National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform 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.

AVIS

La qualité de cette microforme 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.

Canada

VLSI Systolic Array Architectures for the 1-D and 2-D Discrete Fourier Transform

by

Siying Gu

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

ISBN 0-315-82506-5

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

Real-time digital signal and image processing applications demand high speed operations for massive computations. The discrete Fourier transforms(DFT), the frequency domain description of digital samples, are a special case of them. In this thesis, we propose efficient systolic array architectures for the 1-D and the 2-D DFT using the second-order Goertzel algorithm. This algorithm improves efficiency by reducing the number of required multiplications by two, compared to the direct calculation or the first-order Goertzel algorithm, and yields efficient systolic array mappings. For the 1-D DFT, two 1-D and one 2-D systolic arrays are proposed. The two 1-D structures, a semi-systolic array and a pure-systolic array, are characterized by regular, modular cell interconnections. Thus making the arrays compatible with VLSI design principles. They require a total of $N + 1$ processing element, N of which are identical(where N is the length of the DFT). The hardware complexity of a basic cell is 4 adders and 2 multipliers. In addition, for each N cells, there is a shared cell with 4 adders and 4 multipliers. These arrays perform at an effective throughput rate of one DFT sample per clock cycle. The proposed 2-D array structure, obtains a higher throughput rate of one DFT transform per clock cycle by increasing the number of processing elements N times, while keeping the simplicity of the basic cell. As for the 2-D DFT, a 2-D systolic array architecture is developed which does not require a row-column transposition while some delay units are needed between the two stages. The row and the column computations of the 2-D transform are similar and can be realized on two identical

chips. This array has a high throughput rate of one 2-D transform every N clock cycles. All the above proposed systolic arrays can process continuous flow of input data and perform at 100% efficiency. These structures are compared to other DFT systolic arrays regarding complexity and real-time implementation. An error analysis is given for the proposed systolic arrays and other existing DFT systolic arrays.

Acknowledgements

I wish to express my most sincere gratitude to my supervisor Dr. Willem Steenaart for his academic guidance and encouragement during this research work.

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

The author is very grateful to Dr. Tyseer Aboulnasr for her valuable technical advice and her help.

Many thanks to all professors, colleagues and friends at the Department of Electrical Engineering, University of Ottawa, for their help and encouragement during the period of my studies. I am particularly indebted to Drs. Jinyun Zhang, Genzao Zhang and Tet Yeap, for their significant support throughout my studies.

Finally, the author wishes to thank her family for their constant encouragement during her studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 DFT and real-time digital signal processing

Modern digital signal processing(DSP) plays an important role in a wide range of fields, such as speech processing, data communication, weather, sonar, radar, video compression, HDTV and so on. It converts a continuous time signal to a series of discrete data samples where digital computers can be used to extract characteristic parameters or to process these sequences, such as removing interference. In general, the mathematical and algorithmic techniques encompassed in DSP are diverse. However, most DSP algorithms are dominated by convolution/correlation filtering, linear algebraic methods and transform techniques [1], such as the discrete Fourier transform(DFT).

The DFT is a key algorithm in frequency-domain description of signals and linear system functions. The dominating aspects for the DFT, as well as for many other DSP algorithms, are mathematically intensive computations and often real-time operation. For example, the number of complex computations required to compute a N-point DFT is proportional to N^2. This computation is mathematically intensive, especially when N is large. In addition, the DFT must be performed in real-time. A computation rate in excess of 1 billion operations per second may frequently become necessary. This

requires essentially large throughput rates and huge amounts of data and memory.

A variety of DFT processors has been proposed and implemented to meet increasing performance requirements of signal processing applications. In the past decade, two approaches have been used to implement signal processing algorithms. One is the programmable digital signal processors(DSPs). The DSPs strongly favor multiply/accumulate based algorithms. Their programmable nature make them very flexible. They can be used in applications where throughput rates per sample ranging in the tens of kHz are sufficient.

Another possibility is to build special-purpose DFT processors that are optimized for processing specific task - e.g. the DFT computation. These usually require more hardware than the DSPs but can achieve higher throughput rates. At this moment these appears to be no reference with respect to an actually realized circuit. Only a few potential realizations were reported [9] [12] [16] [13] [6] [17].

To implement the time-critical and computationally complex DFT computations in hardware in order to accelerate the speed of execution, the special-purpose DFT processor has to be cost-effective in terms of production cost and design turn-around time. Systolic array processors are a type of special-purpose systems that potentially offer a promising solution to meet real-time processing requirements and has structural properties that are suitable for very-large-scale integration(VLSI).

The fast Fourier transforms, known as FFT, derive their computational advantage by transforming a DFT into a set of lower-order sections. When computed on a general-purpose computer, they usually outperform any DFT algorithms reducing the order of complexity of the transform from N^2 to $NlogN$, where N is the length of the DFT. However, due to the characteristics of the algorithms, it has some drawbacks for systolic array implementations as will be shown in chapter 2. Therefore, we shall focus on VLSI

DFT array processors realizing the DFT in this thesis.

1.2 VLSI architectural design principles

VLSI architectures should fully exploit the potential of VLSI technology and also take into account the cost of the silicon area and the input/output(I/O) pins; the layout constraint and the resultant interconnection costs in terms of area and time. S.Y. Kung summarized some VLSI design principles in [2]:

1. Regularity and Modularity

   In designing special-purpose systems, there is an emphasis on keeping the overall architecture as regular and modular as possible, thus reducing the overall complexity. If a structure can be decomposed into a few types of simple substructures or building blocks, which are used repetitively with simple interfaces, high speed structures can be implemented.

2. Pipeline and Parallel Processing

   Real-time DSP requires extensive concurrency by either pipeline processing or parallel processing. The throughput rate is the overriding factor dictating the system performance in many DSP applications. To optimize the throughput, a different design choice is often made than that of minimizing the total processing time. Pipeline techniques fit our goal of improving throughput rate. For signal processing arrays, pipelining and parallelism at all levels should be pursued. It may bring about an extra order of magnitude in performance with very little additional hardware.

3. Local Communication vs Global Communication

   When a large number of processing elements work simultaneously, coordination and communication becomes significant - especially with VLSI technology. Since communication in VLSI is very expensive in terms of area, power, and time consumption, it has to be restricted to localized interconnections. To avoid global interconnections, local, regular data movement is strongly preferred. Sometimes, the broadcasting technique can be used to exploit the concurrency in algorithms so that the throughput rate can meet the real-time requirements. Whenever broadcasting is used, time skew has to be considered and the circuit layout must be designed carefully.

Systolic array design that conforms to these rules will lead to cost-effective and efficient VLSI implementation.

1.3 Systolic array architectures

Systolic arrays for DFT computation was first proposed by H.T. Kung[3]. Since then, systolic arrays for VLSI implementation of many complex and computation-intensive algorithms have been proposed[4] [26] [31] [34] [29].

A systolic array is a set of processing elements(PEs), each capable of performing a simple function, interconnected in a regular pattern which depends on the function to be performed. The systolic array typically communicates with outside world(memory, I/O, or hose CPU) at the boundary cells. The structure is called *systolic* to draw an analogy to the systolic pumping of blood by the heart in the body's vascular system; in a systolic array, data is rhythmically 'pumped' from one PE to the next. At each PE, the data may undergo some operation to yield a modified value, that is then

pumped into the next PE in line. The key approach is to have local, rather than global communications since each PE is directly connected only to its nearest neighbors. Also, the data undergo computation operations in sequence, and I/O occurs only at the boundaries, thereby satisfying all the conditions for efficient VLSI design that were discussed above.

The basic principle of a systolic architecture is shown in Figure 1.1. By replacing a single processing element of a signal processing computer with an array of cells, faster computation can be achieved. Once a data item is brought in the systolic array it is used effectively at each cell of the array. This is possible for a wide class of compute-bound computations where multiple operations are performed on each data item in a repetitive manner. Here the terminology "compute-bound" means that in a computation, the total number of operations is larger than the total number of input (and output) samples.

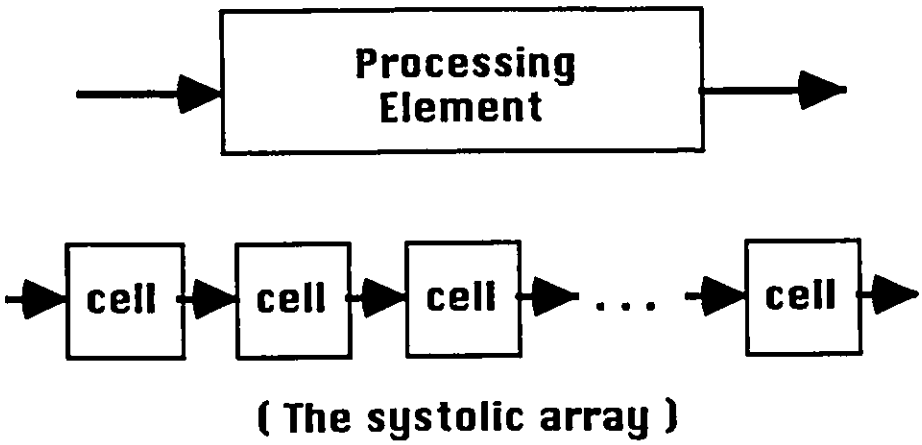

Figure 1.1: Basic principle of a systolic system

Besides the advantage of being able to use each input data item a number of times (and thus achieving high computation throughput, the systolic approach has many

other advantages, such as modular expandability, simple and regular data and control flows, use of uniform cells, elimination of global broadcasting, and fan-in and possibly fast response time.

To evaluate the efficiency of a systolic array structure, one usually uses the following criteria: number of processing elements, complexity of a processing element in terms of multipliers and adders required per cell, the computation time and throughput rate of the structure, communication complexity and whether the structure needs external memory to provide or collect the data samples and results. We will use these criteria to compare existing DFT systolic arrays in section 2.3.3.

1.4 Motivation and thesis plan

As mentioned in the previous sections, many high-performance applications use frequency-domain signal-processing techniques in which the DFT must be computed at very high speed. To implement the time-critical and computationally complex DFT in VLSI, special-purpose processors are desirable. In this thesis, possible VLSI implementations, i.e., systolic array architectures, to implement the DFT are studied.

In chapter 2, the DFT and its systolic array realizations are reviewed. The 1-D and the 2-D DFT and their properties are given and the computational algorithms reviewed. Here, the first-order and the second-order Goertzel algorithm are emphasized due to the fact that this algorithm is used in most of the existing DFT systolic array implementations. These arrays are then surveyed, for the 1-D and the 2-D DFT, respectively. The hardware complexity in terms of number of adders and multipliers, together with some other VLSI implementation requirements, such as the throughput rate and communication requirement, is presented. These structures are compared at

the end of the chapter.

In chapter 3, we introduce an improvement to the systolic array implementation of the DFT by using the second-order Goertzel algorithm. For the 1-D DFT, two 1-D and one 2-D systolic arrays are proposed, and one 2-D structure for the 2-D DFT. The chapter ends with a comparison to the existing structures reported in chapter 2 concerning hardware complexity and several other VLSI implementation issues that we mentioned above.

Chapter 4 gives, for the proposed DFT systolic arrays and other existing DFT arrays in general, an error analysis for fixed-point and floating-point realizations. The analysis shows both advantages and limitations on the use of the DFT systolic arrays. Finally, in chapter 5, a conclusion is given and topics for further studies are mentioned.

Chapter 2

Overview of DFT and its Systolic Array Architectures

2.1 Introduction

In this chapter, the Discrete Fourier Transform is introduced and its systolic array implementations are reviewed.

In the first part of the chapter, the 1-D and 2-D DFTs and their properties are first given. The computational algorithms are reviewed, with emphasis on the first and second-order Goertzel algorithms. The fast Fourier transforms are also introduced. In the second part, existing systolic array architectures are surveyed, for 1-D DFT and 2-D DFT, respectively. The hardware complexity in terms of number of multipliers and adders, together with the need for data exchange is presented. Finally, these structures are compared with respect to hardware complexity and speed of operation.

2.2 The Discrete Fourier Transform

In this section, the 1-D and 2-D discrete Fourier transforms(DFT) are introduced and their properties reviewed. The fast Fourier transforms(FFT) is also reviewed.

2.2.1 The 1-D Discrete Fourier Transform

The Discrete Fourier Transform is the accurate presentation of digital samples in frequency domain.

The one-dimensional Discrete Fourier Transform of a finite duration discrete-time signal is given by

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

where $W_N = e^{-j2\pi/N}$ and $k, n = 0, 1, 2, \cdots, N-1$, where

1. $x(n)$ represents the discrete-time signal after sampling a continuous-time signal at equally spaced time intervals (can be real or complex);
2. N is an integer number of the samples;
3. $X(k)$ represents the discrete-frequency transform function, i.e. the spectral lines.

The DFT is usually called the analysis transform and its inverse transform IDFT, the synthesis transform,

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \tag{2.2}$$

where $W_N = e^{-j2\pi/N}$ and $k, n = 0, 1, 2, \cdots, N-1$.

2.2.1.1 Different computational algorithms for 1-D DFT

In digital signal processing applications, we deal with a sequence of a finite number of samples. The DFT computes the frequency-domain representation of such sequences. The computational algorithms for the DFT can be classified into two categories: the DFT algorithms and the FFT(fast Fourier transforms) algorithms.

Among the many algorithms in the first category, the Goertzel algorithm is of great interest to us, i.e., for systolic array implementation. Three forms of this algorithm

have been developed: the first-order and the second-order Goertzel algorithms[5][7] and the modified first-order Goertzel algorithm[6]. The first-order Goertzel algorithm and the modified first-order Goertzel algorithm have been used for several systolic array implementations which are shown in section 2.3. The second-order Goertzel algorithm is used for our proposed structures in this thesis, as presented in chapter 4. Initially , these three algorithms are described in detail in section 2.2.1.3~2.2.1.5. For comparison, the method of direct calculation of the DFT is also described in section 2.2.1.2. The systolic array implementations of these algorithms are introduced in section 2.3.

For the second category, the FFT algorithm outperforms any DFT algorithm in terms of number of operations. However, this type of algorithm has certain disadvantages for VLSI implementation. For this reason, we focus our attention on DFT systolic array architectures. A general review and discussion for the FFT algorithm is given in 2.2.1.6.

2.2.1.2 Direct Calculation of the DFT

An obvious way to calculate the DFT of a signal x(n) is to implement the definition of the DFT directly. This is done by computing the desired values $X(k)$ according to Eq.(2.1).

The number of arithmetic operations per output sample required by an algorithm is an important measure of its efficiency. For a real input $x(n)$, the number of real multiplications needed for the computation of one frequency sample $X(k)$ is $2N$ and the number of additions is also $2N$. In the case where $x(n)$ is complex, the number of real multiplications and additions is doubled to $4N$. Therefore, to compute N frequency samples $X(k)$, the total number of both real multiplications and real additions is $4N^2$.

As for hardware implementation, $4N$ multipliers and $4N$ adders are needed for direct real time calculation of the DFT.

2.2.1.3 The first-order Goertzel algorithm

The next approach is rather different from the direct calculation of the DFT. It was shown that the DFT is the z transform of $x(n)$ evaluated on the unit circle[7]. In other words, calculating the DFT is the same as polynomial evaluation[5]. The z transform of $x(n)$ is given by

$$X(z) = \sum_{n=0}^{N-1} x(n)z^{-n} \tag{2.3}$$

This is illustrated for a length-4 sequence, which is a third-order polynomial, by

$$X(z) = x(0) + x(1)z^{-1} + x(2)z^{-2} + x(3)z^{-3} \tag{2.4}$$

The DFT from Eq.(2.1) can be found by evaluating Eq.(2.3) at $z = W_N^{-k}$, which can be written as

$$X(k) = X(z)|_{z=W_N^{-k}} = DFT[x(n)] \tag{2.5}$$

where

$$W_N = e^{-j2\pi/N} \tag{2.6}$$

The most efficient way of evaluating a polynomial is by "Horner's rule", also called nested evaluation. This is illustrated for the polynomial in Eq.(2.3) by

$$X(z) = [[x(3)z^{-1} + x(2)]z^{-1} + x(1)]z^{-1} + x(0) \tag{2.7}$$

This sequence of operations can be written as a linear difference equation of the form of

$$y(n) = y(n-1)z^{-1} + x(N-n) \tag{2.8}$$

with initial condition $y(-1) = 0$ and the desired result being the solution at $n = N$. This is given by

$$X(z) = y(N) \tag{2.9}$$

This can be viewed as a first-order filter with the input being the data sequence in reverse order and the value of the polynomial at z being the solution sampled at $n = N$. Applying this to the DFT gives the Goertzel algorithm[7] for a particular k which is

$$y_k(n) = W_N^k y_k(n-1) + x(N-n), \qquad y_k(-1) = 0 \tag{2.10}$$

with

$$X(k) = y_k(N-1) \tag{2.11}$$

and the transfer function of the the system is[7],

$$H_k(z) = \frac{1}{1 - W_N^{-k} z^{-1}} \tag{2.12}$$

The flow graph of the algorithm is shown in Figure 2.1.

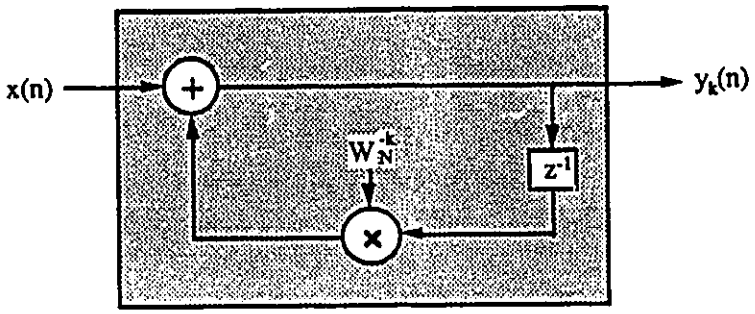

Figure 2.1: Flow graph of an arithmetic cell for the 1st-order Goertzel algorithm

When comparing this algorithm with the direct calculation, it is seen that the number of real multiplications and additions are the same. If the general input $x(n)$

and the coefficient W_N^{-k} are both complex, the computation of $X(k)$ for a particular k using the system of Figure 2.1 requires $4N$ real multiplications and $4N$ real additions. The total number of operations are $4N^2$ real additions and $4N^2$ real multiplications.

Although this procedure is slightly less efficient than the direct method, it avoids the propagation of the coefficients W_N^{-k} since these quantities are stored in each element as in Figure 2.1, and are used there recursively for N cycles of computations.

The hardware requirement for the first-order Goertzel algorithm is $4N$ multipliers and $4N$ adders.

2.2.1.4 The modified first-order Goertzel algorithm

The first-order Goertzel algorithm uses the input data sequence in reverse order. This requires a pre-arrangement of the sequence or a pre-loading phase for hardware implementation. Another form of the algorithm, 'the modified Goertzel algorithm' developed in [6] eliminates this disadvantage.

In order to process a continuous flow of data sequences, the polynomial in Eq.(2.7) should be evaluated with the data samples in the proper order. This is illustrated by rewriting Eq.(2.7) with samples in the proper order and by replacing $X(z)$ by $A(z)$.

$$A(z) = [[x(0)z^{-1} + x(1)]z^{-1} + x(2)]z^{-1} + x(3) \tag{2.13}$$

As a recursive equation, Eq.(2.13) becomes

$$\begin{aligned} y(n) &= y(n-1)z + x(N-n) \\ A(z^{-1}) &= y(N-1) \end{aligned} \tag{2.14}$$

where $y(-1) = 0$. It is possible to find $X(z)$ from $A(z)$,

$$X(z) = A(z^{-1})z \tag{2.15}$$

Therefore, the desired DFT sample at $n = N$ is given by

$$y(n) = y(n-1)z + x(N-n)$$
$$X(z) = y(N-1)z \tag{2.16}$$

Now that the algorithm accepts input data in the right order but the phase requires correction shown as term 'z' in the second equation of (2.16). This can be simplified by eliminating this term 'z', to form Eq.(2.17):

$$X(z) = [[x(0)z + x(1)]z + x(2)]z + x(3)]z \tag{2.17}$$

If the z on the right hand side of Eq.(2.17) is 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,

$$X(z) = [[x'(0)z + x'(1)]z + x'(2))z] + x'(3) \tag{2.18}$$

with

$$x'(z) = x(z)z \tag{2.19}$$

The sequence of operations in Eq.(2.18) and (2.19) can now be described as,

$$y(n) = z[y(n-1) + x(N-n)] \tag{2.20}$$

and

$$X(z) = y(N-1) \tag{2.21}$$

The corresponding linear recursive equations for the modified Goertzel algorithm of a particular k become,

$$y_k(n) = W_N^{-k}[y_k(n-1) + x(N-n)], \qquad y_k(-1) = 0 \tag{2.22}$$

with

$$X(k) = y_k(N-1) \tag{2.23}$$

Like the first-order Goertzel algorithm, the modified Goertzel algorithm does require the storage of only one coefficient W_N^k per section. Further, it improves the algorithm by taking the input data in the natural order. This avoids the problem of pre-arrangement or preloading of the data that the first-order Goertzel algorithm encounters for hardware implementation.

For this algorithm, the computation of a DFT sample at a particular 'k' is $4N$ real multiplications and $4N$ real additions. The total number of operations are $4N^2$ real multiplications and $4N^2$ real additions. The hardware requirement is $4N$ multipliers and $4N$ adders.

2.2.1.5 The second-order Goertzel algorithm

One of the reasons that the first-order Goertzel algorithm and the modified Goertzel algorithm did not improve efficiency is that the multiplication in the feedback or recursive path is complex and, therefore, requires 4 real multiplications. A modification of the scheme to make it second order removes the complex multiplication and reduces the number of required multiplications by two[7].

The system function of the second-order Goertzel algorithm can be obtained by multiplying both the numerator and the denominator of $H_k(z)$ in Eq.(2.12) by the factor $(1 - W_N^k z^{-1})$:

$$\begin{aligned} H_k(z) &= \frac{1 - W_N^k z^{-1}}{(1 - W_N^{-k} z^{-1})(1 - W_N^k z^{-1})} \\ &= \frac{1 - W_N^k z^{-1}}{1 - 2cos(2\pi k/N) z^{-1} + z^{-2}} \end{aligned} \tag{2.24}$$

The arithmetic cell of Figure 2.2 corresponds to the system of Eq.(2.24).

