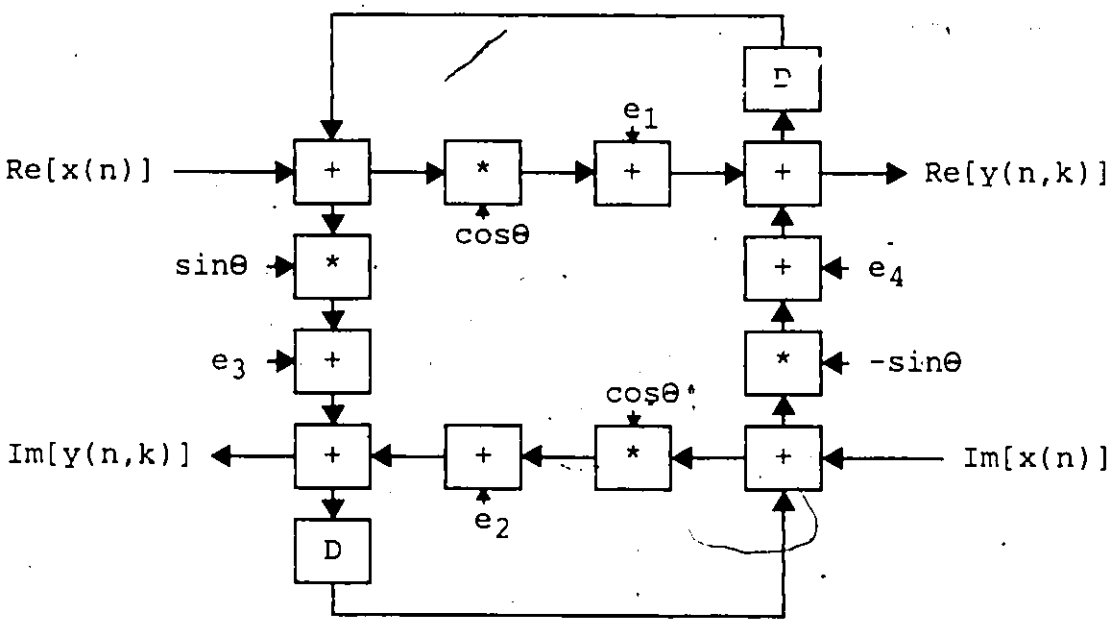


Figure 5.4. Statistical model of a basic cell implementing the modified first-order Goertzel algorithm.

The unit impulse responses required for the analysis are given in the following order: first, the impulse responses from the noise sources towards the outputs Re[y(n,k)], Im[y(n,k)] ("h"'s) and then the impulse responses from the input signals Re[x(n)], Im[x(n)] to the same outputs ("g"'s). Again $\Theta=2\pi k/N$ and, n and k vary between 0 and N-1.

$$Re[h_{e1}(n)]=Re[h_{e4}(n)]=$$
$$=Im[h_{e2}(n)]=Im[h_{e3}(n)]=\cos(\Theta n) \qquad (5.19)$$

$$Re[h_{e2}(n)]=Re[h_{e3}(n)]=-\sin(\Theta n) \qquad (5.20)$$

$$Im[h_{e1}(n)]=Im[h_{e4}(n)]=\sin(\Theta n) \qquad (5.21)$$

$$Re[g_{xr}(n)]=Im[g_{xi}(n)]=$$
$$=\frac{\{\cos(\Theta)\sin(\Theta(n+1))-\sin(\Theta n)\}}{\sin(\Theta)} \qquad (5.22)$$

$$Re[g_{xi}(n)]=-\sin(\Theta(n+1)) \qquad (5.23)$$

$$Im[g_{xr}(n)]=\sin(\Theta(n+1)) \qquad (5.24)$$

For rounding arithmetic, the mean value of the output error signal is equal to zero and the output error signal variance is given by

$$\sigma^2_f = 2^{-2B} N/6 \qquad (5.25)$$

Again, (5.25) applies for both the real and imaginary parts of the complex output signal.

The scaling factor that is applied to the input signal is evaluated by the following expression,

$$x_{max} < \frac{1}{\sum_{n=0}^{N-1} |Re[g_{xr}(n)]| + |Re[g_{xi}(n)]|} \qquad (5.26)$$

By making use of L'Hospital's rule, one finds for θ around 0 and π, a scaling factor of 1/N and for θ around $\pi/2$ and $3\pi/2$, 1/2N. In between, the proper scaling factor varies between 1/2N and 1/N. As mentioned in section 5.2.2, these bounds are very conservative. To calculate σ^2_x, 1/N is chosen.

Using (III.7) and the results above, the output signal variance is given by

$$\sigma^2_y = \sigma^2_x \sum_{n=0}^{N-1} [(Re[g_{xr}(n)])^2 + (Re[g_{xi}(n)])^2] \qquad (5.27)$$

The mean of the output signal is zero. Appendix IV gives the result for the summation in the right hand side of (5.27).

$$\sigma^2_y = (1/3N^2)(N) = 1/3N \qquad (5.28)$$

By combining (5.25) and (5.28), it follows that the SNR equals

$$SNR = 2^{2B+1}/N^2 \qquad (5.29)$$

In the next section, we compare the systolic arrays on the basis of the signal-to-noise ratio. Both predicted and simulation results are presented.

5.4) Comparison:

5.4.1) Generalities:

This section presents a comparative study of the algorithms analysed in this chapter along with other DFT computational algorithms. First, a theoretical comparison is presented for the 1st order realization of the Goertzel algorithm, for the modified first-order Goertzel algorithm and for an FFT algorithm [14]. They are compared for different sequence and word lengths. Subsequently, a simulation on a digital computer of two systolic arrays the first one uses Kung's proposed structure (figure 4.5) and the second, the modified Goertzel algorithm (figures 4.7 and 4.9). Their performance is evaluated for two different input signals: white noise and sinusoidal waveforms.

This section ends with a study, using a simulation, of the error introduced by quantizing the coefficients of a discrete Fourier transform based on the modified first order Goertzel algorithm. In fact, the stored product ROM technique is compared to the use of hardware multipliers.

5.4.2) Theoretical results:

The equations derived for the SNR of the algorithms to be compared are summarized below.

Method #1: 1st order Goertzel algorithm (5.12),

$$SNR = 2^{2B+1}/N(N+1) \qquad (5.30)$$

Method #2: Modified 1st order Goertzel algorithm and DFT systolic arrays (5.29),

$$SNR = 2^{2B+1}/N^2 \qquad (5.31)$$

Method #3: FFT with step-by-step scaling [14],

$$SNR = 2^{2B-2}/N \qquad (5.32)$$

The theoretical results are presented graphically on figure 5.5 through 5.9. On these graphs the accuracy symbolized by B includes the sign bit. The first order Goertzel algorithm and the modified first-order Goertzel algorithm as seen on the graphs give about the same theoretical results. The first one is a fraction of a dB less accurate than the latter. The difference comes from the error introduced in the FIR filter found in the first-order Goertzel algorithm (figure 5.1). An FFT-type computation still yields the highest signal-to-noise ratios. But as it was discussed in sections 3.3 and 4.2.8, a parallel FFT architecture isn't cost-effective for a VLSI implementation. Extending the register's length is the only way to improve the SNR of the DFT systolic arrays mentioned in chapter 4 and 5 to the level of an FFT algorithm.

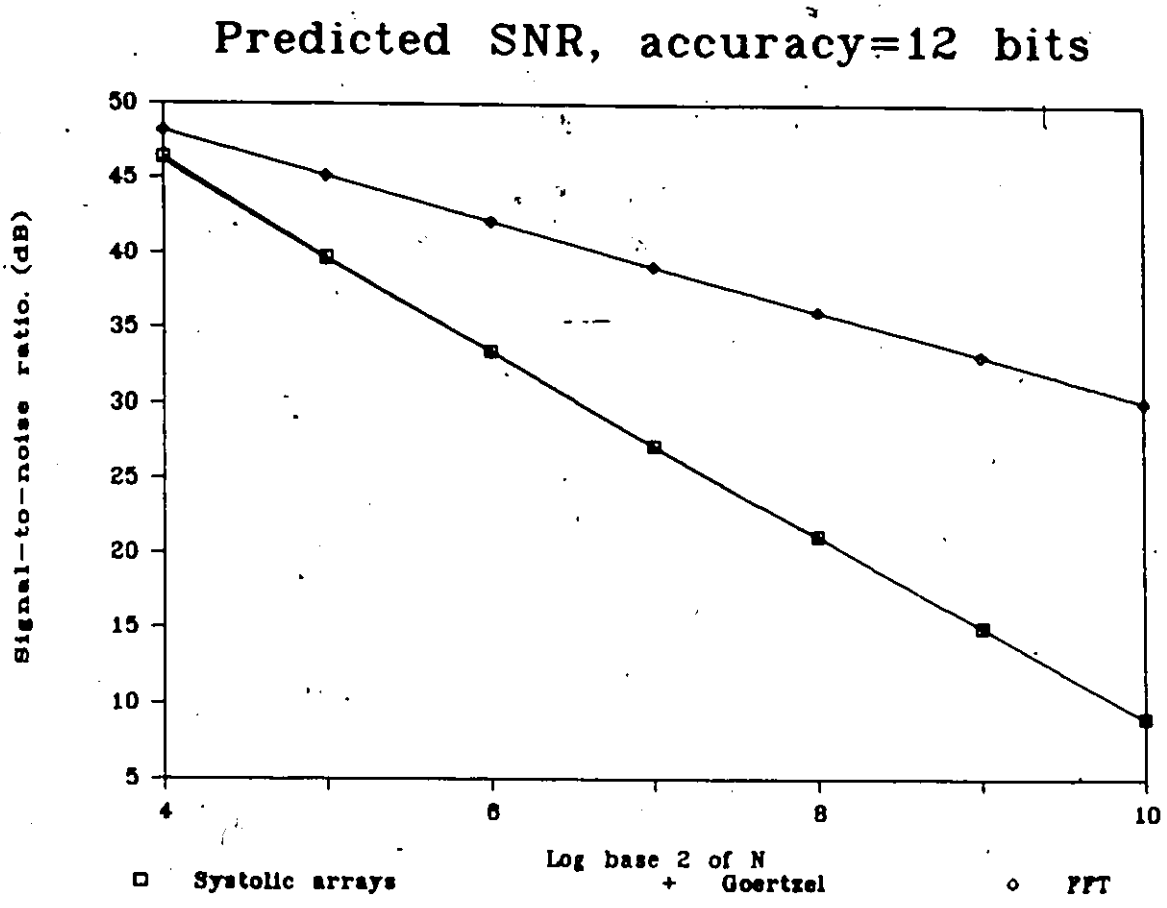


Figure 5.5 Predicted SNR vs N for all three methods when the computations are performed with an accuracy of 12 bits.