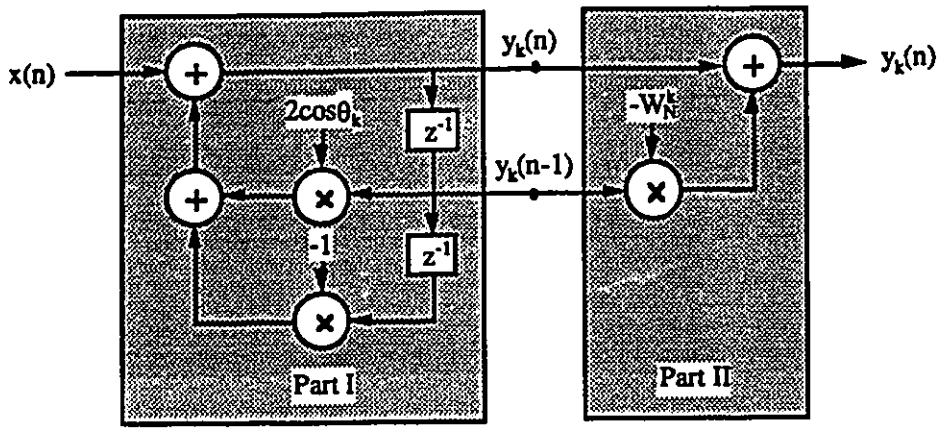

Figure 2.2: Flow graph of an arithmetic cell for the 2nd-order Goertzel algorithm

With complex input, only two real multiplications per sample are required to implement the pole of this system, since the coefficients are real and the factor -1 need not be counted as a multiplication. The complex multiplication by $-W_N^k$ required to implement the zero of the system function is performed only after the Nth iteration. So the total computation is $2N$ real multiplications and N real additions for the poles plus 4 real multiplications and 4 real additions for the zero. Therefore, the total computation is $2(N+2)$ real multiplications and $4(N+1)$ real additions, about half the number of real multiplications required with direct calculation and the first-order Goertzel algorithms. Besides, we still have the advantage of static coefficients of $2cos(2\pi k/N)$, and possibly W_N^{-k} as well, to be computed and stored.

The recursive form of Eq.(2.24) is

$$y_k(n) = x(n) + 2cos(2\pi k/N)y_k(n-1) - y_k(n-2) \tag{2.25}$$

Here, $y_k(n)$ represents the kth DFT sample at the nth recursive step and $k, n = 0, 1, 2, \cdots, N-1$. We suppose $y_k(-1) = 0$, and $y_k(-2) = 0$. After the Nth iteration, $y_k(N-2)$ and $y_k(N-1)$ are available. The frequency sample can be obtained by one

extra step:

$$X(k) = y_k(N-1) - W_N^k y_k(N-2) \tag{2.26}$$

While Eq.(2.25) is performed recursively N times in Part I of Figure 2.2, Eq.(2.26) is computed only once after the Nth iteration in Part II.

2.2.1.6 The Fast Fourier Transforms

The FFT(fast Fourier transforms) 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. Since Cooley and Tukey published the paper "An algorithm for the machine calculation of the complex Fourier series" in 1965, many algorithms [8] for efficient computation of DFT have been developed. The major ones are the Cooley-Tukey and the split-radix FFT, the prime factor algorithm and the Winograd FFT. The motivation for these algorithms is the number of operations needed for a particular algorithm. When computed on a general-purpose computer, they usually outperform any DFT algorithms reducing the order of complexity of the transform from N^2 to $NlogN$, where N is the length of the DFT.

With the development of parallel processing, FFT parallel processors have appeared [9]. Due to the characteristics of the algorithm, these processors have the following disadvantages for systolic array implementations:

1. The length N of input sequence must be a composite number, i.e., N must be a product of two or more factors. This brings about inconveniences for applications where N is not such a number.

2. The FFT processor will not work until all the N-length sampling sequence (or

most of them) have been entered. For a large N, there is a large latency.

3. The input data samples and the intermediate results require a special routing or shuffling from one node to another. Usually, routing is very area consuming in VLSI design.

For these reasons, there has been a great interest in designing DFT array processors. In this thesis, we focus our attention on DFT systolic array implementations that meet the VLSI design guidelines introduced in section 1.2.

2.2.2 The 2-D Discrete Fourier Transform

The 2-D DFT is required for applications such as image processing [10]. To calculate the 2-D DFT directly requires a large number of computations. One efficient way to compute the 2-D DFT is known as the row-column decomposition. This simple method uses 1-D DFT algorithms and offers considerable computational savings compared with direct computation. It is also the most popular 2-D DFT algorithm. In the following, the direct computation and the row-column decomposition method for the 2-D DFT are introduced.

2.2.2.1 Direct calculation of the 2-D DFT

The 2-D DFT for input samples $x(n_1, n_2)$ is

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} \tag{2.27}$$

where $n_1, k_1 = 0, 1, 2, \cdots, N_1 - 1$ and $n_2, k_2 = 0, 1, 2, \cdots, N_2 - 1$.

From Eq.(2.27), directly computing each $X(k_1, k_2)$ requires $N_1 N_2 - 1$ complex additions and $N_1 N_2$ complex multiplications. Since there are $N_1 N_2$ different values of

(k_1, k_2), the total number of arithmetic operations required in computing $X(k_1, k_2)$ from $x(n_1, n_2)$ is $N_1^2 N_2^2$ complex multiplications and $N_1 N_2(N_1 N_2 - 1)$ complex additions, or $4N_1^2 N_2^2$ real multiplications and $4N_1 N_2(N_1 N_2 - 1)$ real additions.

2.2.2.2 The row-column decomposition

To develop the row-column decomposition method, we rewrite Eq.(2.28) as follows:

$$X(k_1, k_2) = \sum_{n_2=0}^{N_2-1} \underbrace{\sum_{n_1=0}^{N_1-1} x(n_1, n_2) W_{N_1}^{n_1 k_1}}_{Y(k_1, n_2)} W_{N_2}^{n_2 k_2} \tag{2.28}$$

where

$$Y(k_1, n_2) = \sum_{n_1=0}^{N_1-1} x(n_1, n_2) W_{N_1}^{n_1 k_1} \tag{2.29}$$

Consider a fixed n_2, say $n_2 = 0$. Then $x(n_1, n_2)|_{n_2=0}$ represents a row of $x(n_1, n_2)$, and $Y(k_1, n_2)|_{n_2=0}$ is nothing but the 1-D N_1-point DFT of $x(n_1, n_2)|_{n_2=0}$ with respect to the variable n_1. Therefore, we can compute $Y(k_1, 0)$ from $x(n_1, n_2)$ by computing one 1-D N_1-point DFT. Since there are N_2 different values for n_2 in $Y(k_1, n_2)$ that are of interest to us, we can compute $Y(k_1, n_2)$ from $x(n_1, n_2)$ by computing N_2 1-D N_1-point DFTs.

Once we compute $Y(k_1, n_2)$ from Eq.(2.29), we can compute $X(k_1, k_2)$ from $Y(k_1, n_2)$ as follows:

$$X(k_1, k_2) = \sum_{n_2=0}^{N_2-1} Y(k_1, n_2) W_{N_2}^{n_2 k_2} \tag{2.30}$$

To compute $X(k_1, k_2)$ from $Y(k_1, n_2)$, consider a fixed k_1, say $k_1 = 0$. Then $Y(k_1, n_2)|_{k_1=0}$ represents a column of $Y(k_1, n_2)$, and $X(k_1, k_2)|_{k_1=0}$ is nothing but the 1-D N_2-point DFT of $Y(k_1, n_2)|_{k_1=0}$ with respect to the variable n_2. Therefore, we can compute $X(0, k_2)$ from $Y(k_1, n_2)$ by computing one 1-D N_2-point DFT. Since there

are N_1 different values for k_1 in $X(k_1, k_2)$ that are of interest to us, we can compute $X(k_1, k_2)$ from $Y(k_1, n_2)$ by computing N_1 1-D N_2-point DFTs.

From the proceeding discussion, we can compute $X(k_1, k_2)$ from $x(n_1, n_2)$ with a total of N_2 1-D N_1-point DFTs for the row operations and N_1 1-D N_2-point DFTs for the column operations. Suppose we compute the 1-D DFTs directly. Since direct computation of one 1-D N-point DFT requires N^2 multiplications and about N^2 additions, the total number of arithmetic operations involved in computing $X(k_1, k_2)$ is $N_1N_2(N_1 + N_2)$ complex multiplications and $N_1N_2(N_1 + N_2)$ complex additions. This is a significant computational saving relative to $N_1^2N_2^2$ complex multiplications and $N_1^2N_2^2$ complex additions required for direct computation of $X(k_1, k_2)$. In terms of real operations, the row-column decomposition needs $4N_1N_2(N_1 + N_2)$ real multiplications and $4N_1N_2(N_1 + N_2)$ real additions.

To further reduce the number of arithmetic operation, we can of course use 1-D FFT algorithms to compute the 1-D DFTs in the proceeding discussion. However, for the reasons presented in section 2.2.1.6, this is beyond our interest.

2.2.3 Discussion

In this section, the 1-D and the 2-D DFT are introduced and different computational algorithms were briefly reviewed. For the 1-D DFT, the Goertzel algorithm was given special attention because its high popularity for systolic array implementation. The main characteristics of the FFT algorithms are also highlighted. As for the 2-D DFT, the row-column decomposition method is introduced. The number of operations and hardware complexity for VLSI implementations are provided for the algorithms studied. These results will be used in the next section where existing systolic array structures,

based on different DFT algorithms, are reviewed.

2.3 Survey of existing systolic arrays for the DFT

In this section, existing systolic array structures for the 1-D and the 2-D DFT are surveyed and analyzed.

2.3.1 Existing systolic arrays for 1-D DFT

Several architectures have been proposed for mapping the 1-D DFT algorithm onto systolic arrays. They can be classified into two categories: one-dimensional and two-dimensional. These architectures are first presented, and the characteristics are compared and analyzed in section 2.3.3.

2.3.1.1 2-D array for matrix-matrix multiplication

The 1-D DFT of a finite length data segment can be expressed as a matrix-vector multiplication:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 \\ W^0 & W^2 & W^4 & W^6 & W^8 \\ W^0 & W^3 & W^6 & W^9 & W^{12} \\ W^0 & W^4 & W^8 & W^{12} & W^{16} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix} \tag{2.31}$$

Since we have, for $N = 5$

$$W^6 = W^1$$

$$W^8 = W^3$$

$$W^9 = W^4$$

$$W^{12} = W^2$$

$$W^{16} = W^1 \tag{2.32}$$

Eq.(2.31) can be rewritten as below

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 & W^4 \\ 1 & W^2 & W^4 & W^1 & W^3 \\ 1 & W^3 & W^1 & W^4 & W^2 \\ 1 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix} \tag{2.33}$$

For a sequence of such segments of data, the operation can be represented as matrix-matrix multiplication.

Urquhart and Wood[11] described a 2-D systolic array for such a matrix operation. Figure 2.3 shows the array for three sequences, $x1(n)$, $x2(n)$ and $x3(n)$ of length $N = 5$. For this structure, a total of N^2 processing elements(PEs) are required, and N^2 twiddle factors are static in the array, one in each PE. The parallelogram shapes represent the order of arrival of (1) the input data (right hand side) (2) the initial conditions (on top) and (3) the output DFT samples. The input $xi(n)$ (here, $i = 1, 2, 3$) are fed as columns of a matrix. The resulting frequency samples $Xi(k)$ are computed in the array and exit in the order shown.

Such an architecture requires N^2 PEs and an extra network to reorder the input data. At each clock cycle, N input samples are entered and N DFT samples leave the array. So the throughput is one N-point DFT per clock cycle. The computation time for a complete 1-D transform is $2N - 1$ clock cycles.

2.3.1.2 1-D array for matrix-vector multiplication

A 1-D systolic array that realizes the 1-D DFT algorithm as a matrix-vector multiplication is shown in Figure 2.4 for $N = 5$.

For this architecture, a total of N PEs are needed. At first, the N input $x(n)$ are

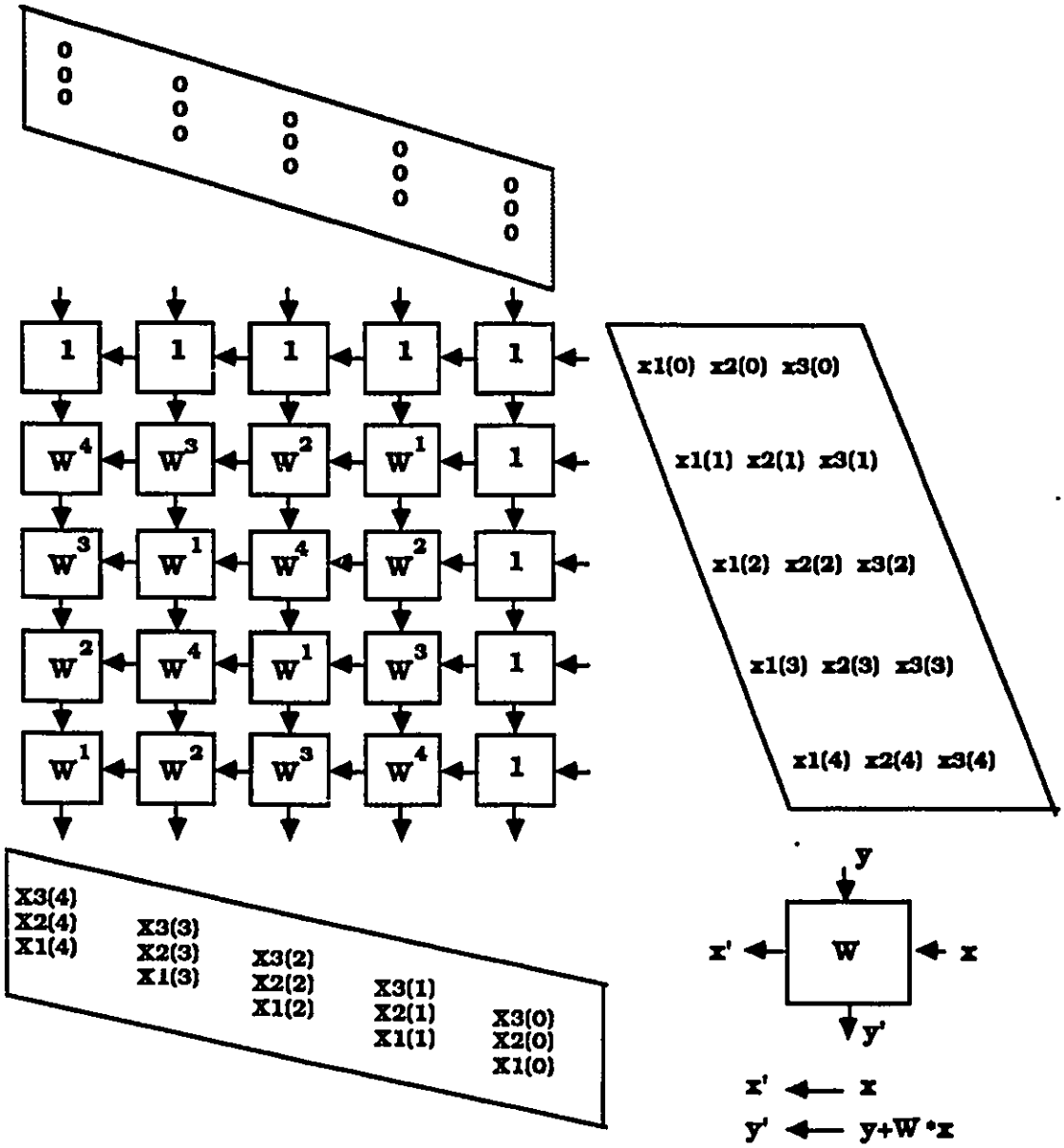

Figure 2.3: Matrix-matrix 2-D systolic array for 1-D DFT, $N = 5$

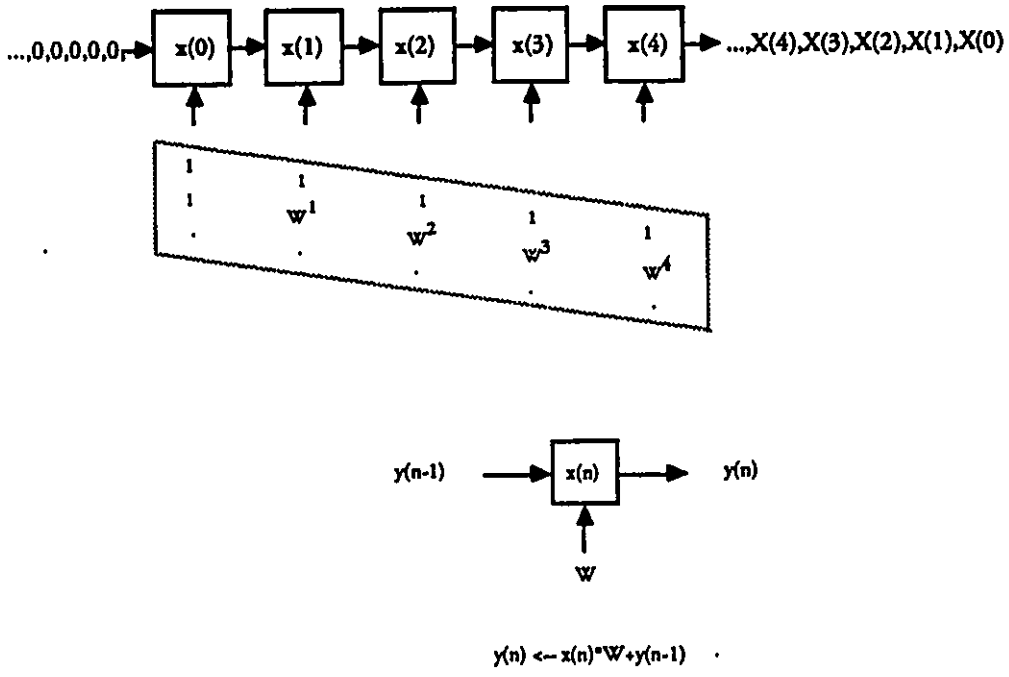

Figure 2.4: Matrix-vector 1-D systolic array for 1-D DFT, $N = 5$

stored in PEs, one in each PE. Then the matrix of the twiddle factors W_N^{nk} is passed into the array with delays between each column on top of the array. A sequence of initial values of zero is entered on the left hand side of the array. The computed DFT samples $X(k)$ exit on the right hand side.

This architecture needs N PEs. This is an improvement in terms of hardware complexity. However, due to the initial preloading of the input data, this array can not process continuous sequences of data. With the initial delay of N clock cycles, the computation time for a DFT of length N is $3N-1$ clock cycles. The throughput rate is one transform every $3N-1$ clock cycles.

2.3.1.3 1-D array using continuous data inputs

One 1-D systolic array structure can be obtained by direct mapping from the DFT equation (2.1). An example of $N=5$ is shown in Figure 2.5.

The input samples $x(n)$ are entered into the leftmost PE and are passed to the other $N-1$ PEs with one clock cycle delay in between. The twiddle factors matrix W_N^{nk} are passed into the array from the top while the partial results are retained in each PE. After N samples have passed through a PE, the partial result at that particular PE is the computed DFT sample. While this result is removed from the PE, the PE is reinitialized to zero therefore to be ready for the computation for the next sequence of inputs. The throughput rate is one DFT per N clock cycles. The computation time for the array is $2N-1$ clock cycles.

2.3.1.4 Kung's 1-D array

Based on the first-order Goertzel algorithm, a 1-D systolic array structure was proposed

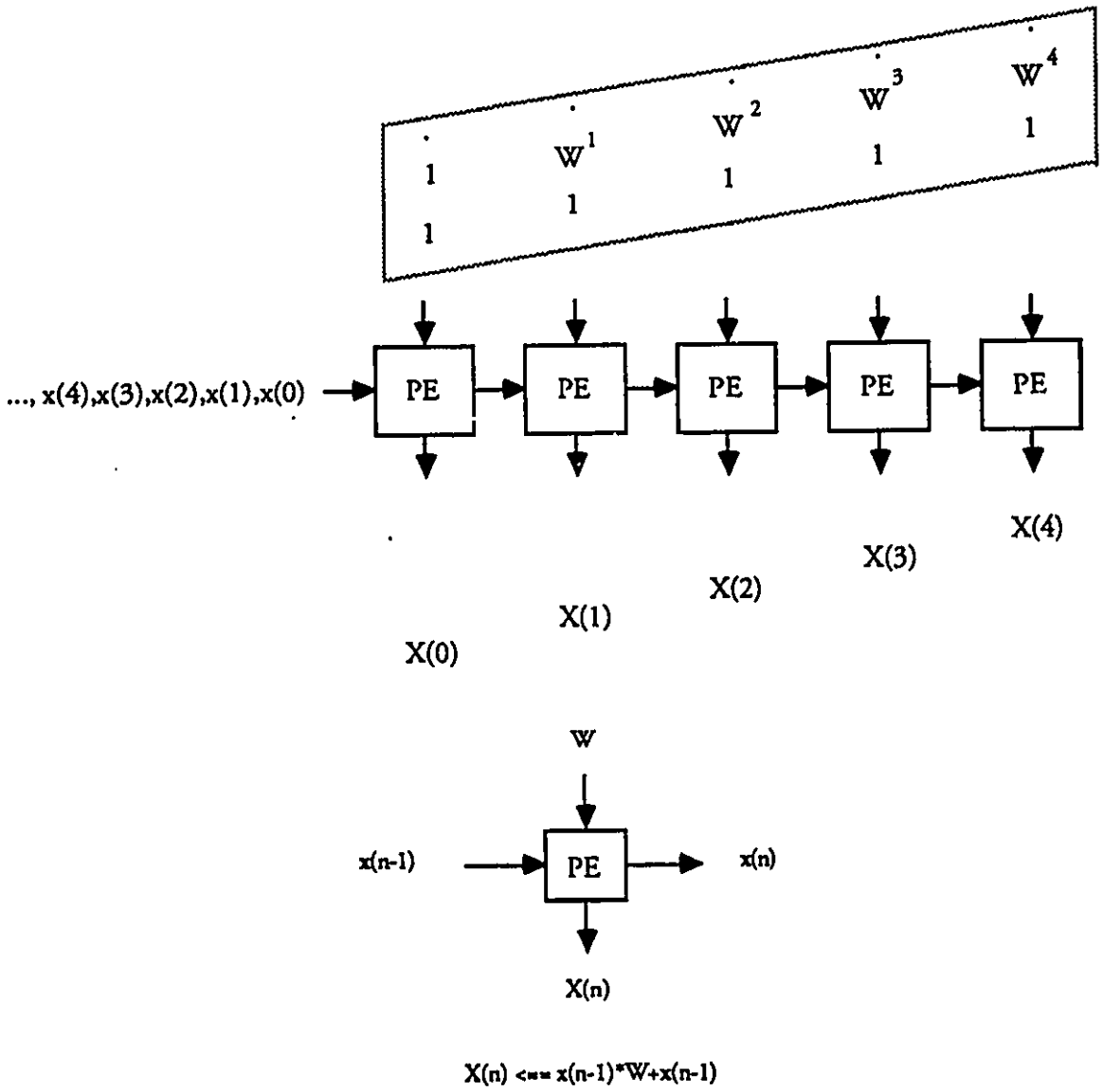

Figure 2.5: 1-D systolic array using continuous input for 1-D DFT, $N = 5$

to compute the DFT by H.T. Kung[3]. The principle of the computation of $N = 5$ is illustrated in Figure 2.6 for continuous clock cycles. The data sequence (except for the last data which here is $x(4)$) is first loaded in the array, as shown in the first line of the figure.

Then by moving the coefficients W^k, $(k = 0, 1, ..., 4)$, into the array, the DFT is computed, as shown in the second line to the fifth line of the figure. The first frequency sample $X(0)$ appears at the output of the rightmost PE when the fifth coefficient is loaded. Then, the other four frequency samples follow in order at each clock cycle. The function of the basic PE is also shown in Figure 2.6.

This architecture can be best used when a single DFT vector is needed or when a series of DFTs are to be computed but not in a continuous fashion. To make it suitable for a continuous flow of data sequences, some buffers for data management are required. This along with the longer computational period ($3N - 3$ cycles for a N-point DFT) is the disadvantages of this systolic array. The throughput rate is one transform every $3N - 3$ cycles.

2.3.1.5 Chang's 1-D array

Chang and Chen[12] proposed an alternative approach for the 1-D DFT that also uses the first-order Goertzel algorithm. Different from Kung's structure, the array is arranged so that it can take continuous input data and no preloading time is necessary.

Figure 2.7 shows the array for $N = 5$. Each PE k is responsible for the computation of $X(k)$ for $k = 0, 1, 2, ..., N - 1$. The coefficients W^k flow through the PEs except the leftmost PE where the corresponding input is always 1. The input samples enter the leftmost PE as the second input and is computed for complex addition and multiplication before the intermediate results are passed to the right hand side PE.

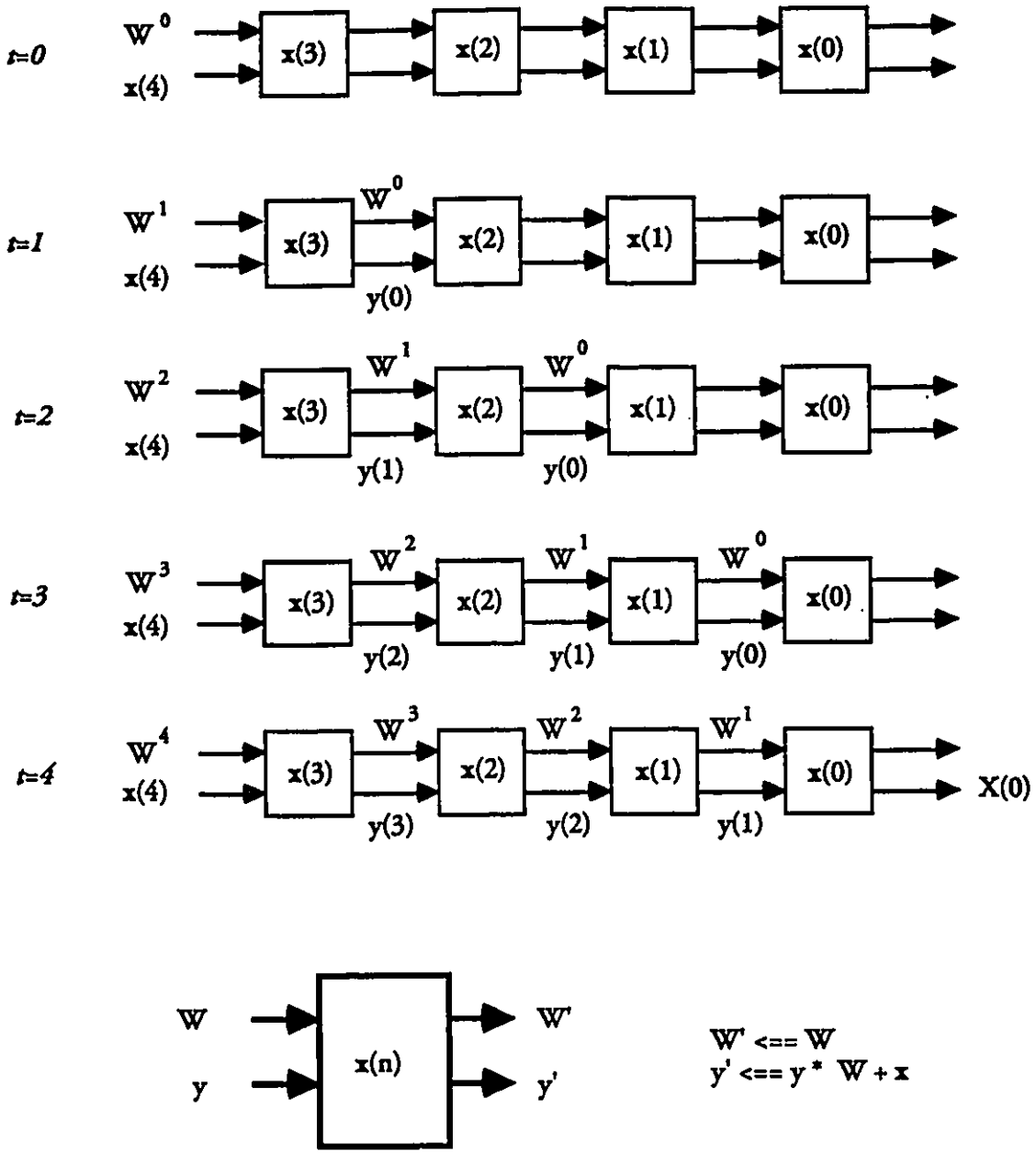

Figure 2.6: Kung's 1-D systolic array for 1-D DFT at successive clocks, $N = 5$

The transformed value X(0) is obtained in PE 0 after N cycles. The intermediate Z is then initialized to zero and is kept in a register. Since the computation of $X(k)$ is one cycle later than that of $X(k-1)$ for $k = 1, 2, ..., N-1$, the transformed values $X(0), X(2), ..., X(N-1)$ are available successively in PE 1, PE 2,..., PE $N-1$, respectively, with one cycle delay between each other.

For the above model, N PEs are required. Although the input sequence Y for PE 0 is different from that of the other $N-1$, it is suitable for a continuous flow of data sequences. An N-point transformed sequence is obtained after $2N-1$ cycles. The throughput is one N-point transform every N clock cycles.

2.3.1.6 Beraldin's 1-D arrays

Beraldin proposed a systolic array structure in [6] that uses the modified first-order Goertzel algorithm also derived in [6]. As shown in Figure 2.8 for $N = 4$, the data samples flow through the array in a pipeline fashion. Each data sample is used by a PE 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 PE for the computation of DFT sample. The recursive Eq.(2.22) is performed N times at each PE and the result is then available. For this semi-systolic array, a bus or tree-like network is used to collect the results.

To avoid the bus or tree-like network, the pure-systolic array was proposed as an alternative approach by adding two registers and combinational logic circuit in each PE, except for the last PE. As shown in Figure 2.9, the DFT samples are sent out systolically without the bus or tree-like network. After an initial delay of $2N-1$ cycles, the DFT samples leave the array in sequence every cycle. So the successive N-point DFT samples are computed every N clock cycles.

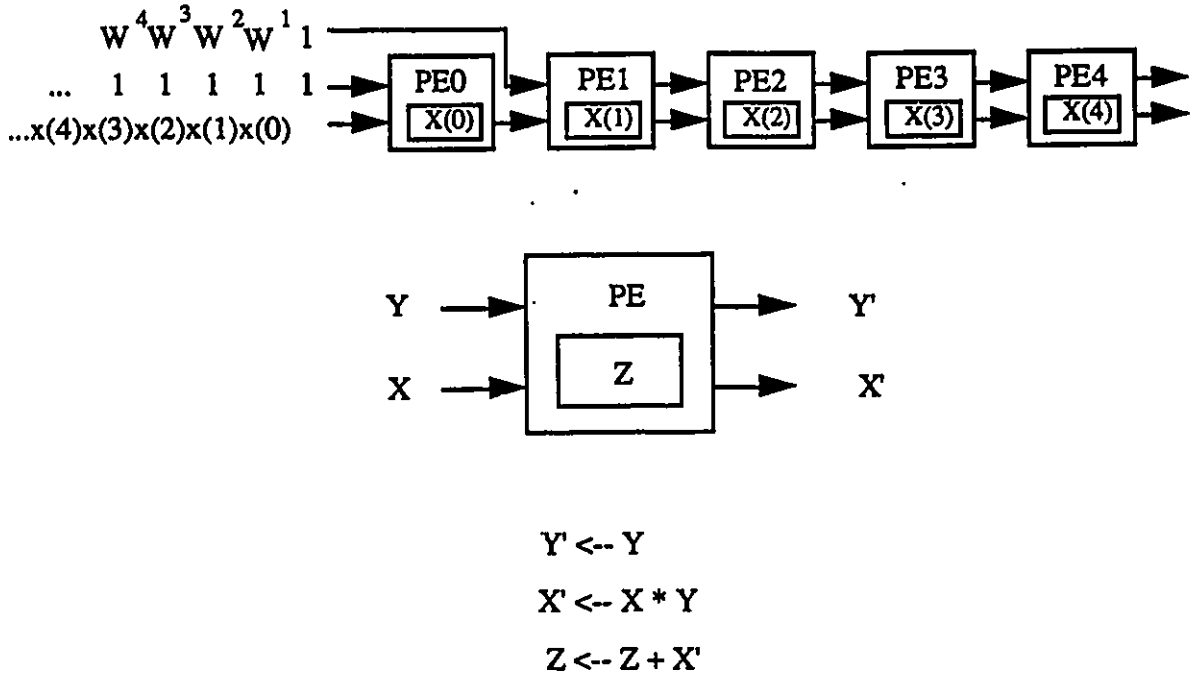

Figure 2.7: Chang's 1-D systolic array for 1-D DFT, $N = 5$

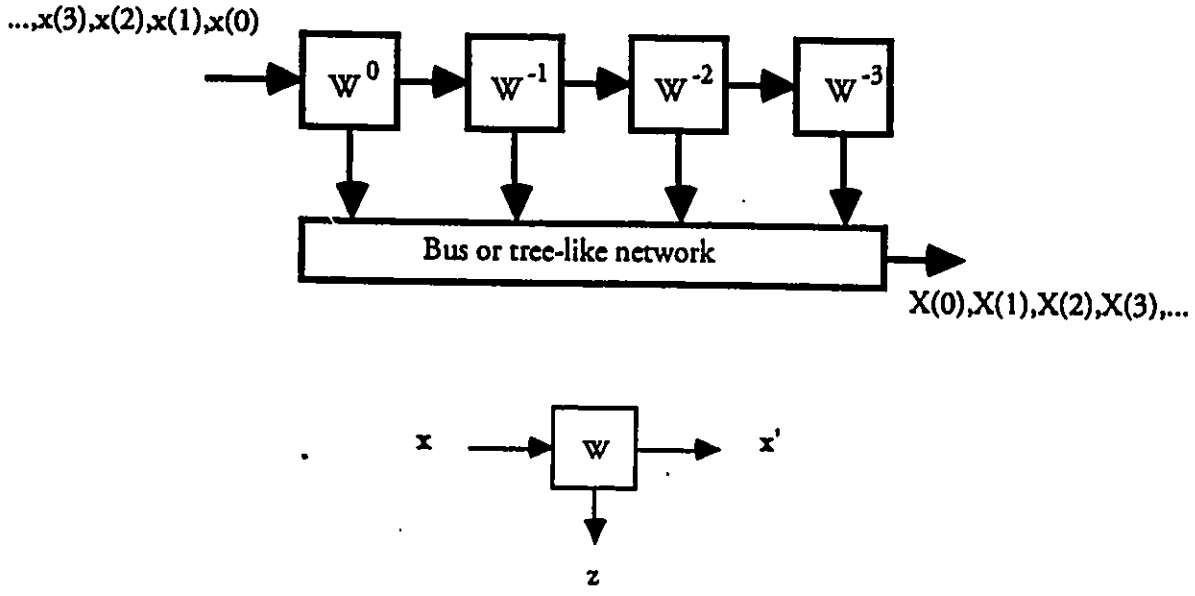

Figure 2.8: Beraldin's 1-D semi-systolic array for 1-D DFT, $N = 4$

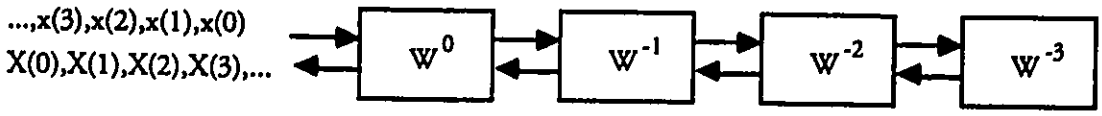

Figure 2.9: Beraldin's 1-D pure-systolic array for 1-D DFT, $N = 4$

This array requires N PEs and can handle continuous sequences of input data. Since the twiddle factors are static in each PE, stored-product ROM's can be used in place of multipliers to reduce area and improve precision accuracy. The throughput rate is one DFT of N per N clock cycles. The computation time for this pure-systolic array is one transform every $3N - 2$ cycles.

2.3.2 Existing systolic arrays for 2-D DFT

As introduced in section 2.2.2, the 2-D DFT is usually computed using the row-column decomposition. This method applies 1-D DFT algorithms for the row computation as well as for the column computation while a row-column transposition is needed between the two stages. Figure 2.10 shows the system diagram of such a structure. The two 1-D DFT are usually identical and can be used for either the 1-D or the 2-D DFT. In this case, any of the 1-D DFT systolic arrays introduced in section 2.3.1 can be used for the computation of the 2-D DFT, where a row-column transposition stage is added between the two 1-D DFT arrays. One such structure is introduced in section 2.3.2.1.

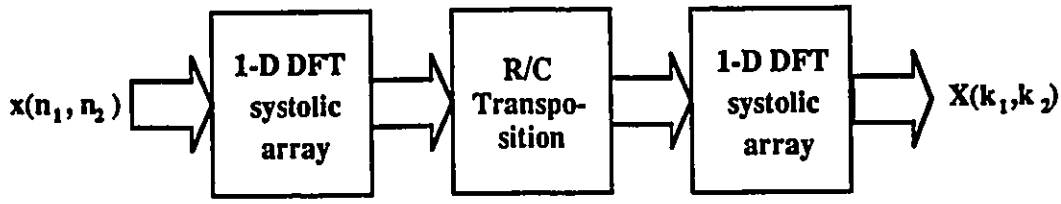

Figure 2.10: System diagram for 2-D DFT array implementation

The structure can be adapted, however, so that the row-column transposition is not required. Recently, a few such systolic array architectures have been presented. These

are introduced in sections 2.3.2.2~2.3.2.4.

2.3.2.1 2-D array with row-column transposition

As we have known, with the row-column decomposition method, any 1-D DFT array can be used for the row computation or for the column computation. In this section, we introduce a structure that applies the 1-D model using continuous data inputs to compute the 1-D DFTs. The row-column transposition is needed between the stages.

Figure 2.11 depicts the 2-D array that computes the 2-D DFT for $N_1 = N_2 = 3$. The row computation part is composed of 3 1-D arrays for continuous data inputs as shown in section 2.3.1.3. The coefficients W^k enter the upper row of the array with one delay between each column entry and are pipelined through to the lower PEs. At the same time, rows of input samples x_{n_1,n_2} enter the leftmost PEs with one delay between each entry to meet the W^k coefficients at appropriate times. A data bus is used for each array to pass the computed samples from the PEs to the R/C, the row-column transposition stage where columns of the inputs become rows of the outputs. The column computation part is identical to the row computation part and is also composed of 3 1-D arrays where the same computation process applies. Finally, the computed DFT samples X_{k_1,k_2} leave the 1-D arrays in sequence with one delay between each row, just like the inputs of the structure.

This architecture needs $2N^2$ PEs. Although no initial preloading of the input data is required, an input buffer and an output buffer are needed to arrange the data. The R/C part increases the hardware complexity and needs at least $N - 1$ delays. The coefficients W flow through the PEs so that they are not static. The throughput is one N by N transform every N clock cycles. The computation time for a 2-D transform is

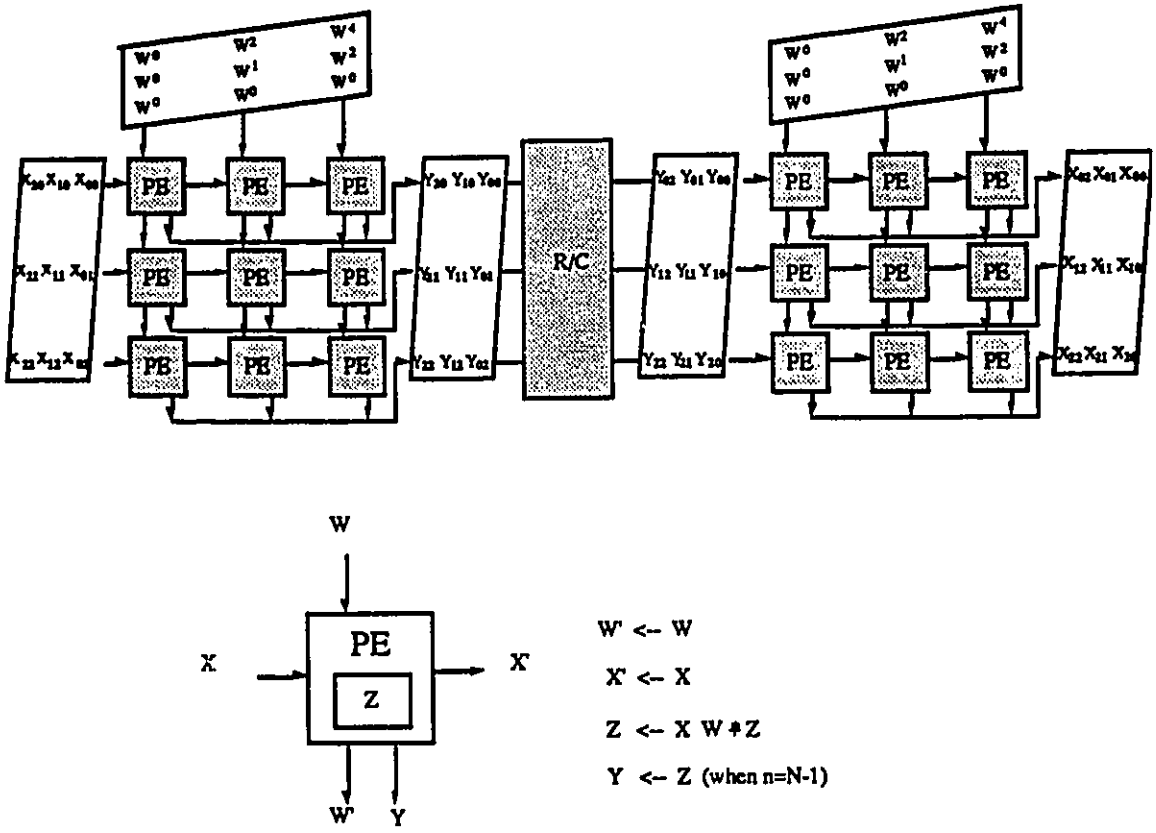

Figure 2.11: 2-D array for 2-D DFT with the row-column transposition

$4N - 4$ cycles.

2.3.2.2 Chang's 2-D array

Chang[12] proposed a 2-D DFT systolic array based on his model(section 2.3.1.5) and Kung's(section 2.3.1.4) for the two 1-D DFTs, both of which apply the Honor's rule. The mixed model eliminates the time consuming preloading phase and no row-column transposition is needed.

As shown in Figure 2.12, the connections of each row and each column are the same as Chang's model and Kung's model, respectively. The 2-D DFT can then be computed as follows: the row transforms are performed in the corresponding rows of the array, i.e., the 1-D DFT of the ith row of the input sequence is the same as the state of Kung's model after the preloading phase. Then each column of the 2-D array computes the column transforms of the 2-D DFT. The transformed sequence is obtained as in Kung's model and leave the array at the bottom.

The advantages of this mixed model are that no preloading phase is necessary before performing the row transforms or the column transforms, and that the transformed sequence leaves from the lower side of the array with no extra time necessary for the PE output. The structure has N^2 PEs(for $N_1 = N_2 = 3$), each of which has the complexity of Chang's model plus that of Kung's. The number of cycles needed to compute the 2-D DFT are $4N - 2$, i.e., $2N - 1$ cycles for the row transforms applying Chang's model, and $2N - 1$ cycles for the column transforms applying Kung's model. The system latency between two successive 2-D DFT is $4N - 2$ cycles. The throughput

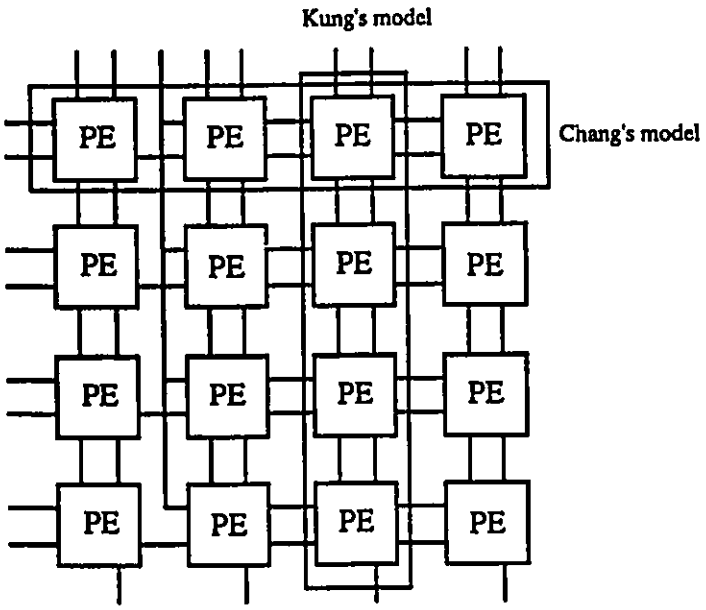

Figure 2.12: Chang's array for 2-D DFT

is one 2-D DFT every $4N-2$ clock cycles.