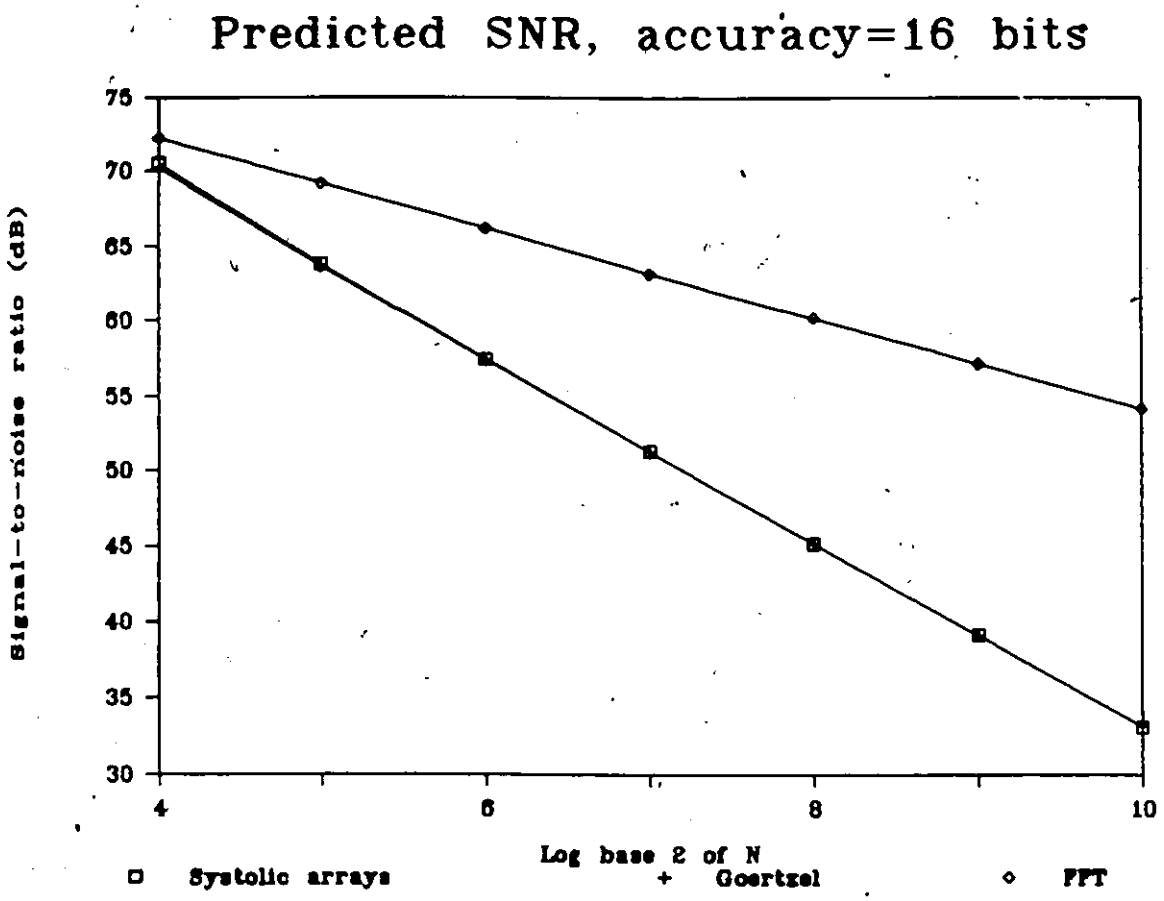


Figure 5.6 Predicted SNR vs N for all three methods when the computations are performed with an accuracy of 16 bits.

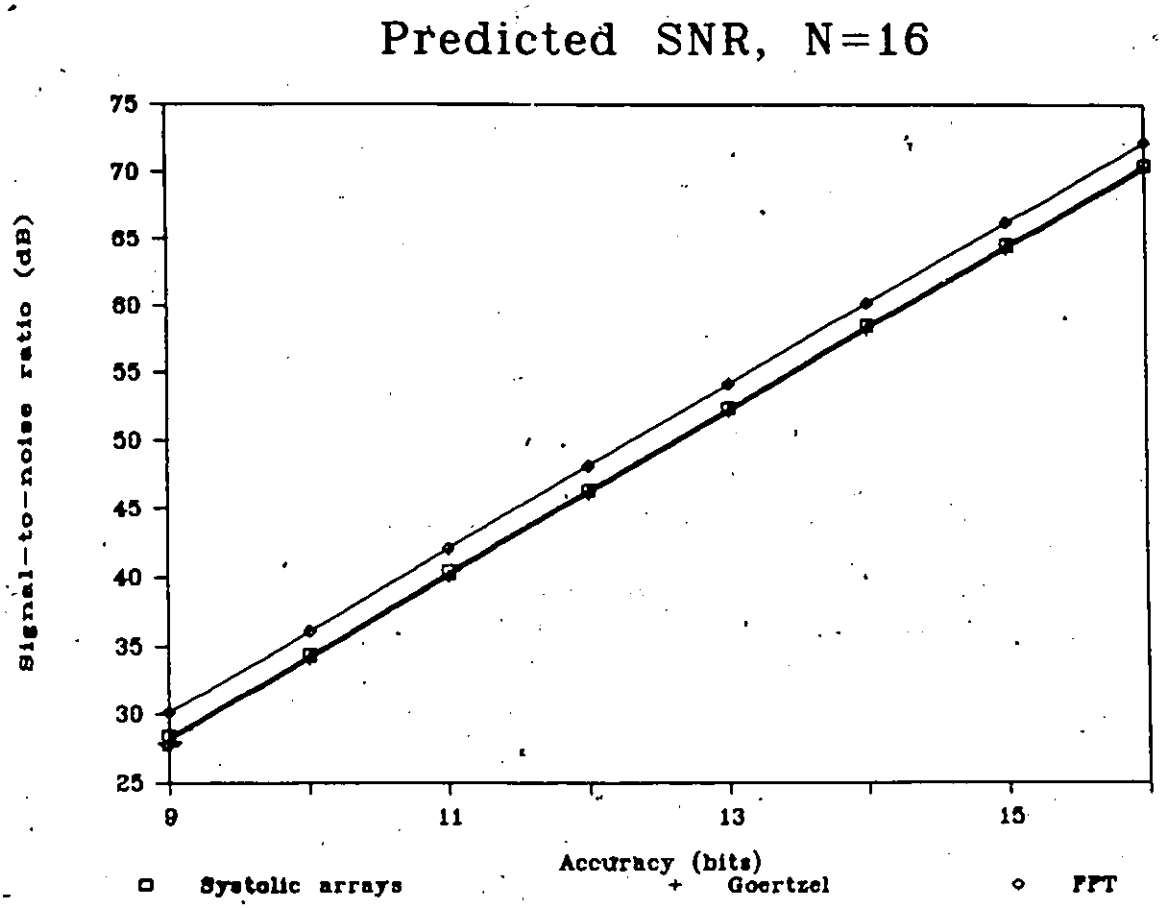


Figure 5.7 Predicted SNR vs Accuracy for all three methods when N=16.

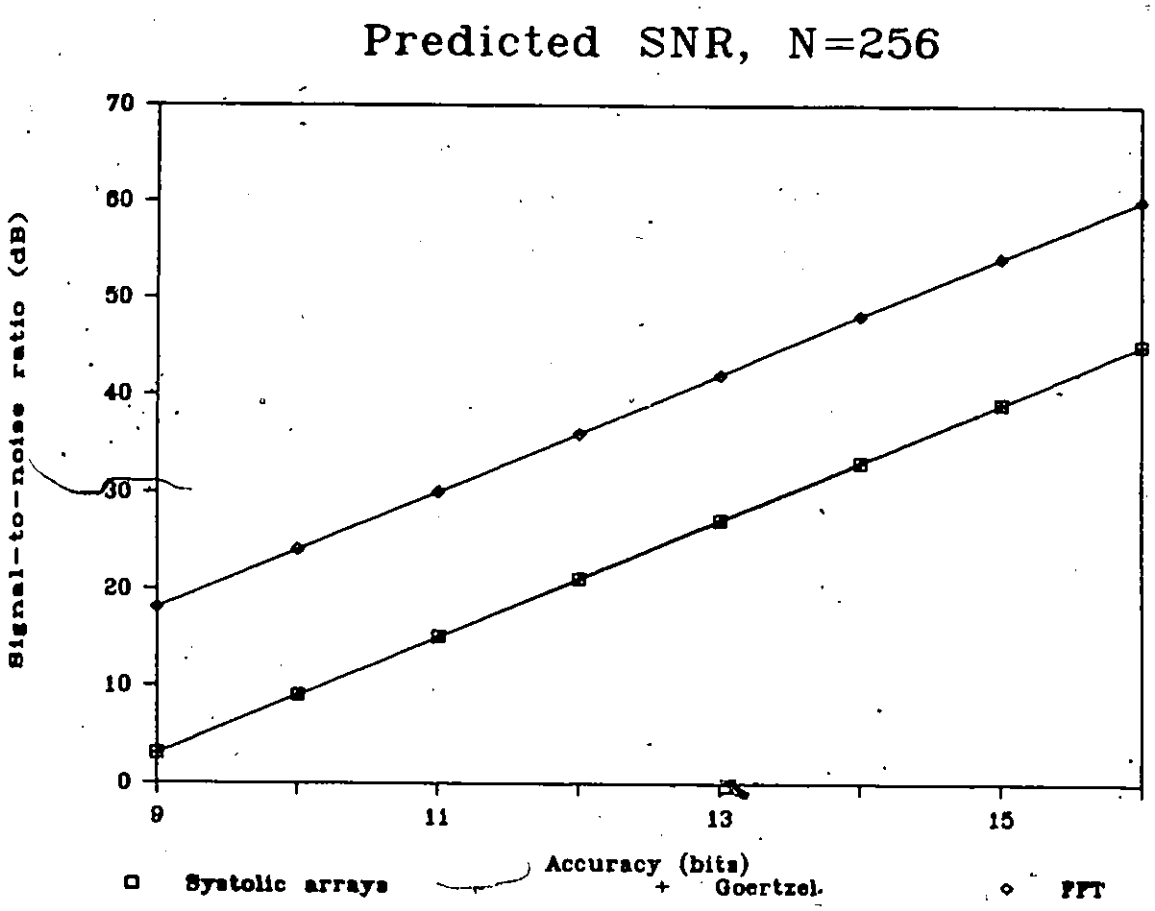


Figure 5.8 Predicted SNR vs Accuracy for all three methods when N=256.