2.3.2.3 Zhu's 2-D array

A 2-D DFT array processor was introduced by Zhu[9]. The array is based on the modified Goertzel algorithm and accomplishes serial input, parallel output, parallel processing in one time.

The recursive equations derived by Zhu are based on two cases:

(1) when input sequence varying from $x(n_1, n_2)$ to $x(n_1+1, n_2)$, then

$$y_{k_1,k_2}(n_1+1, n_2) = y_{k_1,k_2}(n_1, n_2)W_{N_1}^{-k_1} + x(n_1+1, n_2)W_{N_1}^{-(k_1+k_2)} \tag{2.34}$$

(2) when input sequence varying from $x(N_1-1, n_2)$ to $x(0, n_2+1)$, then

$$y_{k_1,k_2}0, n_2+1) = [y_{k_1,k_2}(N_1-1, n_2) + x(0, n_2+1)]W_{N_1}^{-(k_1+k_2)} \tag{2.35}$$

Figure 2.13 shows the 2-D array. The input samples $X(n_1, n_2)$ is broadcast to all the PEs, one each clock cycles. Each PE computes one of the 2-D DFT samples. At each iteration, two complex multiplications and one complex addition are computed in the PEs. The array performs the 2-D DFT in one time, that is it does not need a second processing of the first result. This is different from the conventional method, which is to process first by row, then by column. When the sampling data enters, all the PEs perform in parallel and the 2-D DFT result is carried out after the last data enters and is calculated.

The structure needs N^2 PEs(for $N_1 = N_2 = N$). Global communication is required for the broadcast input. Since the coefficients W^k are static, ROM's can be used in place of multipliers. The computation time for one 2-D transform is N^2 clock cycles.

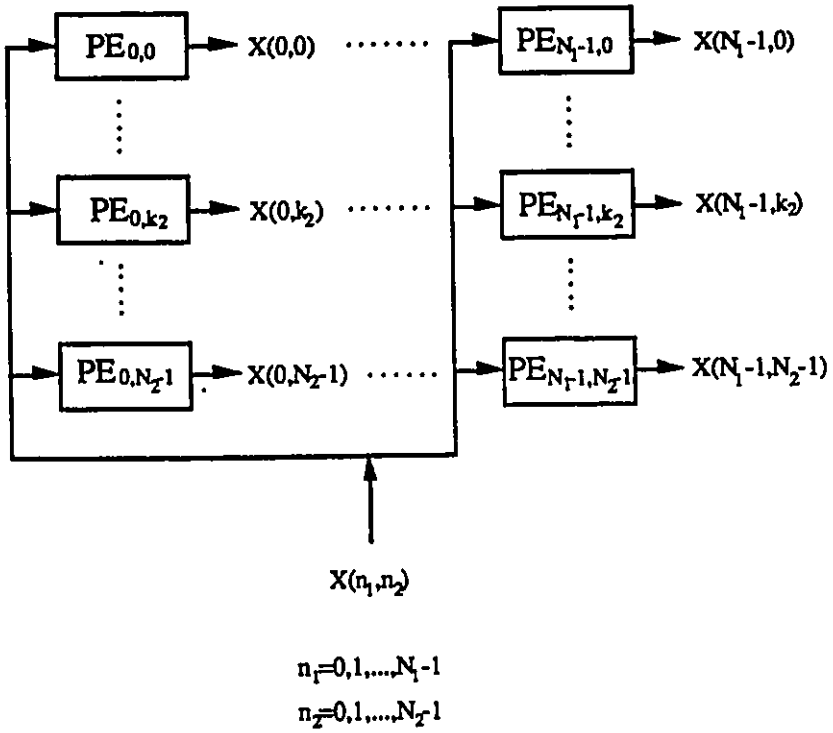

Figure 2.13: Zhu's array for 2-D DFT

The throughput is one 2-D transform every N^2 clock cycles.

2.3.2.4 Chen's 2-D array

Chen proposed a 2-D DFT array in [13]. The structure uses the modified Goertzel algorithm to realize the two 1-D N-point DFT and compute the 2-D DFT in row-column wise. The row-column transposition is not needed.

Figure 2.14 depicts the array. Each row and column is the same as Beraldin's model (section 2.3.1.6) with some modifications. The row transforms are performed by the corresponding rows of the input sequence. The intermediate results are pipelined upward and then reenter the array for the column computation. The final DFT samples leave the array at the bottom, with one delay between successive column.

This array is composed of N^2 PEs, each double the complexity of that of Beraldin's model. The input data enter the array in parallel and no preloading is necessary. The coefficients W^k are static for both the row and the column computations so ROM's can be used in place of multipliers. The throughput is one 2-D transform every N cycles and the computation time is $5N - 2$ cycles.

2.3.3 Discussion

The characteristics and performance of the seven 1-D DFT arrays and four 2-D DFT arrays are given in table 2.1 and 2.2, respectively. The results here were mentioned in the previous sections. All the structures are described in the tables using the following entries:

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

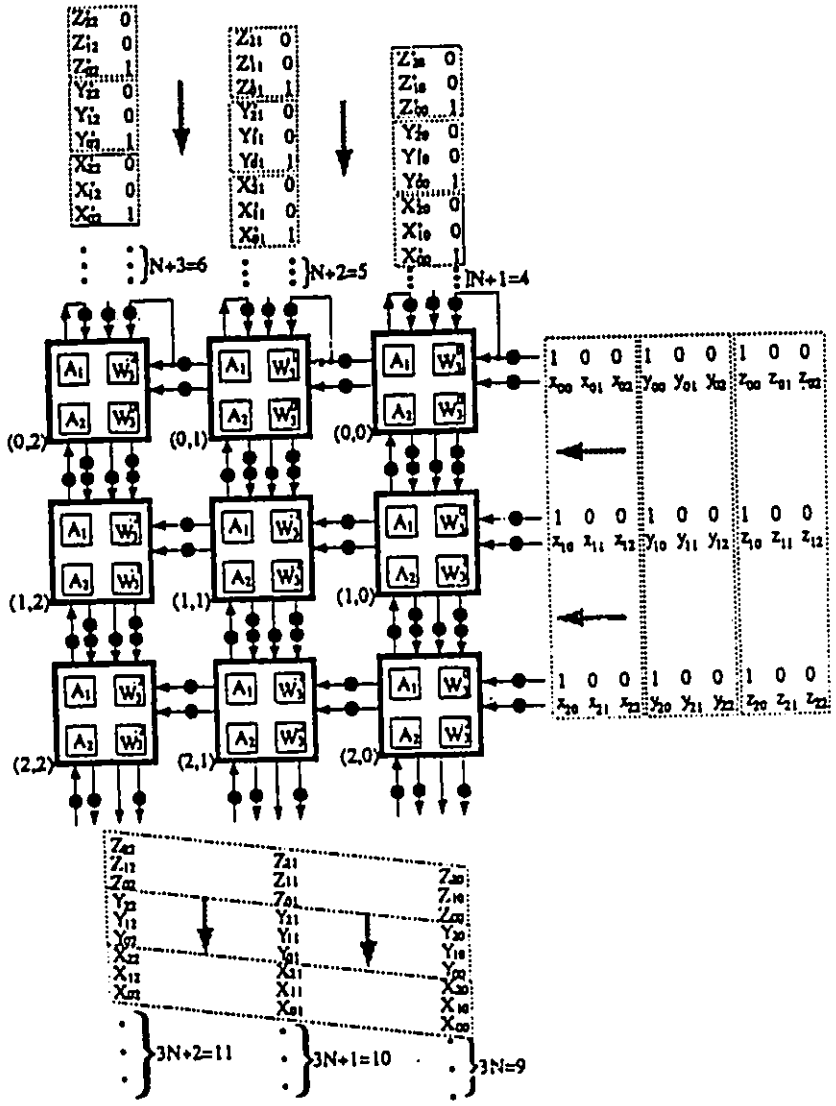

Figure 2.14: Chen's array for 2-D DFT

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

3. Computation time: the number of clock cycles required to calculate a transform from the first data sample to the last DFT sample to exit the array, where the clock cycle is the time delay introduced by one PE.

4. Throughput rate: the number of clock cycles required to calculate successive transforms.

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

6. Communication complexity: type of links needed for exchanging information between the PEs.

7. Row-column transposition: (for 2-D DFT only) if the row-column transposition is needed between the two 1-D DFT stages.

Structure	#PE	PE add.	PE mult.	Comp.	Thrpt	Memory	Comm.
Matrix-m	N^2	4	4*	$2N-1$	1	Note 1	Note 3
Matrix-v	N	4	4	$3N-1$	$3N-1$	Note 1	Note 3
pipeline	N	4	4	$2N-1$	N	Note 2	Note 3
Kung	$N-1$	4	4	$3N-3$	$3N-3$	Note 2	Note 3
Chang	N	4	4	$2N-1$	N	Note 2	Note 3
Beraldin1	N	4	4*	$2N-1$	N	Note 2	Note 4
Beraldin2	N	4	4*	$3N-2$	N	Note 2	Note 3

Table 2.1: Comparison of the 1-D DFT systolic arrays

Notes:

1: Memory large enough to store/retrieve data/results.

2: Very low memory requirement.

3: Local type communications between the PEs.

4: Global communications required for data transfer.

5: The row-column transposition needed(only applies to 2-D DFT).

6: The row-column transposition not required(only applies to 2-D DFT).

$*$: Static twiddle factors used.

Structure	#PE	PE add.	PE mult.	Comp.	Thrpt	Memory	Comm.	R/C
R/C transp	$2N^2$	4	4	$4N-4$	N	Note 1	Note 4	Note 5
Chang	N^2	8	8	$4N-2$	$4N-2$	Note 1	Note 3	Note 6
Zhu	N^2	4	8*	N^2	N^2	Note 1	Note 4	Note 6
Chen	N^2	8	8*	$5N-2$	N	Note 2	Note 3	Note 6

Table 2.2: Comparison of the 2-D DFT systolic arrays

For the 1-D DFT, all the arrays have the same complexity. The 2-D matrix-matrix multiplication structure has the highest throughput of one N-point transform per clock cycle but has N^2 PEs, N times larger than the other models. For the matrix-matrix multiplication and Beraldin's two structures, ROM's can be used in place of multipliers to improve accuracy and save area. Most structures can process continuous data sequences except the matrix-vector multiplication and Kung's structure since they need preloading time. In terms of memory requirement, the matrix-matrix and matrix-vector multiplication structures need fairly large memory to store/retrieve data/results. The first of Beraldin's two structures uses a data bus which requires global communications.

Above all, for a moderate throughput rate, Chang's, the pipeline and Beraldin's structures can all work well. The pure-systolic array of Beraldin's is the best among them in terms of hardware complexity and communication requirement. To achieve a higher throughput rate, the matrix-matrix multiplication structure is a good choice if much larger chip area is acceptable.

For the 2-D DFT, all the arrays have similar complexity in terms of number of PEs and the PE complexity. With the modified Goertzel algorithm, both Zhu's and Chen's structures can use ROM's in place of multipliers. The comparatively longer computation time of Zhu's structure is due to the serial input. The throughput rate for Chang's and Zhu's arrays are not as high as the R/C transposition structure and Chen's. Except Chen's array, all the others need buffers to pre-arrange the input data. Zhu's need global communication which may not be suitable for the VLSI implementation. Chang's, Zhu's and Chen's all managed to eliminate the row-column transposition. However, they lose the flexibility that can be used for either 1-D DFT or 2-D DFT. Since the row transforms and the column transforms of the R/C transposition structure can be built on two identical chips, therefore, keeping the flexibility that can be used for both the 1-D and 2-D DFT.

Overall, Chen's structure outperforms the other three in terms of hardware complexity and communication requirement. If splitting an array onto separate chips is considered, R/C transposition structure would be a good choice because both the row computation part and the column computation part can stand alone.

2.4 Summary

In this chapter, the 1-D and 2-D DFT were introduced and different algorithms were described, emphasizing the Goertzel algorithm. The computation complexity and hardware requirements were also given. Then existing systolic array architectures were surveyed, for the 1-D DFT and 2-D DFT, respectively. Finally, these array structures were compared and analyzed.

Chapter 3

Systolic Array Implementation of the 1-D and the 2-D DFT

3.1 Introduction

The Discrete Fourier Transform is one of the digital signal processing algorithms which are given special attention for systolic array implementation. In chapter 2, we have studied different 1-D and 2-D DFT algorithms. Based on these algorithms, existing systolic array structures were surveyed and analyzed. In this chapter, we will propose several structures using the second-order Goertzel algorithm. For the 1-D DFT, two 1-D and one 2-D systolic arrays are proposed, and one 2-D structure for the 2-D DFT. These arrays yield systems with minimal complexity as well as high throughput rates. The performance analysis is discussed in chapter 4.

3.2 Proposed systolic arrays for 1-D DFT using the second-order Goertzel algorithm

In this section, three new and efficient systolic array realizations for the 1-D DFT are introduced, two 1-D structures and one 2-D structure. Most PEs of the arrays require only real computations and the only complex computation is performed at the last

stage. The structures are mapped from the recursive equations of the second-order Goertzel algorithm. Two 1-D arrays and one 2-D array are presented in section 3.2.1 and 3.2.2, respectively.

3.2.1 1-D systolic array implementation of the 1-D DFT

The second-order Goertzel algorithm reduces the number of real multiplications by a factor of 2 compared to the first-order Goertzel algorithm or the direct computation. To map the DFT onto a systolic array using the second-order Goertzel algorithm, special arrangement is made to incorporate the above property besides modularity, pipelining, and local communication. Two 1-D structures, the semi- and the pure-systolic arrays, are now introduced.

3.3.3.1 Semi-systolic array

Figure 3.1 shows the semi-systolic array for a 3-point DFT, based on the second-order Goertzel algorithm. The array consists $N = 3$ PE-I's (PE#0, PE#1 and PE#2) and 1 PE-II (PE#3). The data samples flow through the three PE-I's in a pipeline fashion. Each data sample is used by a PE-I for the computation of Eq.(2.25), of one DFT sample. The data samples are shifted out to their nearest neighbor. A new data sample is therefore available at each PE-I. When the intermediate results are computed in the PE-I's, they are passed to the PE-II for the computation of Eq.(2.26), which is shared by the 3 PE-I's. One frame of three input samples {x(0), x(1), x(2)} is used to show the process. The PE-I's outputs the intermediate results $y_k(N-1)$ and $y_k(N-2)$ ($y_k(2)$ and $y_k(1)$ for $N = 3$) from the PE-I's on buses, one per clock cycle, as inputs of PE-II. After the last stage of computation in PE-II, the DFT samples leave the array

one per clock cycle.

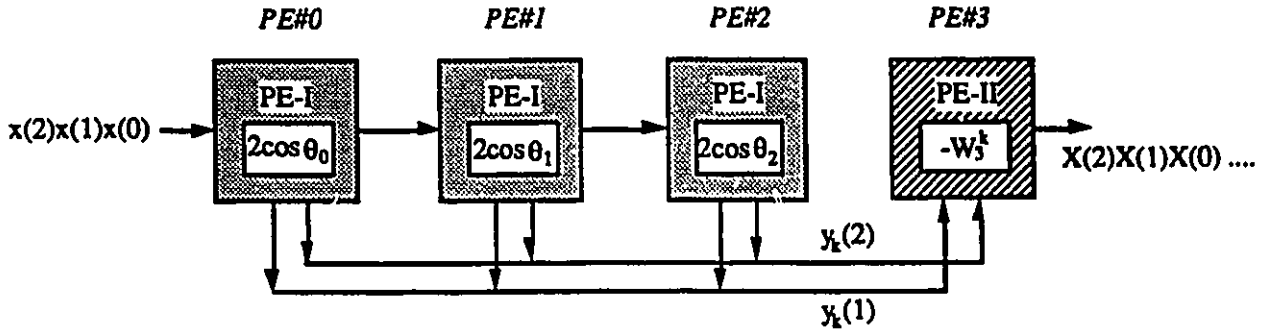

Figure 3.1: 1-D semi-systolic array for 1-D DFT, $N = 3$

Only one static factor is stored per PE-I in the array. Each PE-I contains one fixed coefficient $2cos\theta_k$. The factor $-W_N^k$ in PE-II is not static since it is shared by PE#0, PE#1 and PE#2, and changes with different k.

The semi-systolic array for a three-point DFT at successive clock cycles are shown in Figure 3.2. At each PE-I, the recursive equation (2.25) is performed three times. Then, the intermediate results are available at the PE outputs and passed to PE-II by way of buses. The PE-I's latches in the recursive path are cleared, thus making the PE-I ready for the computation of the subsequent 3-point vector. Three frames composed of three samples each are used to show the process of computing a DFT: $\{x(0), x(1), x(2)\}$, $\{x'(0), x'(1), x'(2)\}$ and $\{x''(0), x''(1), x''(2)\}$. The intermediate results from PE-I's are $\{y(0), y(1), y(2)\}$, $\{y'(0), y'(1), y'(2)\}$ and $\{y''(0), y''(1), y''(2)\}$. The resulting DFT vectors are $\{X(0), X(1), X(2)\}$, $\{X'(0), X'(1), X'(2)\}$ and $\{X''(0), X''(1), X''(2)\}$. The first input sample $x(0)$ enters the array at $t = 0$, and the first output comes out of the array at $t = 5$.

Figure 3.3 shows details of PE-I for the semi-systolic array. At each clock cycle($clk1$),

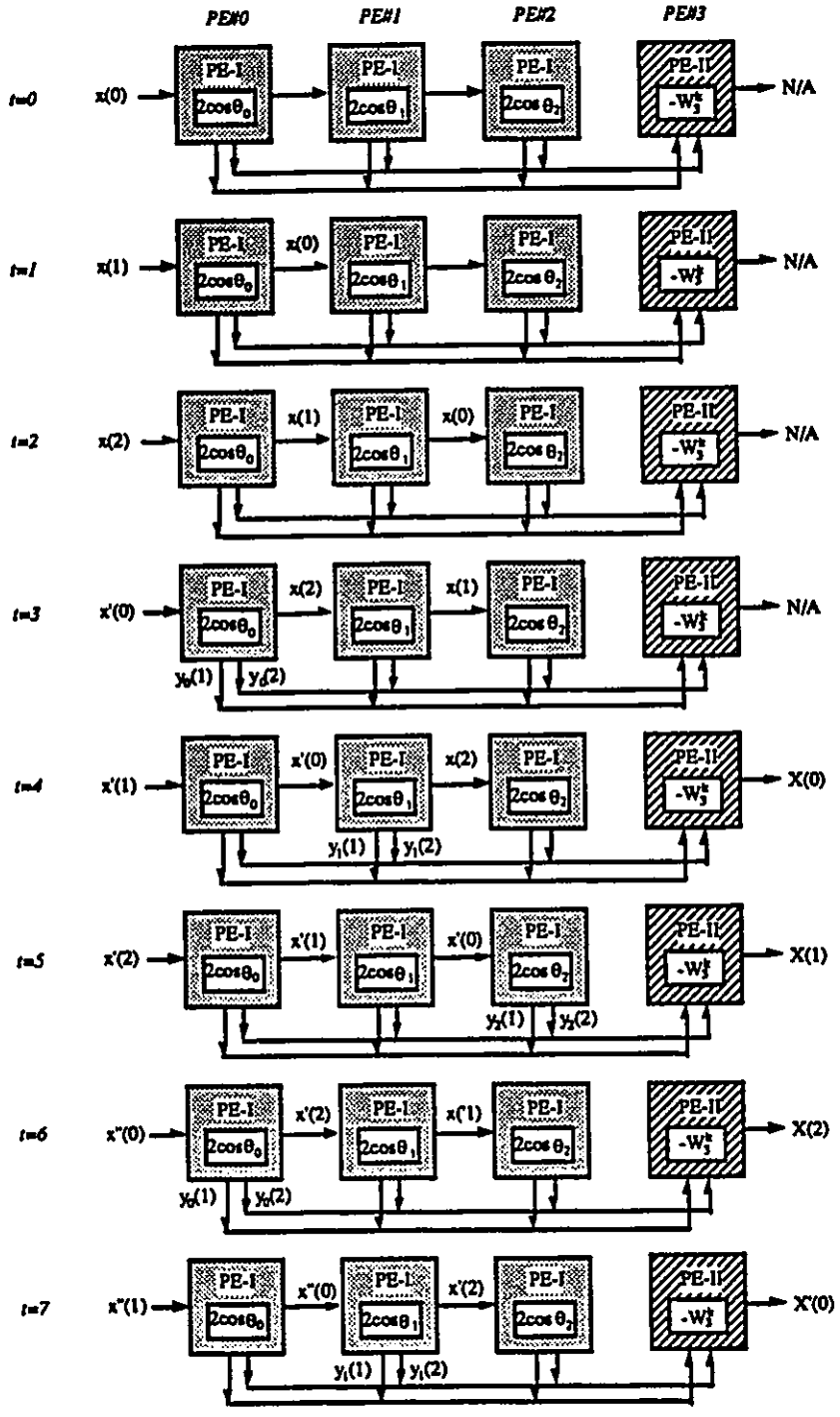

Figure 3.2: Semi-systolic array at successive clock cycles, $N = 3$

one input sample $x(n)$ enters the adder in the PE. When 4 real additions, 2 multiplications and 2 negations are completed, the partial result $y_k(n)$ is obtained and the PE is ready for the next input sample. After N input samples have entered the PE and N operations, the accumulated results are $y_k(N-1)$ and $y_k(N-2)$. Passing through the D-type flip-flops D_1 controlled by a clock signal $clk2$, $y_k(N-1)$ and $y_k(N-2)$ are available on the data buses, which connect all the PE-I's to the PE-II. At the same time, latches D_2 are reset by $clk2$ and the PE is ready to compute a new sequence of input samples.

The PE-II is shown in detail in Figure 3.4. At each clock cycle($clk1$), the $y_k(N-1)$ and $y_k(N-2)$ signals, which come from the PE-I's enter latches D_1. After a complex multiplication and a complex addition, the DFT sample $X(k)$ is available at the next clock $clk1$.

$clk1$ has the maximum clock frequency of the chip and synchronizes all the events for the systolic array. It is set by the delay time of the adder, the multiplier and the latch,

$$f_{max} = \frac{1}{T_{latch} + T_{multiplier} + 2T_{adder}} \tag{3.1}$$

$clk2$ has a frequency that is N times of that of $clk1$ and pipelined through the PE-I's. Figure 3.5 shows the $clk1$ and $clk2$ signals as seen by all the PE's of the array.

Here, buses are used to pass the intermediate results. The semi-systolic array is only good for short-length transforms. The limit on the transform length is affected by the loading capacity or delay of the bus. For long transforms, the pure-systolic array can be used, which yields local communication.

For an N-point DFT transform, N PE-I's containing one static factor each and 1 PE-II with one changing factor are required for this semi-systolic array. The hardware

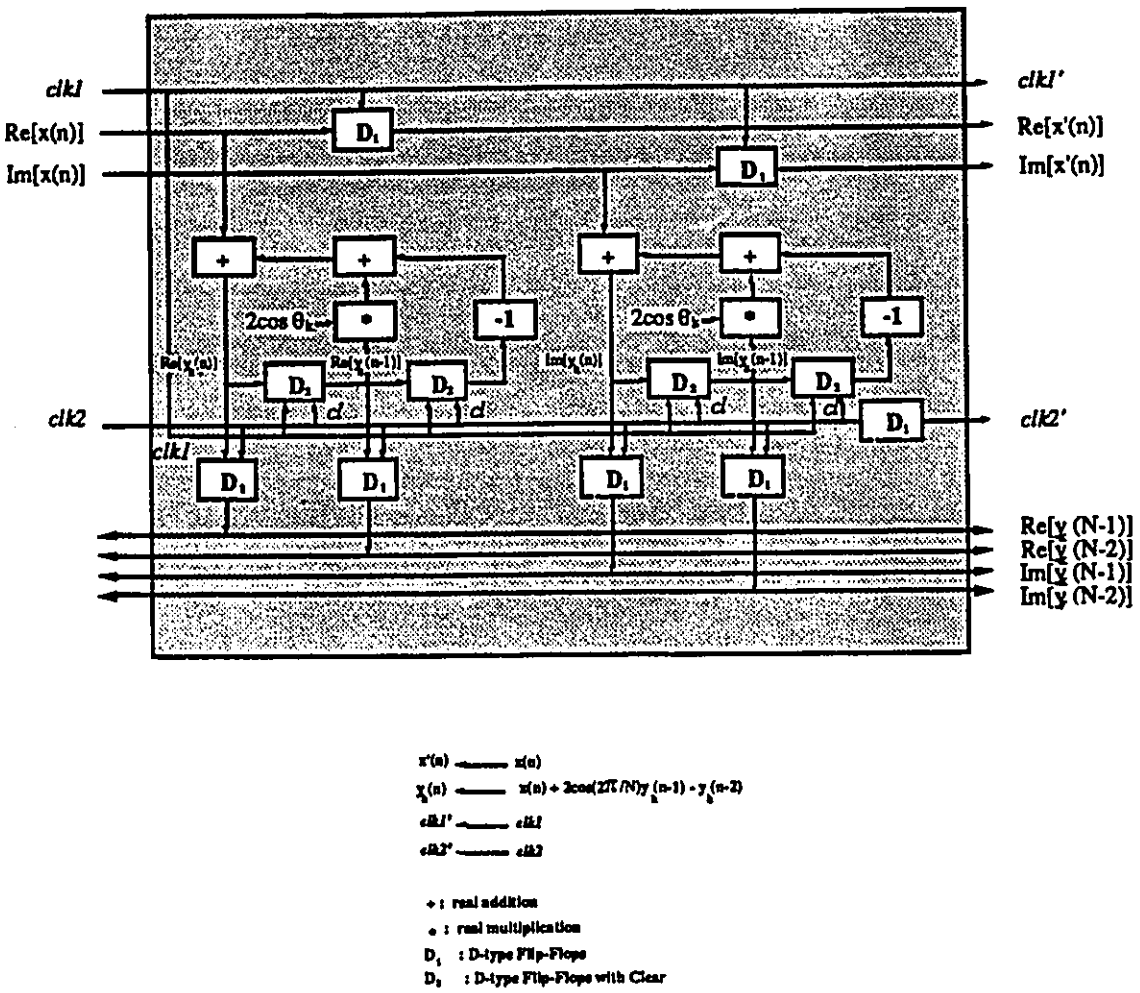

Figure 3.3: Details of PE-I for semi-systolic array

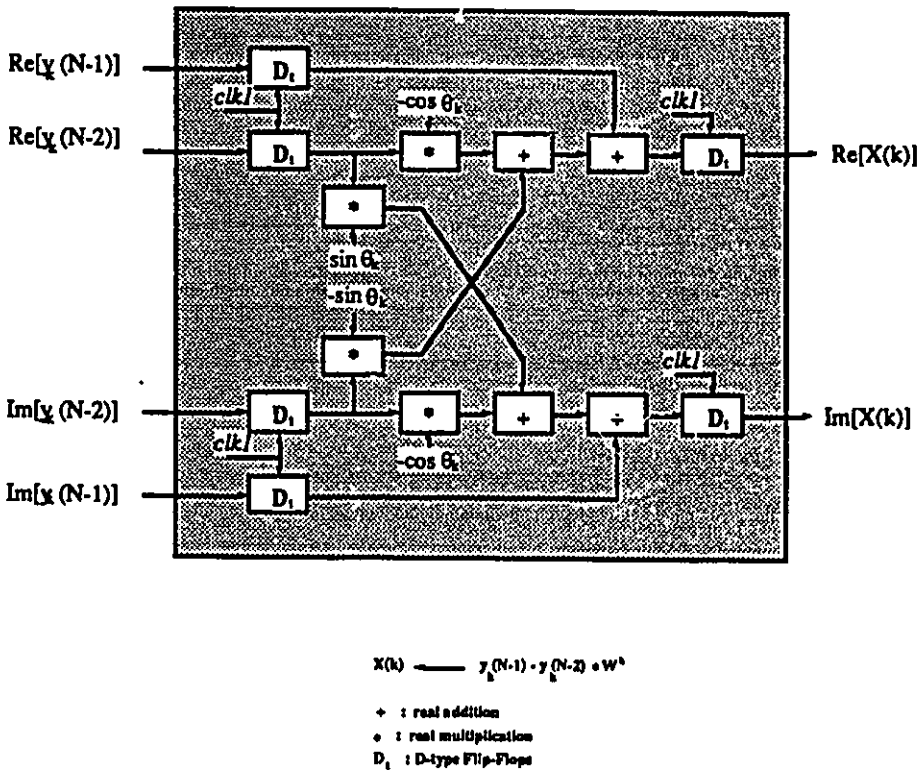

Figure 3.4: Details of PE-II for semi- or pure-systolic array

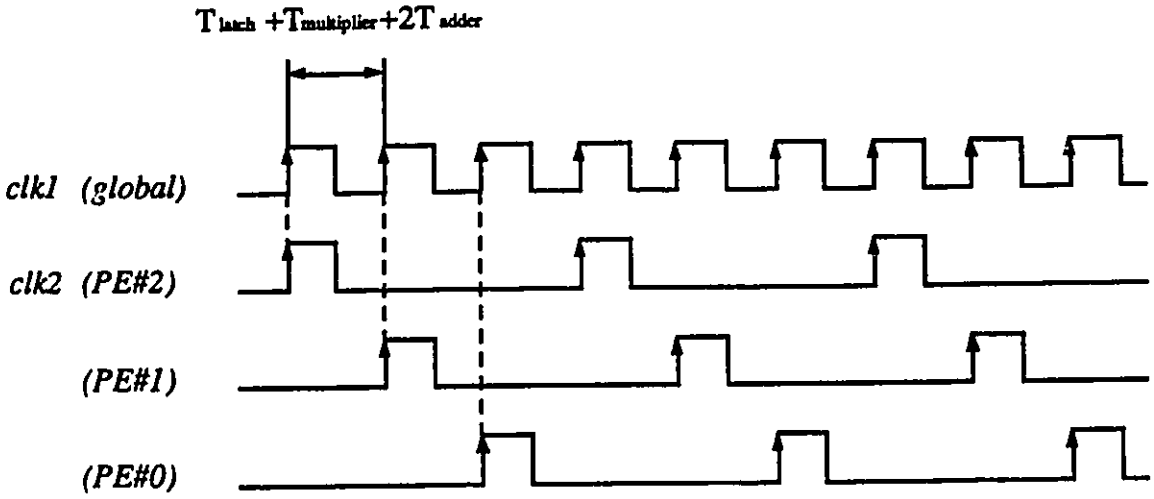

Figure 3.5: Timing diagram for the semi-systolic array

complexity for the array is $2(N+2)$ multipliers (most can be implemented by stored-product ROM's), and $4(N+1)$ adders. The computation time is $2N+1$ clock cycles. Successive frames of input data are computed every N clock cycles.

3.3.3.2 Pure-systolic array

It is possible to avoid the buses found in the semi-systolic array by adding multiplexors and D_1 latches, except in the last PE-I. Figure 3.6 shows the pure-systolic array resulting from this modification for $N = 3$. The three PE-I's (PE#2, PE#1 and PE#0) contain the static factors of $2cos\theta_2$, $2cos\theta_1$ and $2cos\theta_0$ while the PE-II (PE#3) contains the factor of $-W_N^k$, which changes with k. The input samples are {x(0), x(1), x(2)}, the intermediate results $y_k(2)$ and $y_k(1)$, and the outputs {X(0), X(1), X(2)}.

The data samples flow through the array in a pipeline fashion, similar to that of Figure 3.2. The difference with the semi-systolic array is the way the intermediate results from the PE-I's are passed to the next computation stage in PE-II. $y_k(2)$ and

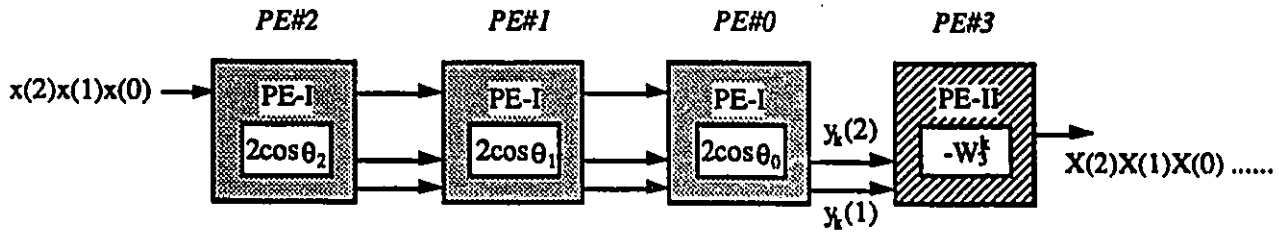

Figure 3.6: 1-D pure-systolic array for 1-D DFT, $N = 3$

$y_k(1)$ are available in PE#k ($k = 0, 1, 2$) at different time but are kept within the PE by the latches which also keep the stored results from interfering with the ones that are being computed for the next transform. When the intermediate results of all the PE-I's are computed, a global clock controls the multiplexors of that PE to start pipelining out the stored $y_k(2)$ and $y_k(1)$ of that PE, to the PE-II (PE#3) at the same time. Figure 3.7 shows the pure-systolic array for a three-point DFT at successive clock cycles. The first input data $x(0)$ enters PE#2 at $t = 0$, and the first output $X(0)$ leaves PE#3 at $t = 6$.

Figure 3.8 depicts the details of PE-I for the pure-systolic array. Figure 3.9 is the last PE-I in the array with multiplexors but no latches D_1 following these. The computation part of both of the PE's is the same as Figure 3.3. When the intermediate results are available, $y_k(n)$ and $y_k(n-1)$ enter the two-to-one multiplexors MLX, instead of latches D_1 in Figure 3.3. With a control signal $clk3$, MLX selects either the results of the PE-I or those of other PE-I's, to go through. The PE-II of this pure-systolic array is the same as in the semi-systolic array, shown in ..ure 3.4.

$clk1$ has the maximum clock frequency of the chip and $clk2$ has a frequency that is N times of that of $clk1$ as for the semi-systolic array. $clk3$ is a broadcast signal that

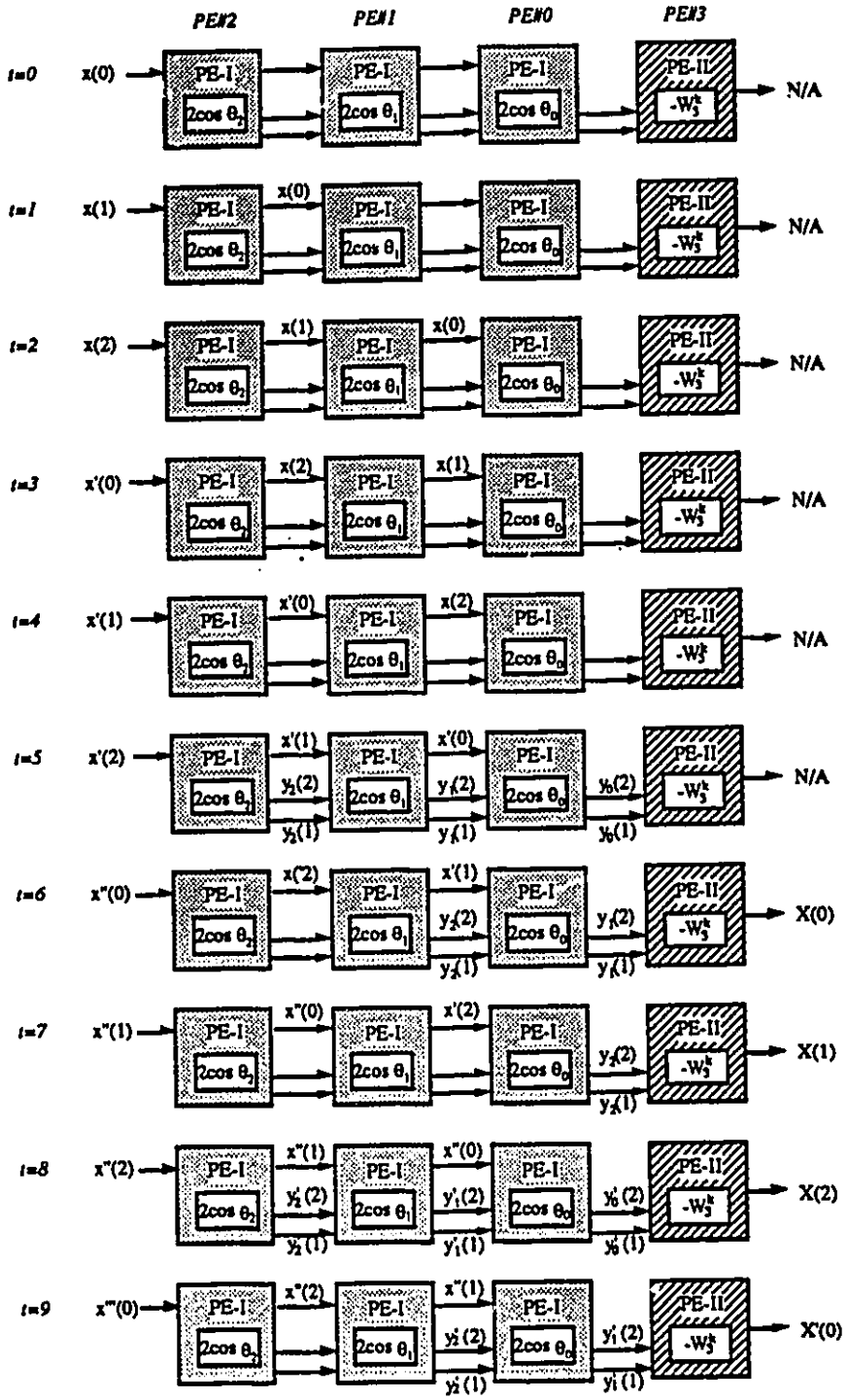

Figure 3.7: Pure-systolic array at successive clock cycles, $N = 3$

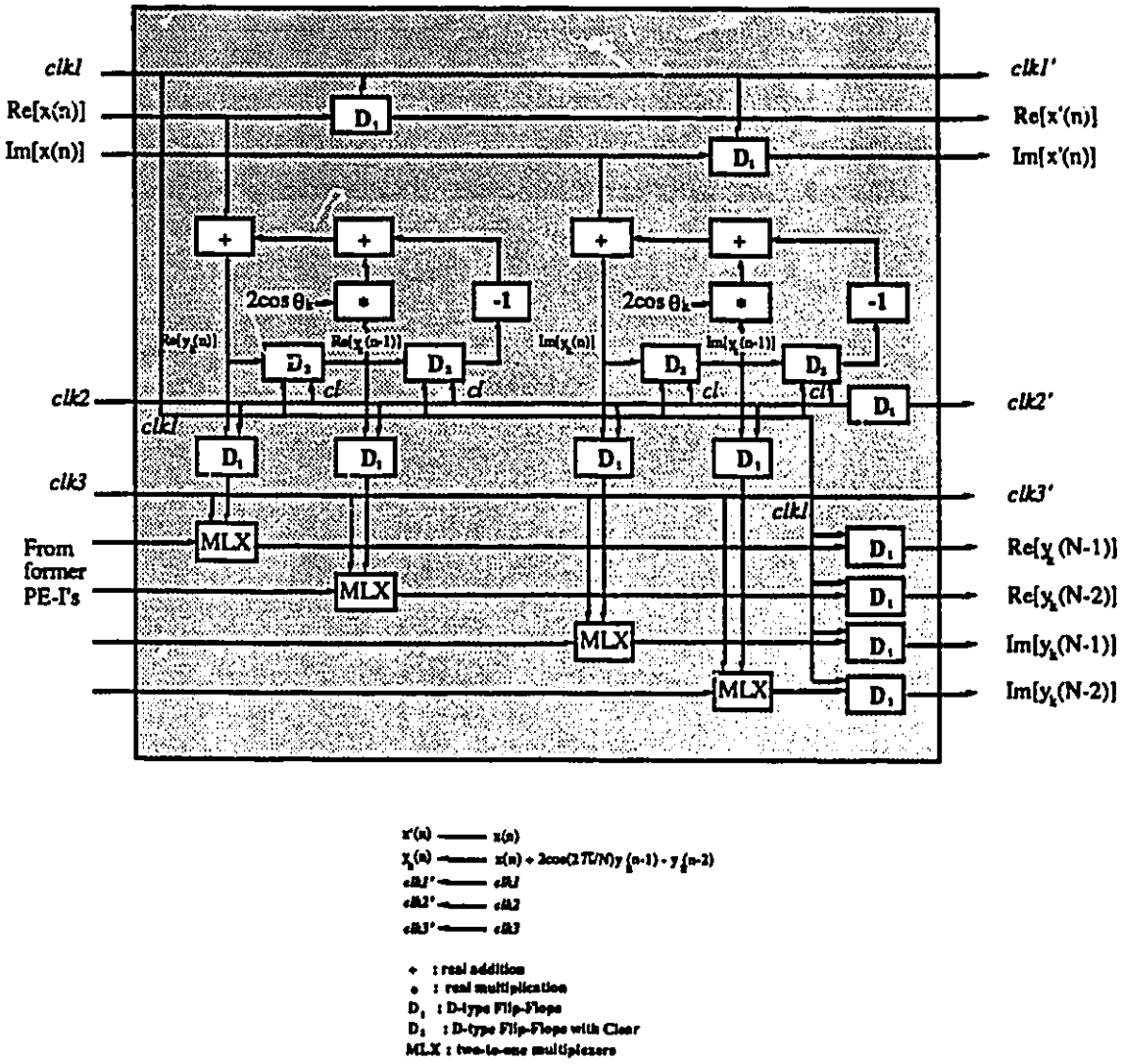

Figure 3.8: Details of PE-I for pure-systolic array

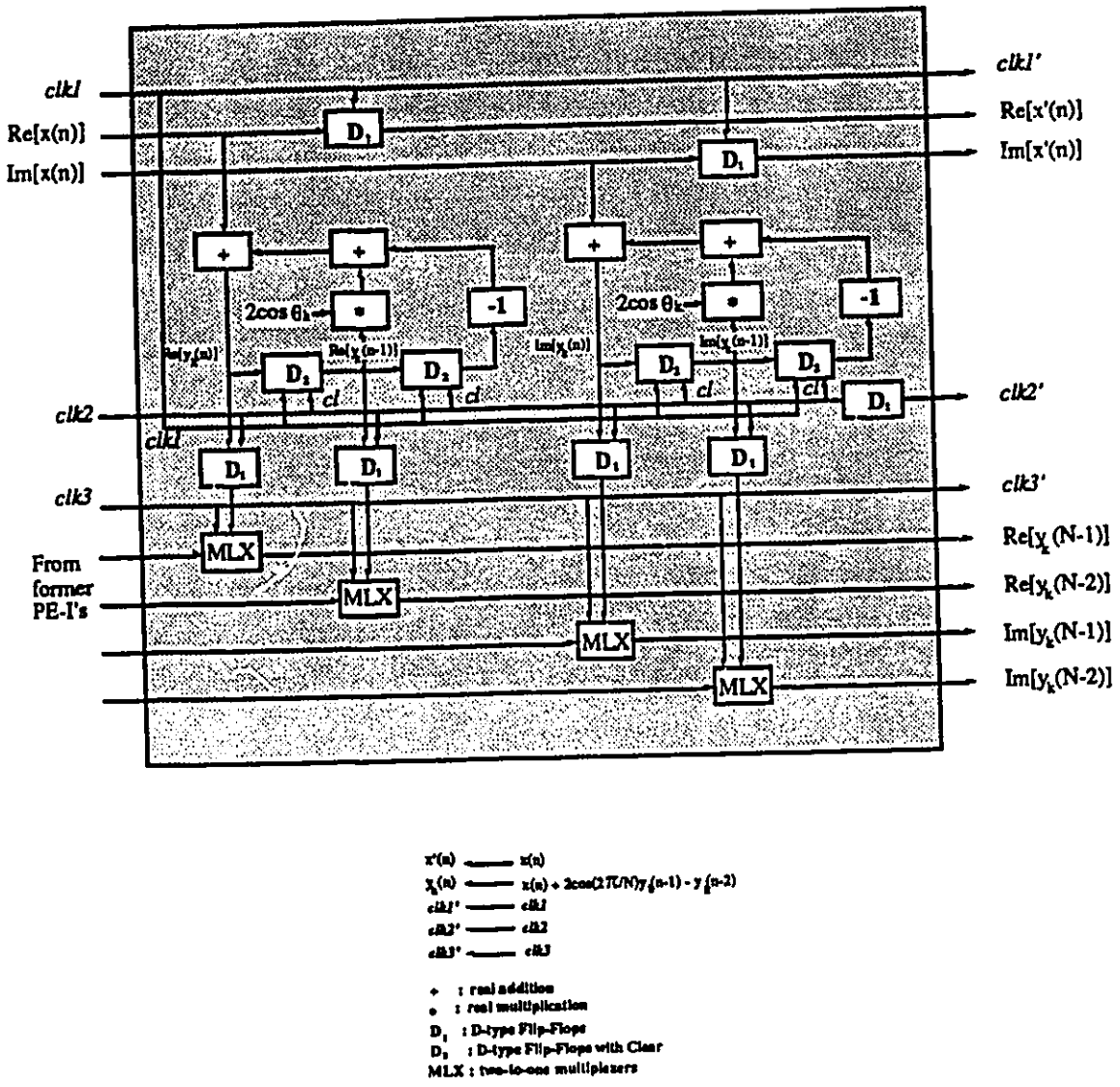

Figure 3.9: Details of the last PE-I for pure-systolic array (without delay following the multiplexors)

selects one direction of the data to go through. Figure 3.10 shows the $clk1$, $clk2$ and $clk3$ signals at each PE.