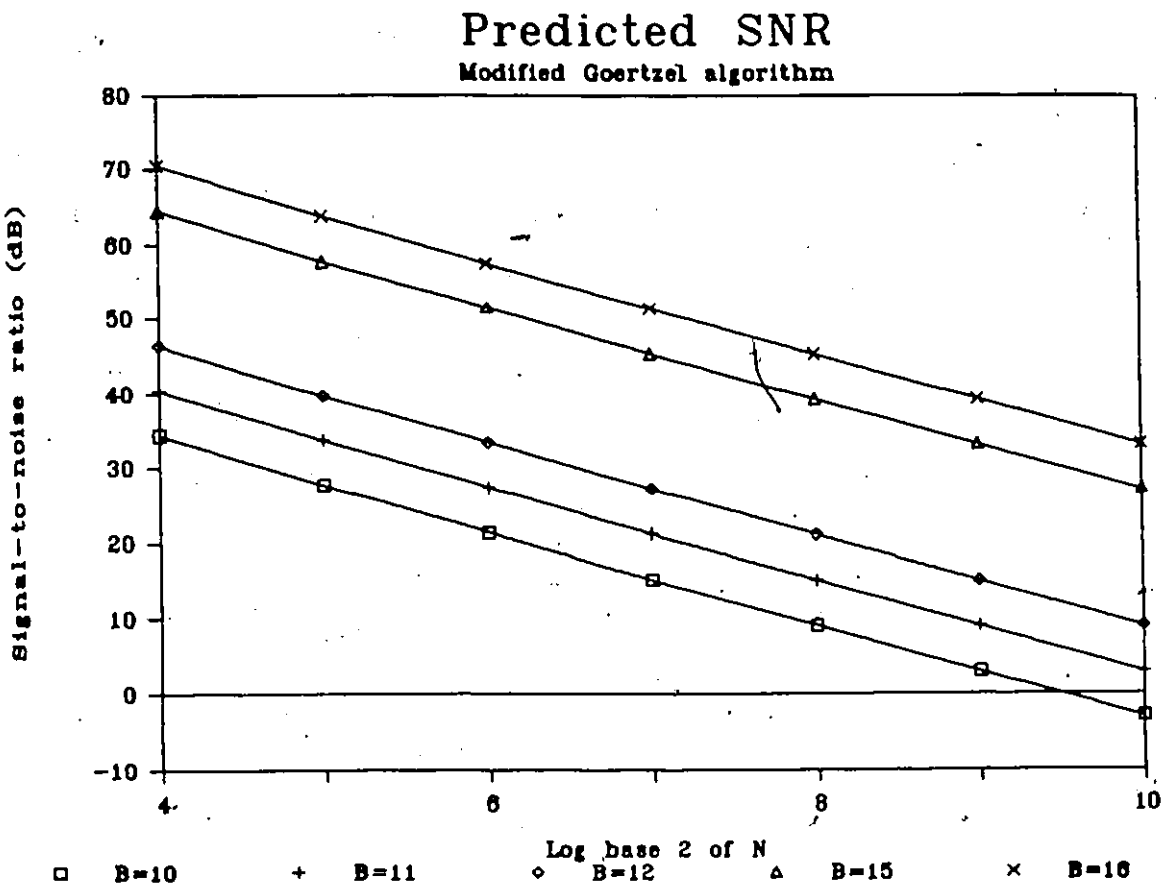


Figure 5.9 Predicted SNR vs N for the modified Goertzel algorithm with different computational accuracies, B (includes sign bit).

5.4.3) Simulation results:

The simulation is intended to verify the validity of the theoretical model. The simulation procedure follows the flow chart of figure III.3 (Appendix III). To improve the simulation speed, the number of samples is chosen to be a power of two. This permits the use of the FFT algorithm to compute the unquantized version of the DFT.

5.4.3.1) White noise input:

The computation of the DFT using a prescribed method is performed on a sequence of zero-mean random numbers which are quantized to B_1 bits. The random numbers are spread between -1 and +1 according to a uniform distribution. The result is allowed to expand up to B bits in two's complement (where B is greater than B_1). The DFT is computed 100 times and an average signal-to-noise ratio in dB is calculated. The restriction on the number of samples is governed by the equation below.

$$\text{Log}_2 N \leq B - B_1 \qquad (5.33)$$

In (5.33) B_1 is set to eight and B to a maximum of sixteen bits. Therefore N can not exceed 256 samples.

Figure 5.10 through 5.13 present the comparison between the simulated and the predicted SNRs for Kung's pure-systolic array (figure 4.5) and the systolic array computing

the DFT using the modified first-order Goertzel algorithm (figures 4.7 and 4.9).

These graphs show that the theoretical analysis describes adequately the behavior of the algorithms when simulated, and this for N equal to 16 up to 256. On these graphs the theoretical results were adjusted so that they account for the fact that for N a power of 2, no error is introduced at the following cells: 1, N/4, N/2, 3N/4. From both analysis techniques, one can conclude that systolic arrays implementing the modified first-order Goertzel algorithm are as good as Kung's pure-systolic array.

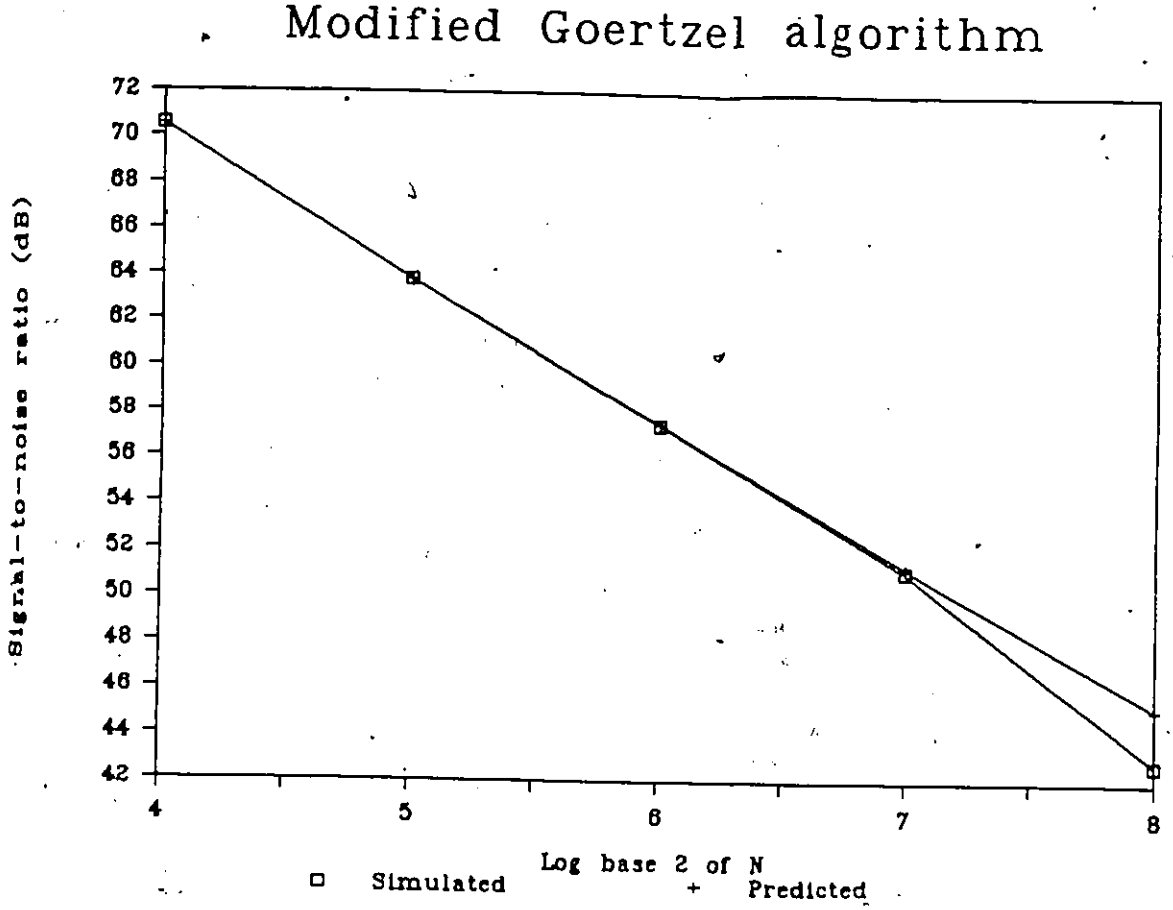


Figure 5.10 Simulated vs predicted SNR as a function of the sequence length for the modified Goertzel algorithm when the accuracy is set to 16 bits.