The hardware complexity for the pure-systolic array is about the same as the semi-systolic array. For an N-point DFT transform, computation time of this array is $3N$ clock cycles. Successive frames of input data are computed every N clock cycles. The throughput is 1 DFT sample per clock cycle.

3.2.2 2-D systolic array implementation of the 1-D DFT

A 2-D systolic array structure is shown in Figure 3.11 which accommodates N parallel inputs at a time so that a higher throughput of one N-point DFT per cycle is achieved. The number of processing elements is, however, N times as large as for the 1-D systolic array structure.

The 2-D array in Figure 3.11 is composed of rows of PE-I's which is shown by its static coefficients $2cos\theta_k$. Each row of the array computes the intermediate results for one frame of input samples as in Figure 3.2. Three frames composed of three samples each enter the 2-D array as $\{x_0, x_1, x_2\}$, $\{x_0', x_1', x_2'\}$ and $\{x_0'', x_1'', x_2''\}$ and flow through one row of PE-I's.

After the three input samples have entered one column of PE-I's, the intermediate results are computed and available at the PE-II shown as $-W^k$, where the computation of Eq.(2.26) is performed in sequence, sharing the same hardware. The DFT samples $\{X_0, X_1, X_2\}$, $\{X_0', X_1', X_2'\}$ and $\{X_0'', X_1'', X_2''\}$ leave the PE-II's with one delay between each column.

The PE-I's for the 2-D systolic array in Figure 3.11 is similar to those for the 1-D array in Figure 3.6 with some modifications. Details are shown in Figure 3.12. Due to the fact that the results of all the columns are pipelined out from the last row of PE-I's,

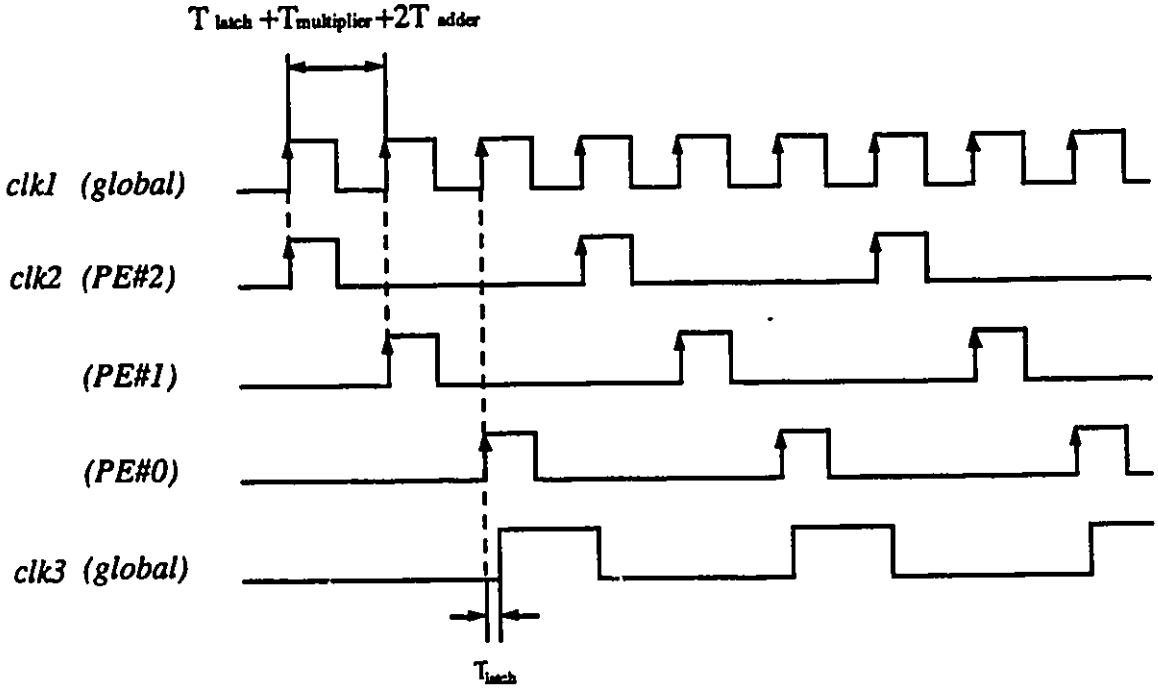

Figure 3.10: Timing diagram for the pure-systolic array

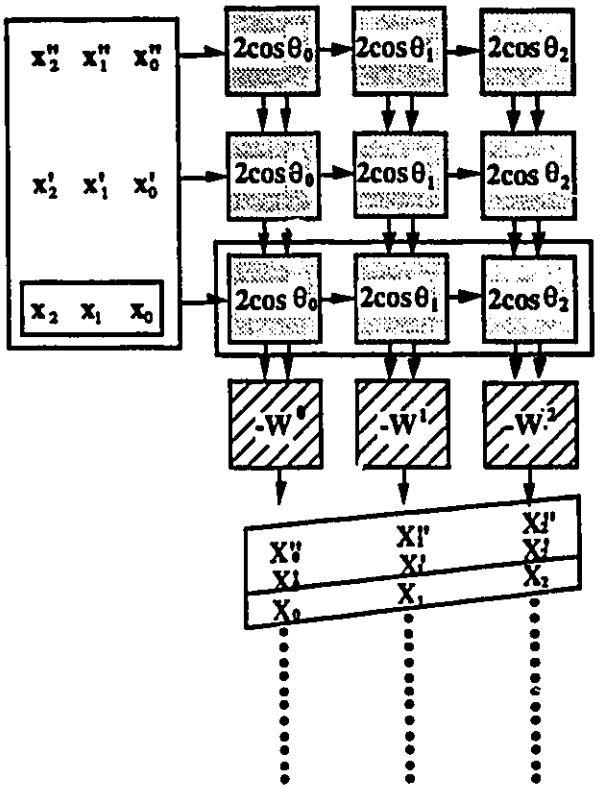

Figure 3.11: 2-D systolic array for 1-D DFT using the 2nd-order Goertzel algorithm

two-to-one multiplexors are used before the flip-flops D_1. With the control signal $clk2$, the multiplexor selects either the intermediate results $y_k(N-1)$ and $y_k(N-2)$ of the PE, or the $y_k(N-1)$ and $y_k(N-2)$ signals called y', y'_{-1} from the other PE-I's of the column, to go through. Then passing the D_1, controlled by $clk1$, the computed intermediate results of this PE and the former ones come out as y', y'_{-1}, in sequence. $clk2$ is N times the frequency of $clk1$ as in Figure 3.5. The 2-D structure needs N^2 PE-I's and N PE-II's. The computation time is $5N+1$.

3.3 Proposed systolic array for 2-D DFT using the second-order Goertzel algorithm

A 2-D DFT is generally realized using two 1-D DFT arrays where a row-column transposition is needed between the stages. The two 1-D DFT arrays are identical and can each be used for either a 1-D or a 2-D DFT. The structures can be arranged so that the row-column transposition is not required. Several systolic arrays for the 2-D DFT have been reviewed in chapter 2.3.2. In this section, a new efficient systolic array structure for the 2-D DFT is presented. Based on the second-order Goertzel algorithm, this system not only avoids the row-column transposition but also keeps the flexibility of being used for either a 1-D or a 2-D DFT. Some delay units are needed between the two stages as shown in Figure 3.13. High computation speed is achieved as well as minimal complexity of the system.

At first, the second-order Goertzel algorithm for the 2-D DFT is introduced[7]. Its recursive form is then mapped to the 2-D systolic array. Like its counterpart for the 1-D DFT, most PE's of the array require only real computations and the only complex computation is performed at the last stage of the row and column computations of the

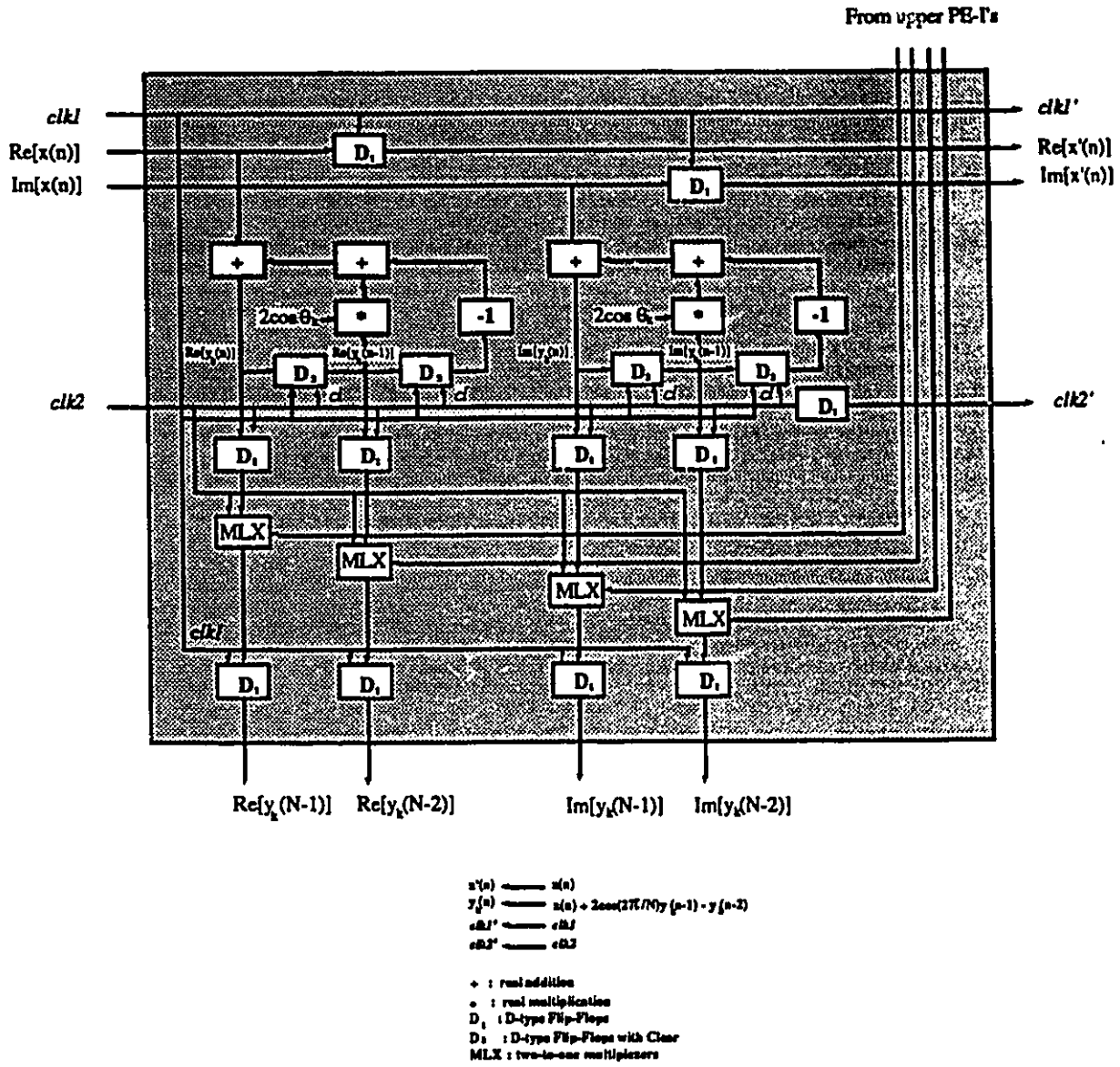

Figure 3.12: Details of PE-I of 2-D systolic array for 1-D DFT

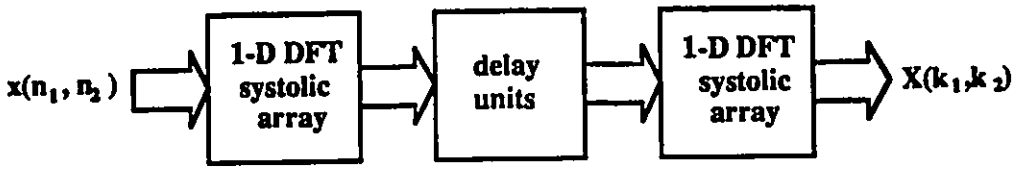

Figure 3.13: System diagram of the proposed array for 2-D DFT

2-D transform. Therefore, the structure has the advantage of significantly reduced PE complexity.

3.3.1 The second-order Goertzel algorithm for the 2-D DFT

Extending the Goertzel algorithm from the 1-D DFT to the 2-D DFT, we get the following recursive form of Eq.(2.29) for $n_1 = 0, 1, 2, ..., N_1 - 1$,

$$y(k_1, n_1, n_2) = x(n_1, n_2) + 2cos(2\pi k_1/N)y(k_1, n_1 - 1, n_2) - y(k_1, n_1 - 2, n_2) \quad (3.2)$$

After the N_1th iteration, $y(k_1, N_1 - 1, n_2)$ and $y(k_1, N_1 - 2, n_2)$ are available. The intermediate result after the row computation is obtained by one extra step:

$$Y(k_1, n_2) = y(k_1, N_1 - 1, n_2) - W_N^{k_1} y(k_1, N_1 - 2, n_2) \quad (3.3)$$

For Eq.(2.30), $n_2 = 0, 1, 2, ..., N_2 - 1$,

$$z(k_1, n_2, k_2) = Y(k_1, n_2) + 2cos(2\pi k_2/N)z(k_1, n_2 - 1, k_2) - z(k_1, n_2 - 2, k_2) \quad (3.4)$$

After the N_2th iteration, $z(k_1, k_2, N_2 - 1)$ and $z(k_1, k_2, N_2 - 2)$ are obtained. The 2-D DFT sample $X(k_1, k_2)$ is completed by:

$$X(k_1, k_2) = z(k_1, N_2 - 1, k_2) - W_N^{k_2} z(k_1, N_2 - 2, k_2) \quad (3.5)$$

Here, $y(k_1, n_1, n_2)$ indicates the k_1, n_2th DFT sample at the n_1th step for the row computation of the 2-D DFT. $z(k_1, n_2, k_2)$ represents the k_1, k_2th DFT sample at the n_2th step for the column computation of the 2-D DFT. We suppose $y(k_1, -1, n_2) = y(k_1, -2, n_2) = 0$ and $z(k_1, -1, k_2) = z(k_1, -2, k_2) = 0$.

3.3.2 2-D systolic array implementation of 2-D DFT

The second-order Goertzel algorithm reduces the number of arithmetic operations and can be mapped onto systolic array architectures efficiently. For the 1-D DFT, several proposed structures were presented in detail in 3.2. Here, an efficient 2-D systolic array is introduced for the 2-D DFT.

Our 2-D DFT systolic array uses two identical 1-D DFT arrays, each accomplishing either the row or the column computation. The usual row-column transposition is avoided by the special arrangement of the data flow after the first 1-D DFT computation.

A 2-D array for $N_1 = N_2 = 3$ is shown in Figure 3.14. The structure consists of three parts: the row computation part on the left hand side, the delay units in the middle and the column computation on the right hand side. The 2-D structure for the 1-D DFT (Figure 3.11) introduced in section 3.2.2 is used for both the row computation part and the column computation part. The input data x_{n_1,n_2} (here, $n_1, n_2 = 0, 1, 2$) enter the array with columns in parallel for the row computation. The three 1-D arrays, each consisting of three PE-I's in a column, process that particular column of input data. The intermediate results from the three 1-D arrays are pipelined out in rows to the PE-II on right hand side. The results after this row computation are Y_{k_1,n_2}. After some delay units, Y_{k_1,n_2} enter the other 2-D array in rows for the column computation. The resulting 2-D DFT samples X_{k_1,k_2} emerge finally.

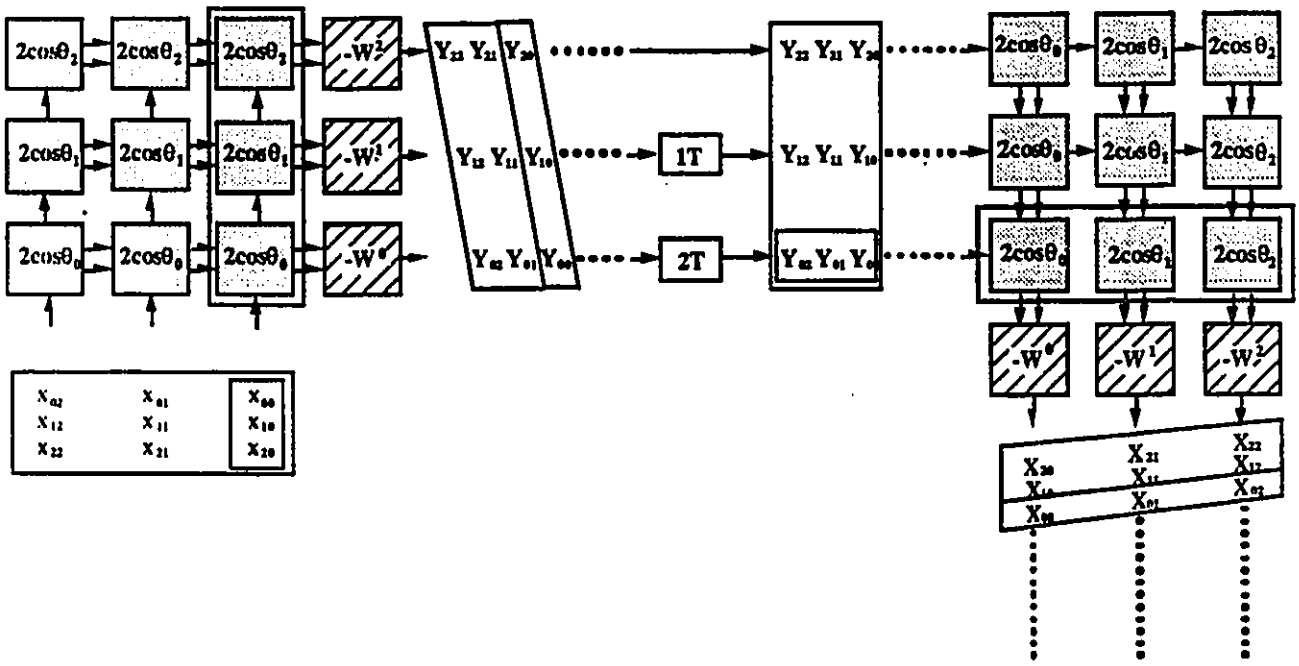

Figure 3.14: 2-D systoiic array for 2-D DFT

We have shown before that the 2-D array that is used for both the row and column computation here, has the advantages of no prearrangement of input data, high throughput rate and low complexity. It has another special property - the direction of the data flow changes from vertical to horizontal. The input data for the row computation enter the array in columns and are computed in a column of PE-I's. The intermediate results, however, are pipelined out in rows and share the second part of the row computation in the PE-II of that row. This same arrangement accomplishes the row-column transposition that otherwise would be necessary before the second column computation stage.

The delays are necessary here. We require that the input samples $x(n_1, 0)$ for the row computations are entered into the PEs at the same time. After the row computation, the outputs $Y(k_1, n_2)$ emerge from the PEs at different times (one cycle delay between each column of the PEs). We put $N-1$ delay units at the end of the first column, $N-2$ units for the second column, $\cdots$, 1 unit for the $(N-2)th$ column and no delays for the $(N-1)th$ column. The delay units introduced between the two stages guarantee that the input samples $Y(0, n_2)$ for the column computations are entered in the PEs simultaneously. The final 2-D DFT samples leave the PE array with one cycle delay between each row.

The PE-I and PE-II for the row and column computations are the same as those for the 2-D systolic array for 1-D DFT. They are shown in Figures 3.12 and 3.4, respectively.

There are some advantages for this structure. First, based on the second-order Goertzel algorithm, it reduces the number of real multiplications by a factor of 2. Second, the inputs are in the natural order so no prearrangement of the data is necessary and the output samples appear in natural order as well. Third, the transposition is

eliminated between the row and column transforms while some delay units are needed. Fourth, the static coefficients require a minimal area and I/O lines for the coefficients and stored-product ROM's can be used in place of multipliers. Further more, the two systolic arrays for the row and the column computations are exactly the same and can be realized on two identical chips. The same chip can be used for either the 1-D or the 2-D DFT computations.