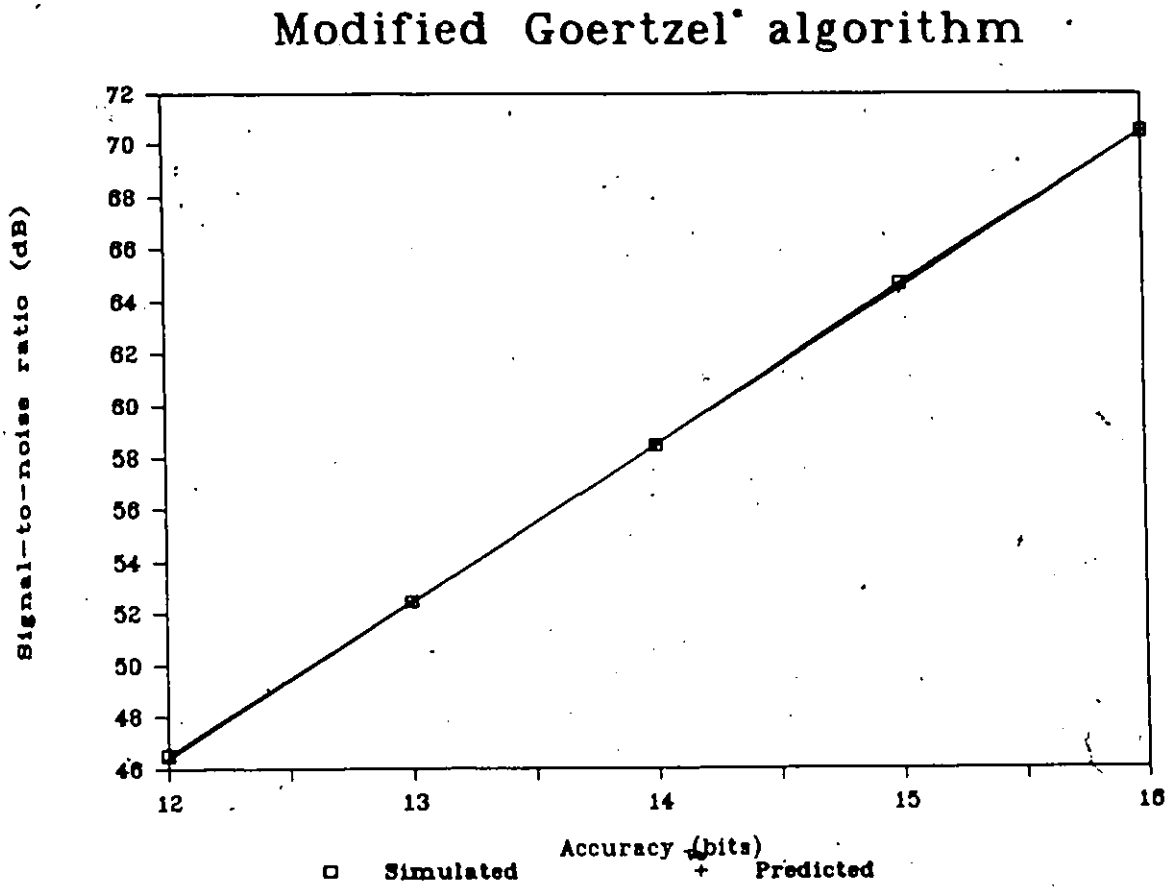


Figure 5.11 Simulated vs predicted SNR as a function of the accuracy for the modified Goertzel algorithm when N=16.

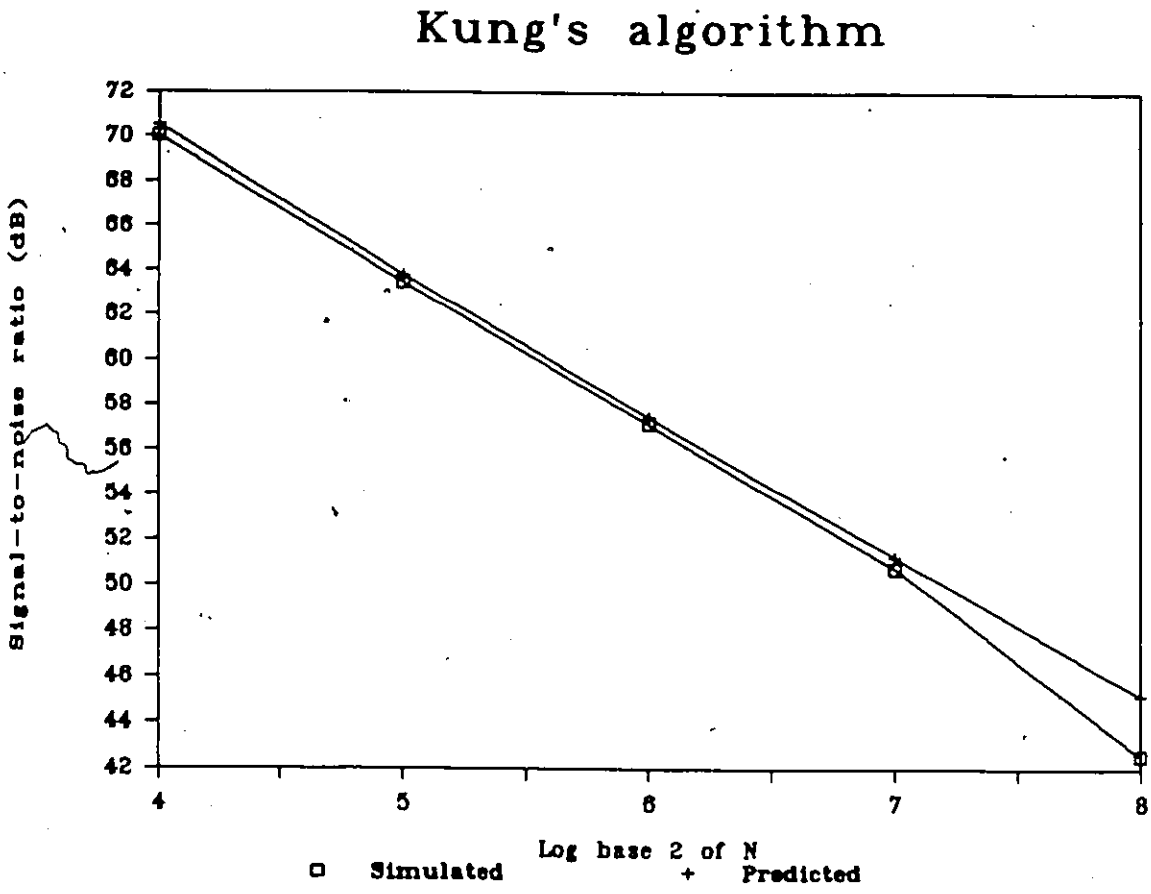


Figure 5.12 Simulated vs predicted SNR as a function of the sequence length for Kung's systolic array [51] when the accuracy is set to 16 bits.

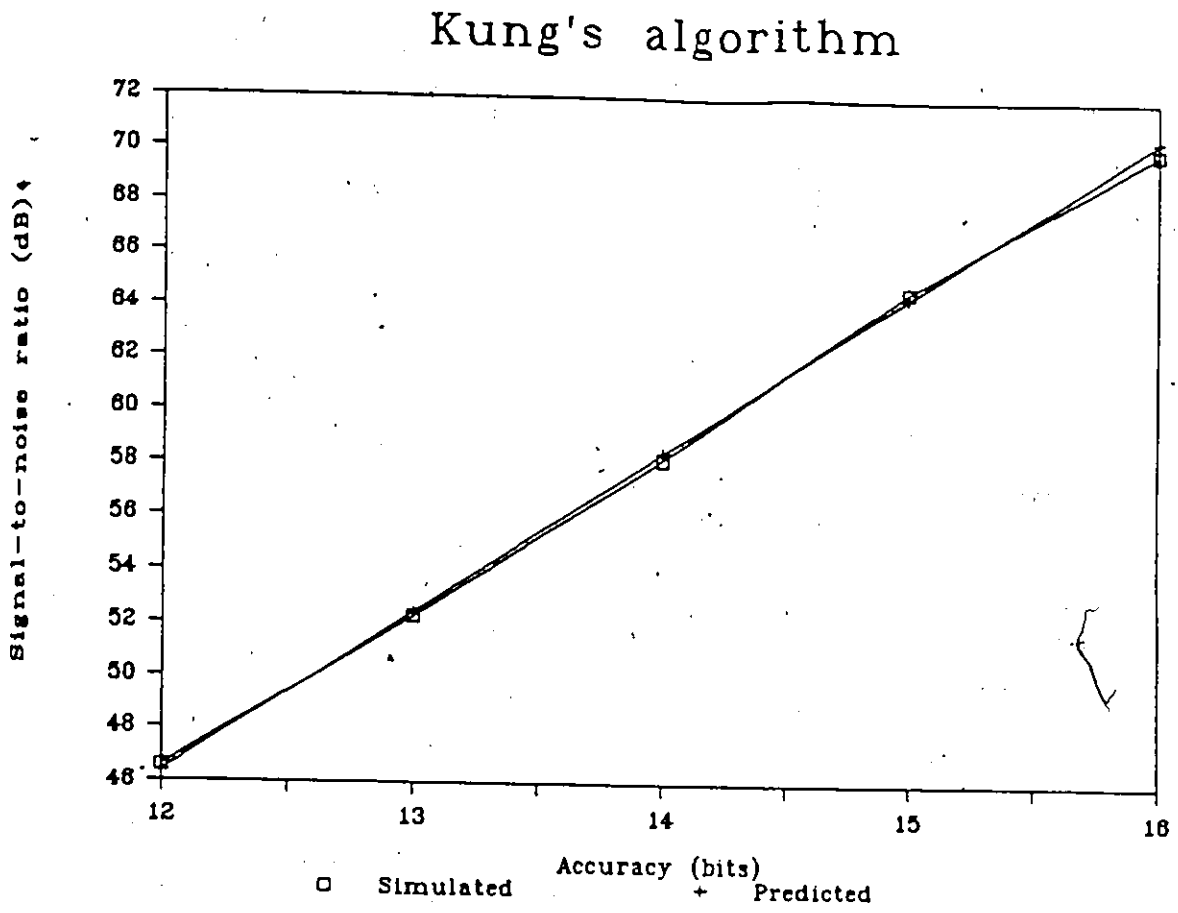


Figure 5.13 Simulated vs predicted SNR as a function of the accuracy for Kung's systolic array [51] when N=16.

5.4.3.2) Sinusoidal input:

Due to the deterministic nature of a sine wave, the correlation between the signal and the roundoff error is higher than that for a white noise input. This implies that a theoretical error analysis won't give accurate results. Therefore, simulations of the systolic arrays on a digital computer, for this new input signal, should provide results with more accuracy than achievable with a theoretical analysis. Here, the performance of systolic arrays based on the modified Goertzel algorithm is reported. Table 5.1 gives the results. The scaling factor was increased to 2/N to avoid the risk of overflow.

Table 5.1 Simulated SNR for the modified Goertzel algorithm with sinusoidal inputs and an accuracy of 16 bits.

N	SNR (dB)
16	83.5
32	77.0
64	71.7

5.4.4) Effect of coefficient quantization in the modified first-order Goertzel algorithm:

So far, the coefficients $\cos\theta$ and $\sin\theta$ were assumed to have infinite precision. In a practical situation, this is often not the case.

It was pointed out in section 4.4 that two methods for implementing a multiplication can be considered. The first one is to use a hardware multiplier where both the input sample and the coefficient have to be quantized to a given wordlength. And, the second is the stored product ROM technique. Here, only the input sample is quantized. All the product results are computed off-line on a digital computer with double-precision arithmetic and stored in a ROM.

Weinstein [60] has analysed the effects of coefficient quantization using a statistical model based on quantization error added to each coefficient. The derivation is rather tedious and the results give only a rough estimate. Instead, a simulation evaluates the signal-to-noise ratio versus the accuracy for different coefficient (twiddle factor) wordlengths. The simulation results are given in figures 5.14-15. Each figure contains three curves representing three quantization situations: one for stored products (double-precision coefficients of 64 bits), a second for 16 bits and a third for 12 bits (including the sign bit).

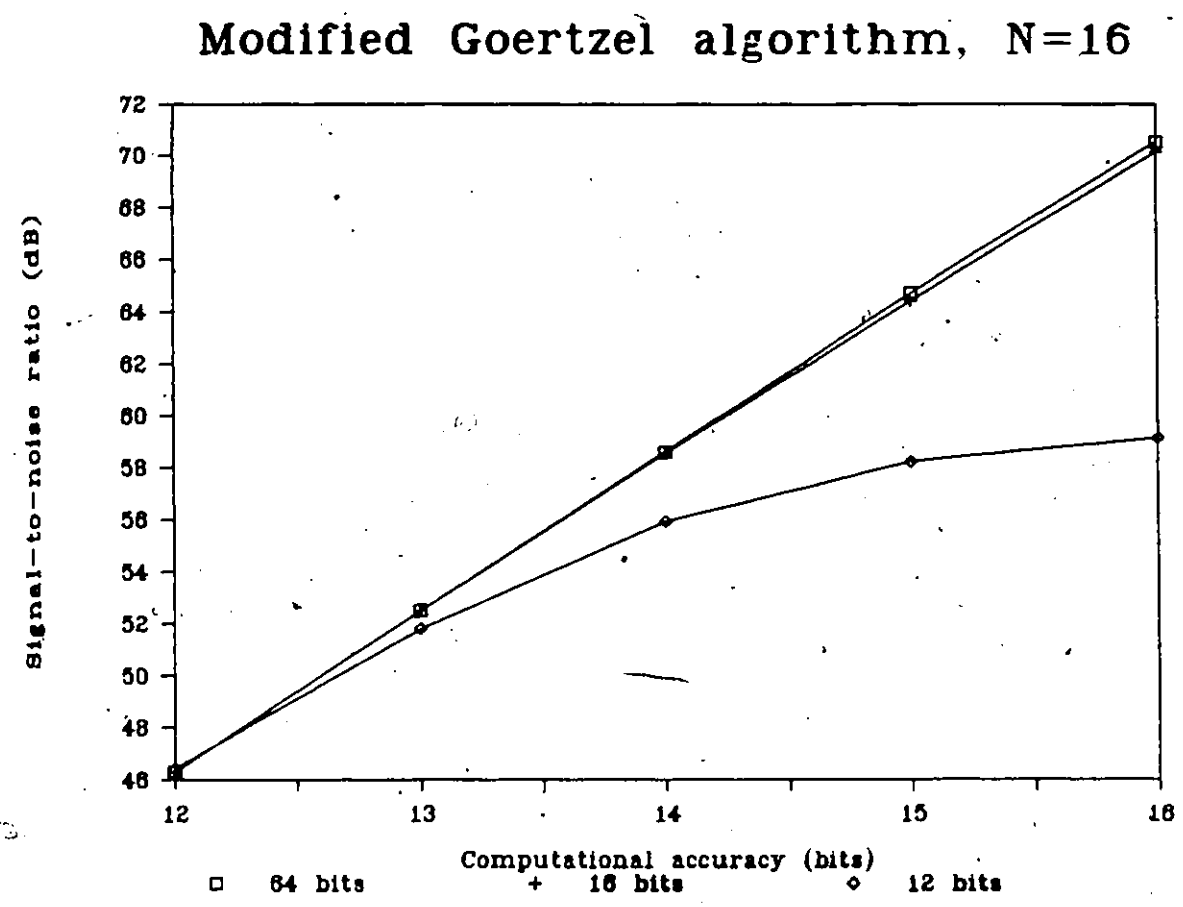


Figure 5.14 Coefficient quantization effects in a systolic array based on the modified Goertzel algorithm for N=16.

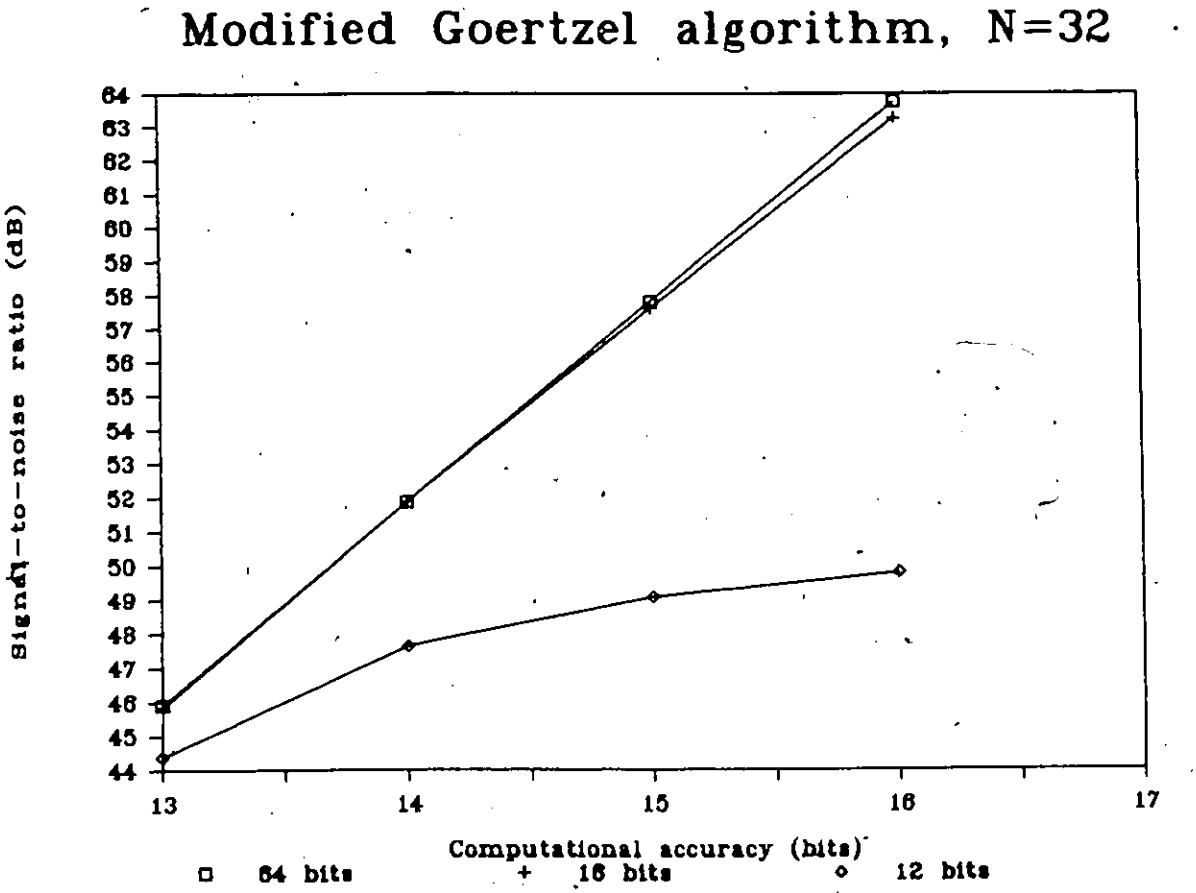


Figure 5.15 Coefficient quantization effects in a systolic array based on the modified Goertzel algorithm for N=32.

From these graphs, it can be concluded that to get a SNR within half a dB from the 64-bit case the coefficients should be rounded at least to the same wordlength used in the computations. This condition is met by both hardware multiplier and stored product ROM implementations.

But with stored product ROM, an improvement of one bit is achieved therefore yielding better signal-to-noise ratios than those found in hardware multiplier implementations. This one bit improvement must not be confused with an increase of the SNR by 6 dB as found in extending the registers wordlength by one bit during computations. It is merely the distance in bits of the break away point on a curve (figure 5.14 or 5.15) from a given computational accuracy; here 16 bits. On these figures, the break away point is located at 15 bits accuracy. The difference gives one bit (16 bits-15bits=1bit). The stored product technique has proven its use in a FFT system reported in [33].

5.5) Discussion:

In this chapter, the effects of using finite precision arithmetic in the implementation of DFT systolic arrays have been analysed. For its simplicity and cost-effectiveness, fixed point arithmetic implementations have been considered.