The 2-D array has a total number of $2N^2$ PE-I's and $2N$ PE-II's. The first output sample emerges $4N+2$ cycles after the first input samples enter into the systolic array. The number of cycles needed to compute the 2-D DFT with this model is $5N+1$. And the throughput of the system is one 2-D transform every N cycles.

3.4 Discussion

In this chapter, several structures have been introduced for the 1-D DFT and the 2-D DFT. The characteristics of these arrays are shown in table 3.1 and table 3.2. The entries of the table are the same as those in section 2.3.3 which pertain to existing arrays.

Notes:

1: Memory large enough to store/retrieve data/results.

2: Very low memory requirement.

3: Local type communication between the PEs.

4: Global communication required for data transfer.

5: The row-column transposition not required(only applies to 2-D DFT).

Array	#PE PE-I	PE-II	PE add.	mult.	Comp.	Thrpt	Memory	Comm.	R/C
semi	N	1	$4+\frac{4}{N}$	$2^*+\frac{4}{N}$	$2N+1$	N	Note 2	Note 4	N/A
pure	N	1	$4+\frac{4}{N}$	$2^*+\frac{4}{N}$	$3N$	N	Note 2	Note 3	N/A
2-D	N^2	N	$4+\frac{4}{N}$	$2^*+(\frac{4}{N})^*$	$2N$	1	Note 2	Note 3	N/A

Table 3.1: Characteristics of the proposed 1-D DFT systolic array

Array	#PE PE-I	PE-II	PE add.	mult.	Comp.	Thrpt	Memory	Comm.	R/C
2-D	$2N^2$	$2N$	$4+\frac{4}{N}$	$2^*+(\frac{4}{N})^*$	$5N+1$	N	Note 1	Note 3	Note 5

Table 3.2: Characteristics of the proposed 2-D DFT systolic array

*: Static twiddle factors used.

For the 1-D DFT, the semi-systolic array and the pure-systolic arrays have the same number of PEs and the same throughput rate. The PE of the pure-systolic array is slightly more complex due to a modified output path and therefore, needs $N-1$ cycles more computation time for one transform. In return, the pure-systolic array enjoys local communication. Both of the two 1-D arrays have about half the number of multipliers compared to the existing 1-D structures in table 2.1. The 2-D array, obtains higher throughput by increasing the number of PEs and enjoys local communication. Compared to the 2-D structure of matrix-matrix multiplication in table 2.1, it has the advantage of reduced PE complexity and no extra memory to store or retrieve data and results is needed. The computation time and throughput rate of both structures are about the same.

For the 2-D DFT, our structure avoids the row-column transposition while some delay units, or extra memory, is needed between the two stages. The two arrays for the row and the column computations are exactly the same and can be realized on two identical chips. The same chip can be used for either the 1-D or the 2-D DFT computations. The R/C transposition structure in table 2.2 uses the similar approach but a row-column transposition is needed, besides the extra memory to arrange input data. Chen's structure combines the two 1-D DFT computations on an N by N array, which has static twiddle factors in the PEs. Chang's and Zhu's structures are not recommended due to the global communication requirement and the low throughput rates. In addition, our structure requires the lowest number of multipliers, about half of that of the other 4 structures.

Above all, our structures enjoy lower PE complexity and static twiddle factors.

Stored-product ROM's can be used in place of multipliers to further reduce area and increase accuracy. With three structures for the 1-D DFT, and one for the 2-D DFT to choose from, we believe these provide a variety of good choices to satisfy different requirements. In chapter 4, we will study the performance of all the DFT systolic arrays for both fixed-point and floating-point realizations.

Chapter 4

Performance Analysis of the DFT Systolic Arrays

4.1 Introduction

In this chapter, the effects of fixed-point as well as floating-point arithmetic in the implementations of DFT systolic arrays are analyzed. The performance of the DFT systolic arrays seen in chapter 2 and 3 are assessed by means of an error analysis for the first and second-order Goertzel algorithm. In the first part of the chapter, the overflow problem and the signal-to-noise ratio(SNR) for the second-order Goertzel algorithm are studied for the fixed-point implementation. These results are then compared to those of the first-order Goertzel algorithm. Finally, the floating-point realizations of DFT systolic arrays are discussed.

4.2 Fixed-point error analysis of the DFT systolic arrays

As seen in chapter 2, the Goertzel algorithm implements the DFT as a finite impulse response recursive filter. Here, we study the SNR and the overflow problem of the Goertzel algorithm for fixed-point implementations. In the next two sections, the

second-order Goertzel algorithm, upon which our proposed structures are based, is studied. The error analysis for the first-order algorithm can be found in [14]. We will use its results for comparison with those for the second-order algorithm in section 4.2.3.

4.2.1 Overflow analysis of the second-order Goertzel algorithm

In this section, the overflow problem is studied and the scaling factor s is derived.

The second-order Goertzel algorithm implements the DFT as a finite impulse response recursive filter. The filter realization of the algorithm is shown in Figure 4.1.

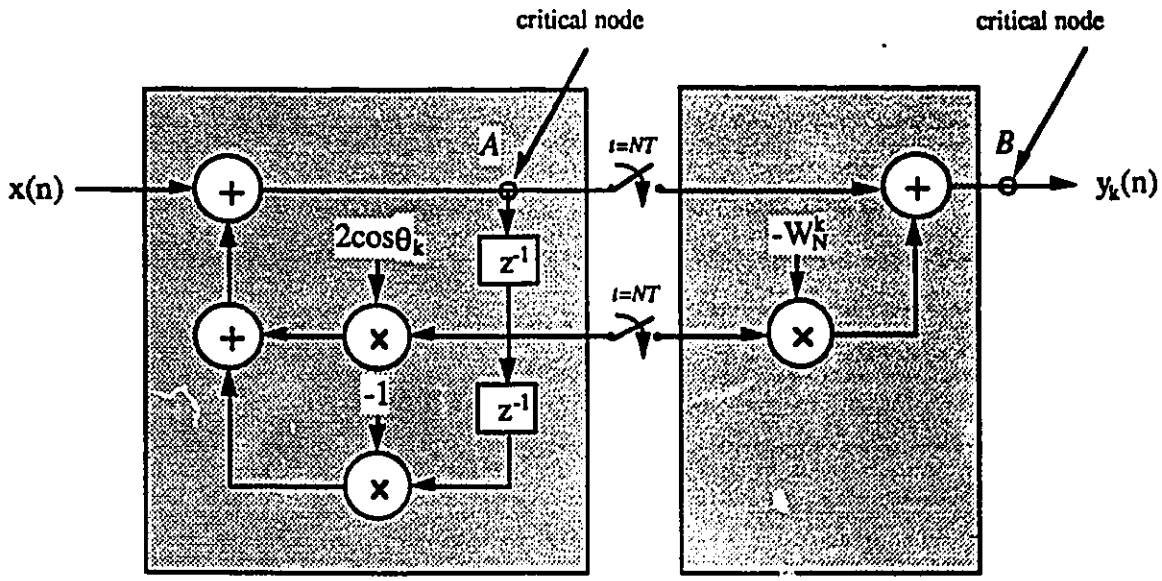

Figure 4.1: Filter realization of the second-order Goertzel algorithm

The signal at each node within the filter must be constrained to a magnitude smaller than 1 in order to avoid overflow [7]. We see that A and B are two critical nodes in Figure 4.1.

If $w(n)$ denotes the value of the signal at node A and $h(n)$ denotes the impulse response from the input $x(n)$ to that node, then,

$$| w(n) | = | \sum_{m=0}^{N-1} [x(n-m)h(m)] | \tag{4.1}$$

Here, the recursion is performed only N times, and if x_{max} denotes the maximum of the absolute value of $x(n)$, then,

$$| w(n) | < x_{max} \sum_{m=0}^{N-1} | h(m) | \tag{4.2}$$

Since $| w(n) |$ must be smaller than 1, thus

$$x_{max} < \frac{1}{\sum_{m=0}^{N-1} | h(m) |} \tag{4.3}$$

The system function at the critical node A in Figure 4.1, where the recursion results accumulate, is

$$H(z) = \frac{1}{1 - 2cos\theta_k z^{-1} + z^{-2}} \tag{4.4}$$

From Eq.(4.4), the impulse response can be derived (shown in Appendix A),

$$h(m) = \frac{sin[\theta_k(m+1)]}{sin\theta_k} \tag{4.5}$$

For $| w(n) | < 1$ or $x_{max} \sum_{m=0}^{N-1} | h(m) | < 1$, thus

$$\begin{aligned} x_{max} &< \frac{1}{\sum_{m=0}^{N-1} | h(m) |} \\ &= \frac{| sin\theta_k |}{\sum_{m=0}^{N-1} | sin[\theta_k(m+1)] |} \end{aligned} \tag{4.6}$$

The right hand side term in Eq.(4.6) varies. When $\theta_k = 0$ or π, the worst case, a bound of $\frac{2}{N(N+1)}$ is found and

$$x_{max} < \frac{2}{N(N+1)} \tag{4.7}$$

thus, the required scaling factor is

$$s = \frac{2}{N(N+1)} \tag{4.8}$$

Similarly, we can find the scaling factor for node B is $\frac{1}{\sqrt{2}N}$, which is larger than that of node A. So the condition for no overflow of the system in Figure 4.1 is Eq.4.8.

When the input is scaled down by s, the signal-to-noise ratio at the output of the system will be reduced, because the noise is injected after the scaling. The ratio of signal power to noise power in the scaled system is s^2 times that of the same system without scaling. We will see this in the next section, where the SNR is discussed in detail.

4.2.2 Signal-to-noise ratio of the second-order Goertzel algorithm

In this section, the signal-to-noise ratio(SNR) of the second-order Goertzel algorithm is studied. We assume that all the partial results are rounded to B bits after every recursion and kept in the PE-I's the DFT array structures until the intermediate results $y_k(N-1)$ and $y_k(N-2)$ are obtained and passed to PE-II (as shown in Figure 3.1 and 3.6).

Expanding Eq.(2.25) into the real and imaginary parts, we can write the following equations for the complex input samples:

$$\begin{aligned} Re[y_k(n)] &= Re[x(n)] + 2cos(2\pi k/N)Re[y_k(n-1)] - Re[y_k(n-2)] \\ Im[y_k(n)] &= Im[x(n)] + 2cos(2\pi k/N)Im[y_k(n-1)] - Im[y_k(n-2)] \end{aligned} \tag{4.9}$$

where, $n, k = 0, 1, ..., N-1$, and

$$\begin{aligned} Re[X(n)] &= Re[y_k(N-1)] - cos(2\pi k/N)Re[y_k(N-2)] - sin(2\pi k/N)Im[y_k(N-2)] \\ Im[X(n)] &= Im[y_k(N-1)] - cos(2\pi k/N)Im[y_k(N-2)] + sin(2\pi k/N)Re[y_k(N-2)] \end{aligned}$$

where, $n, k = 0, 1, ..., N-1$.

For fixed-point arithmetic, roundoff errors are introduced only by multiplications. For fractional numbers, the product of two B-bit numbers(excluding sign bit) gives a result smaller 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.

The rounding effect is treated as an additive noise signal [7]. Figure 4.2 shows the model for the error analysis of the second-order Goertzel algorithm. $x(n)$ is the complex input signal and $y_k(n)$ is the complex output. Part I and Part II are shown for the two computation stages. Node A at the output of Part I is the crucial node for the analysis. The error sources $e(1)$ and $e(2)$ intervene $N-1$ times and $e(3)$, $e(4)$, $e(5)$ and $e(6)$ only once.

Before we start calculating the SNR, let's define the following notations:

- $g_{xr}(n)$: complex impulse response from the input signal $Re[x(n)]$ to the system output.
- $g_{xi}(n)$: complex impulse response from the input signal $Im[x(n)]$ to the system output.
- $h_{ek}(n)$: complex impulse response from the noise source $e(k)$ to the output of the system.
- σ_x^2: Variance of input signals.

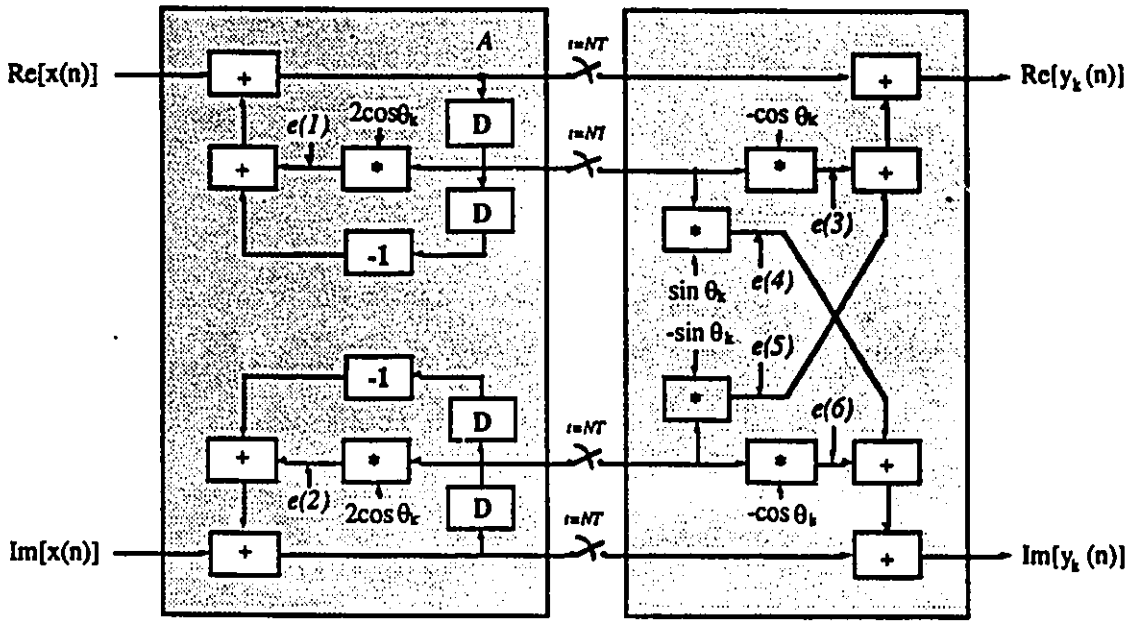

Figure 4.2: Flow graph of a basic cell for the realization of the 2nd-order Goertzel algorithm

- σ_y^2: Variance of output signals.
- σ_e^2: Variance of errors caused by rounding.
- σ_f^2: Variance of roundoff errors at the system output.

The SNR is defined as the ratio of the variance of the output signal σ_y^2 and the variance of the roundoff errors at the output σ_f^2. From the impulse responses of the signal and the errors, one can derive σ_y^2 and σ_f^2, respectively.

From Appendix B, the impulse responses of the system in Figure 4.2 are:

$$\begin{aligned} Re[g_{xr}(n)] &= Im[g_{xi}(n)] = cos(\theta_k n) \\ Im[g_{xr}(n)] &= -Re[g_{xi}(n)] = sin(\theta_k n) \end{aligned} \tag{4.10}$$

The variance of the signal at the real part of the output is,

$$\begin{aligned} \sigma_y^2 &= \sigma_x^2\{\sum_{n=\infty}^{\infty}(Re^2[g_{xr}(n)] + Re^2[g_{xi}(n)]\} \\ &= \sigma_x^2\{\sum_{n=0}^{N-1}[(cos\theta_k n)^2 + (-sin\theta_k n)^2\} \\ &= \sigma_x^2 N \end{aligned} \tag{4.11}$$

With the assumption that the input signal has a uniform distribution of amplitudes, it can be found that the relationship of the scaling factor s and σ_x^2 is

$$\sigma_x^2 = \frac{s^2}{3} \tag{4.12}$$

Since $s = \dfrac{2}{N(N+1)}$, as shown in section 4.2, it is found that,

$$\begin{aligned} \sigma_y^2 &= \frac{1}{3}[\frac{2}{N(N+1)}]^2 N \\ &= \frac{4}{3N(N+1)^2} \end{aligned} \tag{4.13}$$

Similarly, the impulse responses from the noise sources $e(1) \sim e(6)$ to the output of the system can be found,

$$\begin{aligned}
&Re[h_{e1}(n)] = Im[h_{e1}(n)] = cos(\theta_k n) \\
&Im[h_{e1}(n)] = -Re[h_{e2}(n)] = sin(\theta_k n) \\
&Re[h_{e3}(n)] = Im[h_{e6}(n)] = 1 \\
&Im[h_{e3}(n)] = Re[h_{e6}(n)] = 0 \\
&Re[h_{e4}(n)] = Im[h_{e5}(n)] = 0 \\
&Im[h_{e4}(n)] = Re[h_{e5}(n)] = 1
\end{aligned} \tag{4.14}$$

here, $\theta_k = 2\pi k/N, \quad n, k = 0, 1, ...N-1$.

The variance of roundoff errors at the real output is equal to

$$\begin{aligned}
\sigma_f^2 &= \sum_{k=1}^{6} \sigma_{ek}^2 \sum_{\infty}^{\infty} h_{ek}^2(n) \\
&= \sigma_e^2 \{ \sum_{n=0}^{N-1} (Re^2[h_{e1}(n)] + Re^2[h_{e2}(n)]) \\
&\quad + \sum_{n=N}^{N} (Re^2[h_{e3}(n)] + Re^2[h_{e4}(n)] + Re^2[h_{e5}(n)] + Re^2[h_{e6}(n)]) \} \\
&= \sigma_e^2 \{ \sum_{n=0}^{N-1} [(cos\theta_k n)^2 + (-sin\theta_k n)^2 + [1+0+1+0] \} \\
&= \sigma_e^2 (N+2)
\end{aligned} \tag{4.15}$$

The rounding error σ_e^2 can be obtained with the assumption that each noise source has a uniform distribution of amplitudes over the quantization interval [7],

$$\sigma_e^2 = \frac{2^{-2B}}{12} \tag{4.16}$$

So,

$$\sigma_f^2 = (N+2)\frac{2^{-2B}}{12} \tag{4.17}$$

It follows that the SNR for second-order Goertzel algorithm is equal to

$$\begin{aligned} SNR &= \frac{\sigma_y^2}{\sigma_f^2} \\ &= \frac{2^{2B+4}}{N(N+1)^2(N+2)} \end{aligned} \tag{4.18}$$

In the next section, this result is compared to that of the first-order Goertzel algorithm.

4.2.3 Comparison of the first and second-order Goertzel algorithm

This section presents a comparative study of the second-order Goertzel algorithm analyzed in this chapter along with the first-order Goertzel algorithm[14] for fixed-point implementations.

The scaling factors and the SNR of the algorithms to be compared are summarized in table 4.1, where B is the accuracy , or wordlength of the method, and N is the DFT length.

The scaling factors in table 4.1 for different algorithms are based on the worst case. Therefore, they are very conservative and may not be used for actual implementations in this form. Nevertheless, they give the characteristics of the three algorithms.

The results are presented graphically on Figure 4.3 4.4, 4.5, 4.6 and 4.7. The first-order and the modified first-order Goertzel algorithm as seen on the graphs give about the same results. The second-order Goertzel algorithm has the lowest SNR due to its small scaling factor to prevent overflow. This factor is proportional to $1/N^2$ and is introduced by the higher order of the recursive part of the algorithm. However, when N is small, i.e. $N = 8$, the three methods have about the same SNR.

Now, we further our study of the algorithms to the comparison of different wordlength

algorithms	scaling factor	SNR
1st-order	$\frac{1}{\sqrt{2}N}$	$\frac{2^{2B}}{N(N+1)}$
modified	$\frac{1}{2N}$	$\frac{2^{2B-1}}{N^2}$
2nd-order	$\frac{2}{N(N+1)}$	$\frac{2^{2B+4}}{N(N+1)^2(N+2)}$

Table 4.1: Comparison of the scaling factor and SNR for the Goertzel algorithm

requirements for the same SNR. Since the SNRs for the first-order and the modified Goertzel algorithm are very close, we will choose one of them, the first-order Goertzel algorithm, to compare to the second-order algorithm. To achieve the same SNR for the two algorithms, we have

$$SNR_1 = SNR_2 \tag{4.19}$$

So,

$$\frac{2^{B_1}}{N(N+1)} = \frac{2^{B_2+4}}{N(N+1)^2(N+2)} \tag{4.20}$$

Here, B_1 and B_2, SNR_1 and SNR_2 stand for the accuracy and SNR for the first-order and the second-order Goertzel algorithm, respectively. Then,

$$2^{2(B_2-B_1)} = \frac{(N+1)(N+2)}{16} \tag{4.21}$$

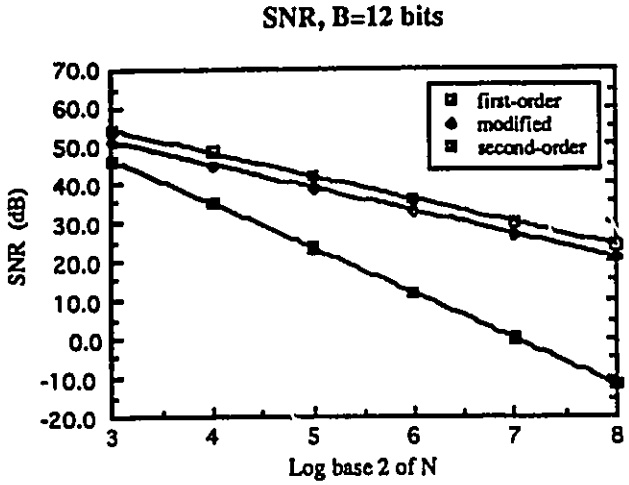

Figure 4.3: SNR vs N for the three methods when B=12 bits

Let $\Delta B = B_2 - B_1$ to represent the extra bits the second-order algorithm has to use vs its first-order counterpart, Eq.(4.21) becomes

$$\begin{aligned} 2^{\Delta B} &= \sqrt{\frac{(N+1)(N+2)}{16}} \\ &\leq \frac{N+2}{4} \quad . \end{aligned} \tag{4.22}$$

Thus,

$$\begin{aligned} \Delta B &= \log_2 \frac{N+2}{4} \\ &= \log_2(N-2) - 2 \end{aligned} \tag{4.23}$$

When $N \gg 1$,

$$\Delta B \doteq \log_2 N - 2 \tag{4.24}$$

Table 4.2 shows some examples for different DFT lengths using Eq.(4.24).