The performance of the DFT systolic arrays seen in chapter 4 are assessed by means of an error analysis of the first and second order Goertzel algorithm. This algorithm has been chosen owing to the fact that it embraces most of the existing implementations of a DFT as a systolic array.

It has been found that the first order Goertzel algorithm is more attractive that its second order counterpart for fixed point implementations. Overflow prevention being the overriding factor in this case, the second order algorithm requires a scaling factor proportional to $1/N^2$. Such a scaling factor implies a very strong attenuation of the input signal. Instead, the first order algorithm needs only a scaling factor proportional to $1/N$.

Subsequently, the modified first order Goertzel algorithm has been analysed. It has been found that the SNR of the proposed DFT algorithm compares favorably with the one of the first order Goertzel algorithm and that the scaling is also proportional to $1/N$.

When compared to an FFT algorithm, all the DFT systolic arrays discussed in this work demonstrate a lower SNR. The difference is emphasized especially for sequence lengths larger than 128 samples. This particularity comes from the fact that the SNR is directly linked to the total number of multiplications performed either sequentially or in parallel. DFT systolic arrays are built around algorithms which require a total of multiplications proportional to N^2; FFT algorithms , instead, need $O(N\text{Log}_2N)$ as was seen in chapter 2.

The advantages of the modified first-order Goertzel algorithm over the other method are more obvious when hardware implementation and real-time signal processing are both considered (see chapter 4).

A simulation on a digital computer proved the validity of the theoretical model for inputs with uniform probability densities. Kung's pure-systolic array has been compared to a systolic array based on the modified first order Goertzel algorithm. Both arrays showed similar signal-to-noise ratios as the analysis predicted. No analysis has been carried out for sinusoidal inputs, the correlation between the input signal samples and the error signals being too high to yield accurate results, a simulation has been used instead.

Then, in another simulation, the effect of

coefficient quantization has been addressed. It has been found that the stored product technique yields an equivalent improvement of one bit over designs based on standard hardware multipliers.

Due to its additional complexity, floating point arithmetic implementations have not been analysed. But if a system designer is ready to pay for this extra complexity, better signal-to-noise ratios can be expected. The scaling factor required in the fixed point realization becomes superfluous. Overflows are virtually avoided by the use of an exponent attached to a mantissa. Therefore the SNR of any systolic array becomes proportional to $1/N$ instead of $1/N^2$ for the fixed point case.

CHAPTER VI

CONCLUSION

VLSI technology has created new opportunities for system integration of digital signal processing algorithms. It promises low-cost, high-density and fast special-purpose devices able to perform tasks like linear filtering, convolutions, correlations and discrete Fourier transforms using parallel architectures. Such a promising future for VLSI devices comes also with totally new challenges for system designers. As the device technology is scaled down to sub-micron level and the chip size is increased, the interconnection problems become a major issue.

These new architectural requirements have therefore entailed a major review of the criteria for evaluating digital signal processing algorithms. The efficiency of an algorithm is based more on the degree of complexity of the communication required between the processing elements rather than on the actual number of arithmetic operations required.

It has been seen in this thesis that parallel architectures in which simple processing elements communicate in a regular fashion with their nearest neighbors will gain the most from VLSI. The systolic array concept has been presented as a simple means for mapping a signal processing algorithm onto an architecture amenable to VLSI implementation.

The computation of the discrete Fourier transform being the algorithm under study in this research work, a survey of available computational algorithms has been presented. Linear filtering and fast Fourier-like algorithms have been described. A comparison of the algorithms based on criteria such as the type of arithmetic operations involved and data I/O requirements has identified two linear filtering algorithms which can be easily mapped onto VLSI parallel architectures. These algorithms are the direct calculation of the DFT and the Goertzel algorithm. FFT-like algorithms have to be discarded because of their need for extensive internal and external communications.

Subsequently, existing systolic arrays for the computation of the DFT have been surveyed. Most of these are based on one of the algorithm mentioned above or on an adaptation to ease their mapping onto systolic arrays. A comparison of these proposed systolic arrays for the DFT has shown that (i) the one-dimensional systolic array for

matrix-vector multiplication [49] and (ii) Kung's one-dimensional systolic array [51] cannot be used in applications where the DFT of a continuous flow of vectors is required. Others like (i) the two-dimensional systolic array for matrix-matrix multiplication [49], (ii) one-dimensional systolic array using pipelined data inputs [39] and (iii) Kung-Leiserson's one-dimensional systolic array [53] don't make use of the processing elements in an efficient way.

Therefore, two new systolic arrays have been introduced for the calculation of the DFT. The first one is a semi-systolic array where input data samples enter sequentially by one end and results appear shifted in time at the output of each processing element. These results are collected by a bus or a tree-like network. The second one is a pure-systolic array in which input data samples again enter sequentially and results exit sequentially. Both architectures achieve a data throughput of one complex sample per clock cycle. This throughput rate is attained by pipelining the input data samples through a network of N locally interconnected identical cells. Here, N is the length of the sequence to be transformed and can be any integer number. Because the twiddle factors are static in the array along with partial results, inter-cell communications are kept to a minimum. Only the input data

samples with usually smaller wordlength are allowed to travel between cells.

Each cell is built around four adders and four multipliers. These figures represent a minimum of arithmetic elements for this class of DFT systolic arrays. Registers are also required. Two approaches have been proposed to implement the multiplications in a basic cell. The first one uses hardware multipliers and require a loading phase for the twiddle factors before starting the computations. The second method is based on stored product ROM. In this case, all possible products are preloaded in a read-only-memory (ROM).

The two systolic arrays have been made possible by the use of a new linear filtering algorithm based on a modification of the Goertzel algorithm. This new DFT algorithm avoids the external phase correction term required at the output of each cell usually associated with the first order Goertzel algorithm. The new algorithm has been called the modified first order Goertzel algorithm.

We then proceeded with a fixed point error analysis of the first and second order Goertzel algorithm. This algorithm has been chosen as a basic algorithm underlying most of the proposed DFT systolic arrays. It has been found that the second order algorithm is more sensitive to overflows than the first order algorithm. Therefore, the

first order algorithm is favored over the other algorithm for fixed point implementations. Following this, an error analysis for the modified first order Goertzel algorithm has been carried out. This algorithm demonstrated slightly better results in terms of signal-to-noise ratio than the first order Goertzel algorithm.

A simulation has verified the predicted behavior of the systolic arrays based on the algorithms under study. It has been found that, in general, the systolic arrays give approximately the same SNRs for fixed point arithmetic. Another simulation has given some insights into the effects of coefficient quantization. It has been established that a systolic array based on stored product ROMs yields an equivalent improvement of one bit over a systolic array based on conventional hardware multipliers.

It has also been ascertained that to get signal-to-noise ratios approaching those found in FFT-based systems longer wordlengths must be used during the computations. Adding one bit to the registers improves the SNR by six dB. This solution could bring some severe I/O limitations especially if partial results and coefficients must travel between processing elements (cells). This problem is experienced in most of the systolic arrays surveyed in chapter four. In a systolic array based on the modified Goertzel algorithm, increasing the wordlength during the

computations doesn't create any I/O overheads. Partial results and twiddle factors are static in the network.

Though, this research work has been concerned only with the mapping of the discrete Fourier transform onto VLSI systolic arrays, many more digital signal processing algorithms can benefit from similar mappings. VLSI technology has started a new era in the design of special-purpose parallel architectures and also parallel computer architectures. What one can read in today's current literature on the subject is probably only the tip of the iceberg.

APPENDIX I

Hardware multipliers

Fixed point multipliers

Company	Part number	Size (bits)	Delay (ns)	Power diss.(mW)	S-P (pJ)	Technology	Price 100's
TRW	MPY016K	16*16	45	4000	180	TTL	$125ea
Analog Devices	ADSP1081	8*8	90	75	6.75	CMOS	N/A
	1012	12*12	130	125	16.25	CMOS	
	1016	16*16	170	125	21.25	CMOS	
	1024	24*24	235	125	29.38	CMOS	
TI	THCT1010	16*16	100	150	15	CMOS	N/A
Integrated Device Tech.	IDT 7210	16*16	65	250	16.25	CEMOSII	$210ea
	7243	16*16	65	250	16.25	CEMOSII tm	$210ea
Logic Devices	LMU 12PC	12*12	65	100	6.55	CMOS	$61.65
	16PC	16*16	80	125	10	CMOS	$61.65
Micro Power Systems	MP 1010	16*16	75	125	9.38	CMOS	$105ea

Floating point units

Company	Part number	Size (bits)	Megaflops	Power diss.(W)	Technology	Price 100's
Weitek	1064/65	64	16	2.0	n-MOS	$1100
	1032/33	32	5	1.5	n-MOS	N/A

N/A : not available

APPENDIX II

ROMs

Fixed point multipliers using stored product ROMs

Company	Part number	Word length	Access time ns	Power diss.(mW)	S-P (pJ)	Technology	Price 100's
Fujitsu	MB 7134 3-4K*4	12	50	900	45	TTL	N/A
Signetics	82HS195 3-4K*4	12	40 35 25	2175 2175 2175	87 76 54	TTL TTL TTL	$60 ea $60 ea $86 ea
Harris	HM6646 3-4K*4	12	200	N/A	--	CMOS	N/A
Signetics	23256A15 2-32K*8	15	150	1000	150	CMOS	$28 ea
Intel	27256 2-32K*8	15	250	1000	250	HMOSIIE	$28 ea
Toshiba	TMM23256 2-32K*8	15	150	400	60	nMOS	N/A
Solid State Scientif.	SCM23256 2-32K*8	15	100 120 150	150 150 150	15 18 22.5	CMOS CMOS CMOS	N/A
Signetics	23256A20 2-64K*8	16	200	1150	230	CMOS	N/A
Mostek MPX I/O	MK3901M 1-64K*16	16	120	N/A	--	CMOS	$76 in 1000's
Advanced Micro Devices	Amd27C-1024 1-64K*16	16	170	250	42.5	CMOS	$400ea 03/86 only

MPX : multiplex

APPENDIX III

The SNR of a LSI system for fixed point arithmetic:

For fixed point arithmetic, roundoff errors are introduced only by multiplication. If we consider fractional numbers, the product of two B-bit numbers (excluding sign bit) gives a result less than one but with an accuracy of 2B bits. Therefore rounding or truncation to B bits becomes necessary when no extra bits for increased accuracy are provided. Overflows can cause large errors in a digital system. They are avoided by properly scaling the input signal.

The method of analysis used [14] treats the effect of rounding as an additive noise signal, as seen in figure III.1. Here, w(n) is the quantized output signal, y(n) the exact output and e(n), the error introduced by rounding the result of the multiplication.

For the analysis, the following assumptions are considered

1. The error sequence e(n) is a white-noise sequence,
2. the error sequence has a uniform distribution over one quantization interval,
3. the error sequence is uncorrelated with the input signal x(n).

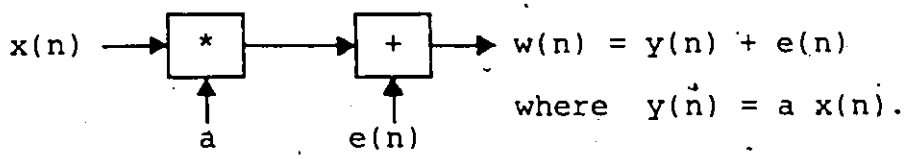
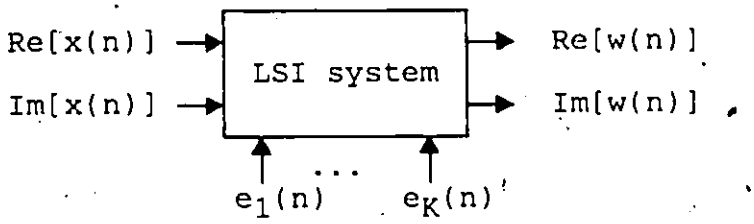


Figure III.1. Representation of the roundoff error introduced in multiplication.

When we deal with linear shift invariant systems , the errors due to rounding for several multipliers are considered separately and their effects superimposed at the output. An LSI system adapted to the DFT systolic arrays seen in chapter 4 is given in figure III.2. This figure shows the system for complex input data, and with "K" error sources.



$$\text{where } Re[w(n)] = Re[y(n)] + \sum_{k=1}^{K} Re[f_{ek}(n)]$$

$$\text{and } Im[w(n)] = Im[y(n)] + \sum_{k=1}^{K} Im[f_{ek}(n)]$$

Figure III.2. Statistical model for fixed-point roundoff noise in an LSI system.

In figure III.2, Re[y(n)] and Im[y(n)] denote the output that would be obtained due to Re[x(n)] and Im[x(n)] if there were no quantization error and $Re[f_{ek}(n)]$ and $Im[f_{ek}(n)]$ represent the output errors due to the noise

sources $e_k(n)$. Using the assumptions stated before, an error signal e(n) has, in the case of rounding, a mean equal to zero and a variance (σ^2_e) equal to (1/12) 2^{-2B}.

The mean and the variance of the output noise signal can be expressed as

$$m_f = \sum_{k=1}^{K} m_{ek} \sum_{n=-\infty}^{\infty} h_{ek}(n) \qquad \text{(III.1)}$$

$$\sigma^2_f = \sum_{k=1}^{K} \sigma^2_{ek} \sum_{n=-\infty}^{\infty} h^2_{ek}(n) \qquad \text{(III.2)}$$

where m_{ek} and σ^2_{ek} are respectively the mean and variance of the k^{th} noise source and $h_{ek}(n)$ the unit-sample response from a noise source input $e_k(n)$ to the output of the system.

In order to compute the SNR, we need to calculate the output signal variance σ^2_y. This quantity is related to the input signal variance which in turn determines the possibility of overflow at any node of the network. Therefore we must be careful in choosing the maximum input signal. Letting x(n) denote the system input, either real or imaginary and $v_l(n)$ and $h_{lm}(n)$ denote respectively the output at the l^{th} node in the system and unit-sample response from a given input to that particular node "l", then

$$v_l(n) = \sum_{m=1}^{M} \sum_{r=-\infty}^{\infty} h_{lm}(n)\, x(n-r) \qquad \text{(III.3)}$$

Assuming that x(n-r) does not exceed x_{max} in absolute value, we can rewrite (III.3) as

$$|v_l(n)| < x_{max} \sum_{m=1}^{M} \sum_{r=-\infty}^{\infty} |h_{lm}(r)| \qquad \text{(III.4)}$$

With computations performed using fractional arithmetic, we require that $|v_l(n)| < 1$, and to meet (III.4) we require,

$$x_{max} < \frac{1}{\sum_{m=1}^{M} \sum_{r=-\infty}^{\infty} |h_{lm}(r)|} \qquad \text{(III.5)}$$

for all the nodes in the network. (III.5) represents an upper bound on the maximum value of the inputs so that no overflow occurs at any particular node "l".

We are now able to compute the mean and variance of the exact output signals i.e. Re[y(n)] and Im[y(n)]. The equations below give the results for Re[y(n)] but apply also to Im[y(n)].

$$m_y = m_{xr} \sum_{n=-\infty}^{\infty} Re[g_{xr}(n)] + m_{xi} \sum_{n=-\infty}^{\infty} Re[g_{xi}(n)] \qquad \text{(III.6)}$$

$$\sigma^2_y = \sigma^2_{xr} \sum_{n=-\infty}^{\infty} Re[g^2_{xr}(n)] + \sigma^2_{xi} \sum_{n=-\infty}^{\infty} Re[g^2_{xi}(n)] \qquad \text{(III.7)}$$

In (III.6) and (III.7), $Re[g_{xr}(n)]$ and $Re[g_{xi}(n)]$ are the unit-sample responses from a given input signal, Re[x(n)] and Im[x(n)], to the output of the system, Re[y(n)]. Knowing σ^2_f and σ^2_y, one can easily compute the SNR as,

$$SNR = \sigma^2_y / \sigma^2_f \qquad \text{(III.8)}$$

The above steps give the method to find the signal-to-noise ratio. A simulation on a general purpose computer can

give an insight into the SNR of a digital system realized with finite precision arithmetic. A usual procedure is to

1. Generate random sequences Re[x(n)] and Im[x(n)],
2. pass the random sequences through an almost ideal system and then through a replica of the system but realized with finite precision arithmetic,
3. finally compute the SNR by comparing the two system outputs with each other.

The block diagram on figure III.3 summarizes the procedure using $N=2^q$. This choice permits the use of the FFT algorithm for computing the near exact transform therefore reducing the computing time on the general purpose computer.

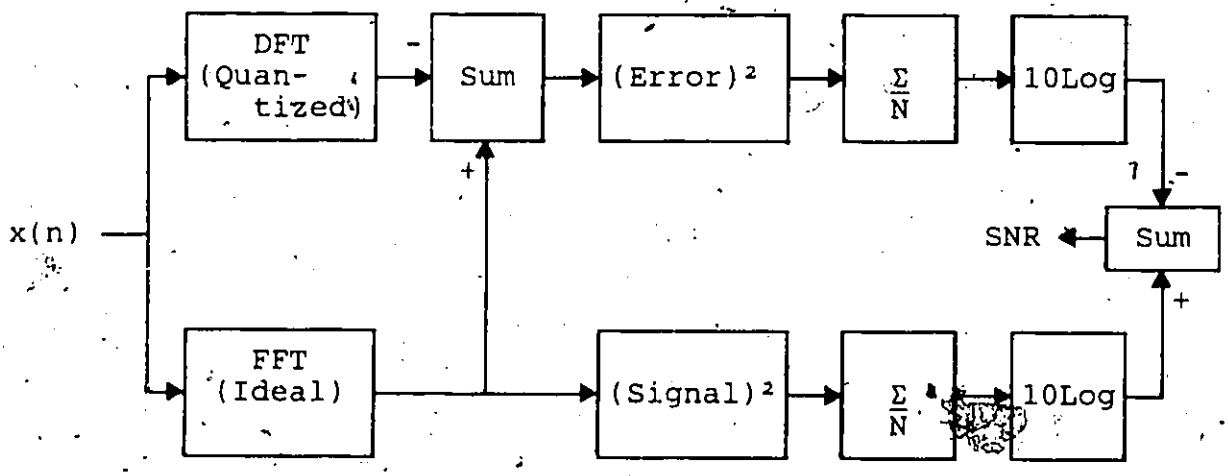


Figure III.3. Evaluation of the SNR by a computer simulation for N a power of 2.

APPENDIX IV

Summation of a complex trigonometric function, see (5.27)

$$\sum_{n=0}^{N-1} \frac{\cos(\theta)\,\sin(\theta(n+1)) - \sin(\theta n) + \sin^2(\theta(n+1))}{\sin^2(\theta)} =$$

$$\sum_{n=0}^{N-1} \frac{\sin^2(\theta(n+1)) + \sin^2(\theta n) - 2\cos(\theta)\,\sin(\theta n)\,\sin(\theta(n+1))}{\sin^2(\theta)} =$$

Expanding $\sin(\theta(n+1))$ into sin and cos function of (θn) and (θ) only, yield the following equation,

$$\sum_{n=0}^{N-1} \frac{(1-\cos^2(\theta))\,\sin^2(\theta n) + \cos^2(\theta n)\,\sin^2(\theta)}{\sin^2(\theta)} =$$

$$\sum_{n=0}^{N-1} \frac{(\cos^2(\theta n) + \sin^2(\theta n))\,\sin^2(\theta)}{\sin^2(\theta)} = \sum_{n=0}^{N-1} 1 = N$$

REFERENCES

[1] "Fourier,Jean-Baptiste-Joseph,Baron",Encyclopaedia Britannica,vol.7, 1974,p.577.

[2] B.Gold and C.M.Rader,"Digital processing of signals",N.Y., McGraw-Hill,1969.

[3] M.T.Heideman,D.H.Johnson and C.S.Burrus,"Gauss and the history of the fast Fourier transform",IEEE ASSP mag.,Vol.1, No.4,1984,pp.14-21.

[4] A.V.Oppenheim,ed.,"Application of digital signal processing" ,Prentice-Hall,Englewood Cliffs,N.J.,1978.

[5] J.W.Cooley and J.W.Tuckey,"An algorithm for the machine computation of complex Fourier series",Math.Comput.,19,1965,pp. 297-301.

[6] B.C.Cole,"Signal processing:a big switch to digital",Electronics:Special report,Vol.58,No.34,August 26,1985,pp.42-46.

[7] T.Feldt,"Digital signal processors:doing more with less", Electronics Week:Special report,Vol.,No.,November 19,1984, pp.65-69.

[8] F.Ware et al.,"Fast 64-bit chip set gangs up for double-precision floating-point work",Electronics,Vol.57,No.28, July 12 1984,pp.99-103.

[9] "Memories",Electronics Week:Technology readout,Vol.58,No.16, April 22,1985,pp.52-54.

[10] D.G.Fairbairn,"VLSI:A new frontier for systems designers", IEEE computer mag.,Vol.15,January 1982,pp.87-96.

[11] C.L.Seitz,"Concurrent VLSI architectures",IEEE Trans. on comp.,Vol.C-33,No.12,December 1984,pp.1247-1265.

[12] O.Monkewich and W.Steenaart,"Companding for digital filters" ,in IEEE Proc. ISCAS,1975,pp.68-71.

[13] "EPROM prices dive to less than $5",Electronics,Vol.58, No.25,June 24,1985,p.24.

[14] A.V.Oppenheim and R.W.Shafer,"Digital signal processing", Prentice-Hall,Englewood Cliffs,N.J.,1975.

[15] W.D.Stanley,G.R.Dougherty and R.Dougherty,"Digital signal processing",Reston,Va.,2nd edition,1984.

[16] R.W.Ramirez,"The FFT:Fundamentals and concepts",1985.

[17] M.Kunt,"Traitement numérique des signaux",Dunod,Paris,3ième édition,1981.

[18] H.J.Nussbaumer,"Fast Fourier transform and convolution algorithms",Springer-Verlag,N.Y.,1981.

[19] E.O.Brigham,"The fast Fourier transform",Prentice-Hall,Englewood Cliffs,N.J.,1974.

[20] G.Goertzel,"An algorithm for the evaluation of finite trigonometric series",Amer. math. monthly,Vol.65,January 1958,pp. 34-35.

[21] C.S.Burrus and T.W. Parks,"DFT/FFT and convolution algorithms",John-Wiley & Sons,N.Y.,1984.

[22] T.E.Curtis and J.T.Wickenden,"Hardware-based Fourier transforms:algorithms and architectures",IEE Proc.,Vol.130,Pt.F, No.5,August 1983,pp.423-432.

[23] Y.S.Wu,A.G.Constantinides,T.E.Curtis and L.J.Wu,"Architectural approach to alternate low-level primitive structures (ALPS) for acoustic signalprocessing", IEE Proc., Vol.131, Pt.F,No.3,June 1984,pp.327-333.

[24] J.S.Ward,"Number representations for prime length DFT's", Electronics Letters,Vol.20,No.7,March 29,1984,pp.301-302.

[25] L.R.Rabiner,R.W.Shafer and C.M.Rader,"The Chirp-z transform",IEEE Trans. on Audio and electroacoustics,Vol.AU-17,No.2,June 1969,pp.86-92.

[26] _____,The Chirp-z transform and its applications",BSTJ,May-June,1969.

[27] L.I.Bluestein,"A linear filtering approach to the computation of discrete Fourier transforms",IEEE Trans. Audio and electroacoustics,Vol.AU-18,December 1970,pp.451-455.

[28] C.M.Rader ,"Discrete Fourier transforms when the number of data samples is prime",Proc. IEEE,56,1968,pp.1107-1108.

[29] B.Arambepola,"Discrete Fourier transform processor based on the prime-factor algorithm",IEE Proc.,Vol.130,Pt.G,No.4,August 1983,pp.138-144.

[30] I.J.Good,"The relationship between two fast Fourier transforms",IEEE Trans. on computers,C-20,1971,pp.310-317.

[31] S.Winograd,"On computing the discrete Fourier transform", Math.Comput.,32,1978,pp.175-199.

[32] D.P.Kolba and T.W.Parks,"A prime factor FFT algorithm using high speed convolution",IEEE Trans.,ASSP-25,1977,pp.281-294.

[33] N.U.Chowdary,"Architectural considations of special purpose FFT processors and their applications to radar signal processing",Ph.D.Thesis,1984,Chap.4.

[34] T.F.Quatieri,Jr.,"Short-time spectral analysis with the conventional and sliding CZT",IEEE Trans.;ASSP-26,No.6,December 1978,pp.561-566.

[35] J.H.Halberstein,"Recursive,complex Fourier analysis for real -time applications",Proc.IEEE,1966,p.903.

[36] C.H.Ting,"Fourier transform faster than fast Fourier transform (FFT)",SPIE,Vol.241,Real-time signal processing III, 1980,pp.167-171.

[37] J.Choi and J.R.Johnson,"A modified discrete Fourier transform algorithm: a delta rotation algorithm",IEEE Proc. Int. Conf. Acoustics, Speech, and Signal Processing,ICASSP,May 1984, pp.28.A.1.1-4.

[38] C.Mead and L.Conway,"Introduction to VLSI systems"Addison-Wesley,1980.

[39] S.Y.Kung,"VLSI array processors",IEEE ASSP mag.,Vol.3,No.3 ,July 1985,pp.4-22.

[40] M.J.Foster and H.T.Kung,"The design of special-purpose VLSI chips",IEEE computer mag.,Vol.13,January 1980,pp.26-40.

[41] J.K.Hartt and L.Sheats,"Application of pipeline FFT technology in radar signal data processing",Eascon Record, 1971, pp.216-221.

[42] L.J.Siegel,"Highly parallel architectures and algorithms for speech analysis",IEEE Proc. ICASSP,May 1984,pp.25.A.1.1-4.

[43] H.M.Halpern and R.P.Perry,"Digital filters using fast Fourier transforms",Eascon Record, 1971, pp.222-230.

[44] E.E. Swartzlander Jr. and G. Hallnor, "Fast transform processor implementation", Proc. IEEE Int. Conf. ASSP, 1984, pp.25.A.5.1-4.

[45] H.T.Kung,"Why systolic architectures?",IEEE computer mag., Vol.15,January 1982,pp.37-46.

[46] H.T.Kung,"Let's design algorithms for VLSI systems",Caltech conf. on VLSI,January 1979,pp.65-90.

[47] J.B.Dennis,"Dataflow super computer",IEEE computer mag., Vol.13,November 1980,pp.48-56.

[48] S.Y.Kung et al.,"Wavefront array processor:language,architecture,and applications",IEEE trans. on computers,Vol.C-31,No.11,November 1982,pp.1054-1066.

[49] R.B.Urquhart and D.Wood,"Systolic matrix and vector multiplication methods for signal processing,"IEE Proc.,Vol.131 ,Pt.F,No.6,October 1984,pp.623-631.

[50] J.R.Lineback,"Parallel processing: why a shakeout nears", Electronics,Vol.58,No.43,October 28th 1985,pp.32-34.

[51] H.T.Kung,"Special-purpose devices for signal processing and image processing: an oppurtunity in very large scale integration (VLSI)",SPIE,Vol.241,Real-Time Signal Processing III, 1980,pp.76-84.

[52] H.T.Kung and C.E.Lieserson,"Systolic arrays (for VLSI)", in I.S.Duff and G.W.Stewart (Edits),Sparse matrix proceedings, SIAM,1978,pp.256-262.

[53] T.Willey,R.Chapman,H.Yoho and D.Preis,"Systolic implementations for deconvolution, DFT and FFT",IEE Proc.,Vol.132, Pt.F,No.6,October 1985,pp.466-472.

[54] G.H.Allen,P.B.Denyer and D.Renshaw,"A bit serial linear array DFT",Int.Conf.ASSP,San-Diego,May 1984,pp.41.A.1.1-4.

[55] C.D.Thompson,"Fourier transforms in VLSI",IEEE Trans. on computers,Vol.C-32,No.11,November 1983,pp.1047-1057.

[56] D.E.Knuth,"Seminumerical algorithms",in The art of computer programming,Vol.2.Reading,MA:Addison-Wiley Inc.,1980.

[57] F.Bonzanigo,"An improvement of tribolet's phase unwrapping algorithm",IEEE Trans. ASSP, Vol.ASSP-26, No.1, February 1978, pp.104-105.

[58] J.V.McCanny and J.G.McWhirter,"Bit-level systolic array circuit for matrix vector multiplication",IEE Proc.,Vol.130, Pt.G,No.4,August 1983,pp.125-130.

[59] D.Bhagat and H.W.Mergler,"A high-performance microprocessor based FFT processor",IEEE Trans. Industrial electronics and control instrumentation, Vol.25, No.2, May 1978, pp.102-107.

[60] C.J.Weinstein,"Roundoff noise in floating point fast Fourier transform computation", IEEE Trans. Audio Electroacoust., Vol.AU-17, September 1969, pp.209-215.

[61] S.Sweitzer,"A low cost FFT Chip set",Proc. Int. Conf. ASSP, 1984, pp.44.3.1-3.

[62] G.D.Covert,"A 32 point monolithic FFT processor chip", Proc. Int. Conf. ASSP, 1982,pp.1081-1083.

[63] K.Hwang and F.A.Briggs,"Computer architecture and parallel processing",McGraw-Hill,1984.

[64] Earl E.Swartzlander,"VLSI signal processing systems", Kluder Academic Publishers, 1986.

[65] Peter N.Kogge,"The architecture of pipelined computers", McGraw-Hill, 1981.

[66] J.A. Beraldin, T. Aboulnasr and W. Steenaart,"Efficient one-dimensional systolic array realization of the discrete Fourier transform", EUSIPCO,The Hague, September 2-5, 1986.