As we can see, as N getting larger, ΔB is very close to $(\log_2 N - 2)$. Therefore, to achieve the same SNR, the second-order algorithm has to use $(k-2)$ bits more for the

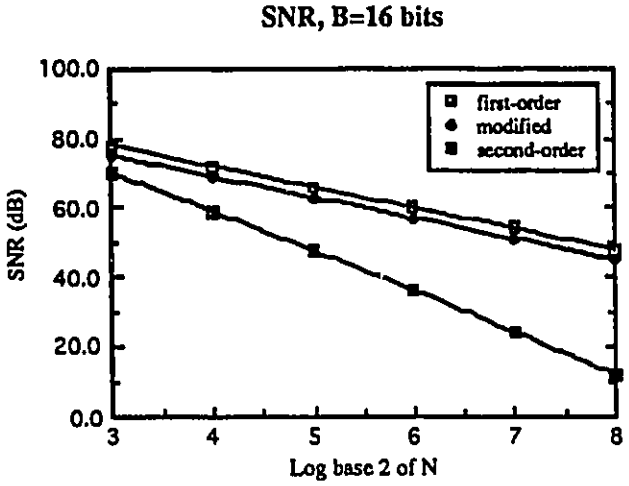

Figure 4.4: SNR vs N for the three methods when B=16 bits

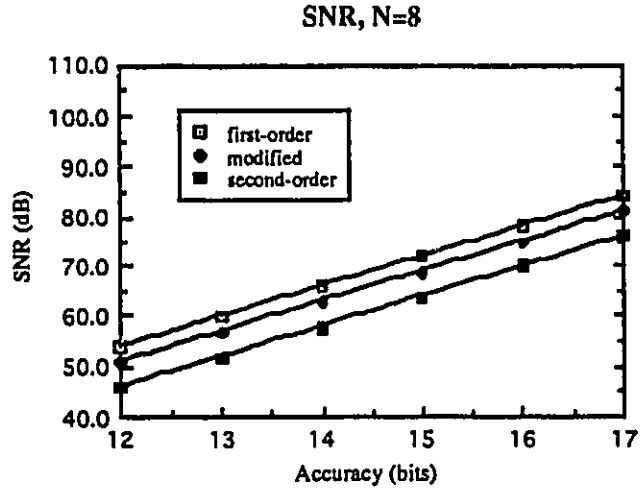

Figure 4.5: SNR vs accuracy for the three methods when N=8

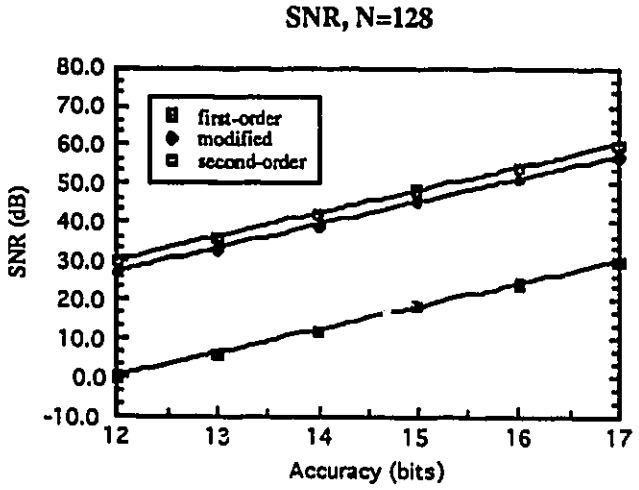

Figure 4.6: SNR vs accuracy for the three methods when N=128

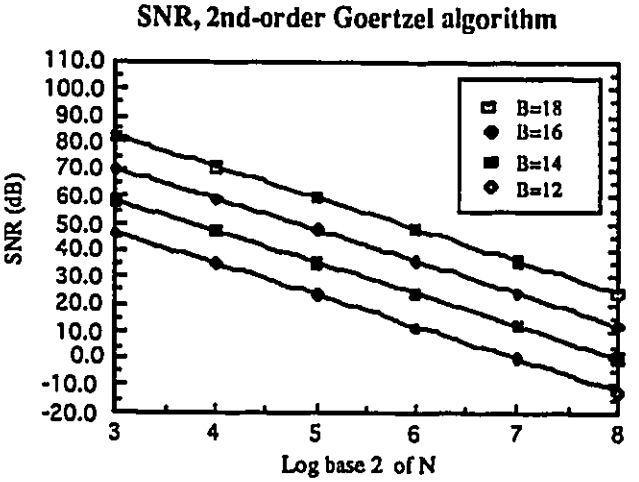

Figure 4.7: SNR vs accuracy for the 2nd-order Goertzel algorithm when B=12,14,16,18

DFT length N	$\triangle B = B_2 - B_1$
8	1.32
16	2.13
32	3.07
64	4.04
128	5.02
256	6.01

Table 4.2: Examples of wordlength increases for the 2nd-order Goertzel algorithm over the wordlength required for the 1st-order Goertzel algorithm with different lengths of the 1-D DFT

accuracy (where $2^k = N$, and $k = 0, 1, 2, ...$) than the first-order algorithm.

From the above analysis, it has been found that for fixed-point implementations, the second-order Goertzel algorithm requires a scaling factor proportional to $1/N^2$. Instead, the first-order and the modified first-order algorithm need only a scaling factor proportional to $1/N$. Subsequently, the SNR study favors the two first-order algorithms.

However, as discussed in chapter 3, the systolic array architectures based on the second-order Goertzel algorithm greatly reduces hardware complexity. Therefore, we have a choice of using the second-order algorithm for less multipliers but longer wordlength, or using the first-order algorithm for shorter wordlength but more multipliers. The situation would be different if the floating-point realizations are used. The need for such scaling can be essentially eliminated by using the floating-point realizations.

In floating-point representations, a real number is represented by the exponent of the scale factor *characteristic* and the fractional part *mantissa*. This provide a convenient means for maintain both a wide dynamic range and small quantization noise. Therefore, the use of floating-point arithmetic entails increased wordlength and increases complexity in the arithmetic unit. Its major advantage is that it essentially eliminates the problems of overflow, and if a sufficiently long mantissa is used, quantization also becomes much less of a problem.

For the three forms of the Goertzel algorithm in the floating-point representation, the scaling factor required in the fixed-point realizations, becomes unnecessary. Overflows are can be avoided and better SNR can be expected.

Thus, for DFT systolic array implementations, the hardware complexity in terms of number of multipliers and adders becomes the main consideration. The proposed systolic array structures are the best among the existing structures. Based on the

second-order Goertzel algorithm, these structures use about half of multipliers of other structures, therefore, greatly reduced hardware complexity.

4.3 Discussion

In this chapter, the effects of using fixed-point arithmetic in the implementations of DFT systolic arrays have been analyzed.

The performance of the DFT systolic arrays seen in chapter 2 and 3 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 effective than its second-order counterpart for fixed-point implementations. The second-order algorithm required a scaling factor proportional to $1/N^2$. Instead, the first-order and the modified first-order algorithm need only a scaling factor proportional to $1/N$. When N is small, i.e. $N = 8$, their SNR are about the same.

However, our proposed systolic array structures that are based on the second-order Goertzel algorithm greatly reduce the hardware complexity. Therefore, we have a choice of using the second-order algorithm for less multipliers but longer wordlength, or using the first-order algorithm for shorter wordlength but more multipliers.

For floating-point realizations, the proposed systolic arrays perform the best among existing structures. Since the need for scaling can be essentially eliminated, the hardware requirement in terms of number of multipliers and adders has become the main consideration.

Chapter 5

Conclusion

The field of special purpose digital processor development is evolving rapidly. This results in part from advances in the VLSI technology, which promises low-cost, high density and fast special-purpose devices able to perform tasks like the discrete Fourier transforms using parallel architectures. The repetitive use of common structures (i.e., modules) and minimization of "random" interconnections are crucial to reducing the design efforts and to creating producible integrated circuits.

The systolic array concept has been presented as simple means for mapping a signal processing algorithm onto a parallel architecture. The simple processing elements communicate in a regular fashion with their nearest neighbors, which is amenable to VLSI circuit implementation.

In this thesis, the systolic array realizations for the computation of the DFT have been studied. A survey of available computational algorithms, has been presented, with emphasis on the first and second-order Goertzel algorithm, which are most interesting for systolic array realization. The criteria for evaluating these algorithms are based on the actual number of arithmetic operations required as well as the degree of complexity of the communication between the processing elements.

Subsequently, existing systolic arrays for the computation of 1-D and the 2-D DFT

have been surveyed. Most of these are based on one of the Goertzel algorithms or on the direct calculation of the DFT. A comparison of seven 1-D DFT systolic arrays has shown that the matrix-vector multiplication and Kung's 1-D structures cannot be used in applications where the DFT of a continuous flow of sequences is required. The 2-D matrix-matrix multiplication structure has the highest throughput rate by increasing the number of processing elements but need an input buffer to pre-arrange the data. For Beraldin's two arrays, stored-product ROM's can be used for the multipliers due to the static coefficients. All these arrays, together with a few other structures introduced, require processing elements that have 4 multipliers and 4 adders.

Then four 2-D DFT systolic arrays have been reviewed. Zhu's structure has a low throughput rate and needs global communication, which is not suitable for VLSI implementation. Chang's structure combines two 1-D array structures but has a low throughput rate as well. The 2-D DFT requires two 1-D DFT arrays and a row-column transposition between the two stages. The R/C transposition structure, as its name, needs a row-column transposition section between the two 1-D DFT stages. Chang's structure is the best among the four available in terms of throughput rate and hardware complexity. However, this 2-D DFT array cannot be used for the 1-D DFT computation due to its structure.

Therefore, several new systolic arrays have been proposed for the calculation of the 1-D and the 2-D DFT using the second-order Goertzel algorithm. For the 1-D DFT, the semi-systolic array and the pure-systolic array have the same number of PEs and throughput rate. The PE of the pure-systolic array is slightly more complex due to a modified output path and therefore, needs $N-1$ cycles more computation time for one transform. In return, the pure-systolic array enjoys local communication. Both of the two 1-D arrays have about half the number of multipliers of that of the existing 1-D

structures. The 2-D array obtains higher throughput rate by increasing the number of PEs and enjoys local communication. Compared to the 2-D structure of matrix-matrix multiplication, it has the advantage of reduced PE complexity and no extra memory to store or retrieve data and results is needed. The computation time and throughput rate of both structures are about the same. For the 2-D DFT, our structure not only has reduced PE complexity but also avoids the row-column transposition while some delay units, or extra memory, is needed between the two stages. The two arrays for the row and the column computations are exactly the same and can be realized on two identical chips. The same chip can be used for the 1-D or for the 2-D DFT computations.

Then we proceeded with an analysis of the effects of the fixed-point as well as the floating-point arithmetic in the implementation of DFT systolic arrays for the first-order, the modified first-order and the second-order Goertzel algorithms, which realize the proposed DFT systolic arrays. It has been found that the first-order Goertzel algorithm is more effective than its second-order counterpart for fixed-point implementations. The second-order algorithm required a scaling factor proportional to $1/N^2$. Instead, the first-order and the modified first-order algorithm need only a scaling factor proportional to $1/N$. When N is small, i.e. $N = 8$, their SNRs are about the same. Only when N gets large, the second-order arithmetic need about $(log_2 N - 2)$ bits more for the wordlength to achieve the same SNR as the first-order algorithm. However, our proposed systolic array structures, that are based on the second-order Goertzel algorithm, greatly reduce the hardware complexity. Therefore, we have a choice of using the second-order algorithm for fewer multipliers but longer wordlength, or using the first-order algorithm for shorter wordlength but more multipliers. For floating-point realizations, the proposed systolic arrays perform the best among existing structures. Since the need for scaling can be essentially eliminated, the hardware requirement in

terms of number of multipliers and adders has become the main consideration.

Though, this research work has been concerned only with systolic array realizations of the 1-D and the 2-D DFT, other digital signal processing algorithms can benefit from the study, i.e. the discrete cosine transforms(DCT). Since the DCT is nothing but the real part of the DFT, all the proposed systolic array structures can be used to efficiently compute the DCT with minor modifications. Today, 8×8 2-D DCT is widely used for video compression and HDTV applications, the structure we proposed for the 2-D DFT is a promising choice for the task.

The original contributions of this thesis research are the development of efficient systolic array structures for the computation of the 1-D and the 2-D DFT by means of the second-order Goertzel algorithm. Another contribution is the error analysis of the second-order Goertzel algorithm.

Appendix A

In this part, the impulse response at the critical node A in Figure 4.1 of section 4.2.1 is derived from its system function.

The system function at the critical node A in Figure 4.1 is, as shown in Eq.(4.5),

$$H(z) = \frac{1}{1 - 2cos\theta_k z^{-1} + z^{-2}} \tag{5.1}$$

The roots of $H(z)$ are,

$$\begin{aligned} z_1 &= cos\theta_k + jsin\theta_k \\ z_2 &= cos\theta_k - jsin\theta_k \end{aligned} \tag{5.2}$$

Eq.(5.1) can be rewritten as,

$$\begin{aligned} H(z) &= \frac{1}{(1 - z_1 z^{-1})(1 - z_2 z^{-1})} \\ &= \frac{A}{1 - z_1 z^{-1}} + \frac{B}{1 - z_2 z^{-1}} \end{aligned} \tag{5.3}$$

The facters A and B can be found,

$$\begin{aligned} A &= \frac{1 - jctg\theta_k}{2} \\ B &= \frac{1 + jctg\theta_k}{2} \end{aligned} \tag{5.4}$$

The impulse response of the system can be found from the transfer function,

$$\begin{aligned} h(n) &= Az_1^n u[n] + Bz_2^n u[n] \\ &= \frac{1 - jctg\theta_k}{2}(cos\theta_k + jsin\theta_k)^n u[n] \\ &\quad + \frac{1 + jctg\theta_k}{2}(cos\theta_k - jsin\theta_k)^n u[n] \\ &= \frac{1}{2}\{[(cos\theta_k + jsin\theta_k)^n + (cos\theta_k - jsin\theta_k)^n] \\ &\quad -jctg\theta_k[(cos\theta_k + jsin\theta_k)^n - (cos\theta_k - jsin\theta_k)^n]\}u[n] \end{aligned} \tag{5.5}$$

Let, $a = cos\theta_k$ and $b = jsin\theta_k$, then

$$h(n) = \frac{1}{2}\{[(a+b)^n + (a-b)^n] - j\frac{a}{b}[(a+b)^n - (a-b)^n]\}u[n] \tag{5.6}$$

Since,

$$\begin{aligned} (a+b)^n &= C_n^0 a^n b^0 + C_n^1 a^{n-1} b + C_n^2 a^{n-2} b^2 + C_n^3 a^{n-3} b^3 \\ &\quad + ... + C_n^{n-1} a^1 b^{n-1} + C_n^n a^0 b^n \end{aligned} \tag{5.7}$$

$$\begin{aligned} (a-b)^n &= C_n^0 a^n b^0 - C_n^1 a^{n-1} b + C_n^2 a^{n-2} b^2 - C_n^3 a^{n-3} b^3 \\ &\quad + ... - C_n^{n-1} a^1 b^{n-1} + C_n^n a^0 b^n \end{aligned} \tag{5.8}$$

one can find,

$$\begin{aligned} (a+b)^n + (a-b)^n &= 2[C_n^0 a^n b^0 + C_n^2 a^{n-2} b^2 + ... + C_n^{n-2} a^2 b^{n-2} + C_n^n a^0 b^n] \\ &= 2[C_n^0 (cos\theta_k)^n - C_n^2 (cos\theta_k)^{n-2} (sin\theta_k)^2 + ... \\ &\quad -C_n^{n-2} (cos\theta_k)^2 (sin\theta_k)^{n-2} + C_n^0 (sin\theta_k)^n] \\ &= 2cos\theta_k n \end{aligned} \tag{5.9}$$

$$\begin{aligned} (a+b)^n - (a-b)^n &= 2[C_n^1 a^{n-1} b^1 + C_n^3 a^{n-3} b^3 + ... + C_n^{n-3} a^3 b^{n-3} + C_n^{n-1} a^1 b^{n-1}] \\ &= 2j[C_n^1 (cos\theta_k)^{n-1} (sin\theta_k) - C_n^3 (cos\theta_k)^{n-3} (sin\theta_k)^3 + ... \\ &\quad -C_n^{n-3} (cos\theta_k)^3 (sin\theta_k)^{n-3} + C_{n-1}^0 (cos\theta_k)(sin\theta_k)^{n-1}] \\ &= 2jsin\theta_k n \end{aligned} \tag{5.10}$$

Thus, the impulse response of the system at node A is equal to

$$\begin{aligned} h(n) &= \frac{1}{2}\{[(a+b)^n + (a-b)^n] - j\frac{a}{b}[(a+b)^n - (a-b)^n]\}u[n] \\ &= [(cos\theta_k n) - jctg\theta_k(sin\theta_k n)]u[n] \\ &= \frac{1}{sin(\theta_k n)}[cos(\theta_k n)sin\theta_k + cos\theta_k sin(\theta_k n)]u[n] \\ &= \frac{1}{sin(\theta_k n)}sin(\theta_k + \theta_k n)u[n] \\ &= \frac{sin[\theta_k(n+1)]}{sin\theta_k} \end{aligned} \tag{5.11}$$

Appendix B

In this part, the impulse responses of the system in Figure 4.2 of section 4.2.2 are derived.

From the transfer function of the second-order Goertzel algorithm in Eq.(2.24), one can find the transfer function from input $Re[x(n)]$ to the real part of the output $Re[y_k(n)]$ in Figure 4.2,

$$Re[G_{xr}(z)] = Re[H(z)] = \frac{1 - cos\theta_k z^{-1}}{1 - 2cos\theta_k z^{-1} + z^{-2}} \tag{5.12}$$

From Eq.(5.12) and (4.5) the impulse response from input $Re[x(n)]$ to the real part of the output $y_k(n)$ can be found,

$$\begin{aligned} Re[g_{xr}(n)] &= \frac{Re[y_k(n)]}{Re[x(n)]} \\ &= (1 - cos\theta_k \delta(n-1)) \frac{sin[\theta_k(n+1)]}{sin\theta_k} \\ &= \frac{sin[\theta_k(n+1)] - cos\theta_k sin[\theta_k n]}{} \\ &= \frac{(sin[\theta_k n]cos\theta_k + cos[\theta_k n]sin\theta_k) - cos[\theta_k n]sin\theta_k}{sin\theta_k} \\ &= cos(\theta_k n) \end{aligned} \tag{5.13}$$

The transfer function from input $Re[x(n)]$ to the imaginary part of the output $Im[y_k(n)]$ is:

$$Im[G_{xr}(z)] = Im[H(z)] = \frac{sin\theta_k z^{-1}}{1 - 2cos\theta_k z^{-1} + z^{-2}} \tag{5.14}$$

Then the impulse response from the input $Re[x(n)]$ to the imaginary part of the output $y_k(n)$ is equal to,

$$\begin{aligned} Im[g_{xr}(n)] &= \frac{Im[y_k(n)]}{Re[x(n)]} \\ &= sin\theta_k\ \delta(n-1)\frac{sin[\theta_k(n+1)]}{sin\theta_k} \\ &= sin(\theta_k n) \end{aligned} \tag{5.15}$$

Bibliography

[1] S.Y. Kung, "VLSI Array Processors", Engle-Wood Cliffs, NJ: Prentice Hall, 1988.

[2] S,Y, Kung, "Systolic Signal Processing Systems",E.E. Swantzlander, editor, Marcel Dekker, Inc., New York, N.Y., 1987.

[3] H.T. Kung, "Why Systolic Architectures?", IEEE Transaction on Computer, pp.37-46, January, 1982.

[4] S. Singh and J.Y. Han, "Systolic Arrays", IEEE Potentials, pp.7-12, February, 1991.

[5] C.S. Burrus and T.W. Parks, "DFT/FFT and Convolution Algorithms, Theory and Implementation", John Wiley & Sons, Inc., New York, N.Y., 1985.

[6] J.A. Beraldin, T. Aboulnasr and W. Steenaart, "Efficient One-dimensional Systolic Array Realization of the Discrete Fourier Transform", IEEE Trans. on Circuits and Systems, Vol.36, No.1, pp.95-100, January 1989.

[7] A.V. Oppenheim and R.W. Schafer, "Discrete-time Signal Processing", Engle-Wood Cliffs, NJ: Prentice Hall, 1989.

[8] P. Duhamel and M. Vetterli, "Fast Fourier Transforms: A Tutorial Review and a State of Art", IEEE, Signal Processing, Vol.19, No.4, pp.259-299, April, 1990.

[9] Y. Zhu, L. Jin and Z. Zheng, "Architecture of Array Processors of 1-D, 2-D Complex and Real DFT", International Conference on Circuits and Systems, China, June, 1991.

[10] J.S. Lim, A.V. Oppenheim, *Advanced Topics in Signal Processing*, Chap. 7, Engle-Wood Cliffs, NJ: Prentice Hall, 1988.

[11] R.B. Urquhart and D. Wood, "Systolic Matrix and Vector Multiplication Methods for Signal Processing", IEE Proc., Vol.131,Pt.F, No.6, pp.623-1066, October 1984.

[12] L.W. Chang and M.Y. Chen, "A New Systolic Array for Discrete Fourier Transform", IEEE Transactions on Acoustics, Speech, and Signal Processing, vol.36, No.10, pp.1665-1666, October 1988.

[13] C.Y. Chen and C.L. Wang, "A New Efficient Systolic Architecture for the 2-D Discrete Fourier Transform", International Symposium of Circuits and Systems, pp.689-692, San Diego, CA, May 1992.

[14] J.A. Beraldin, "VLSI Systolic Array Architecture for the Computation of the Discrete Fourier Transform", Master's Thesis, University of Ottawa, Ottawa, Ontario, Canada, 1986.

[15] J.A. Beraldin and W. Steenaart, "Overflow Analysis of a Fixed-Point Implementation of the Goertzel Algorithm", IEEE Trans. on Circuits and Systems, Vol.36, No.2, pp.323-324, February 1989.

[16] N.U. Chowdary and W. Steenaart, "A High Speed Two-Dimensional FFT Processor", Proceedings ICASSP, pp.4.11.1-4.11.4, San Diego, CA, 1984.

[17] S. Gu and W. Steenaart, "2-D Discrete Fourier Transform Realization Using the Modified Goertzel Algorithm", Proceedings of Canadian Conference on Electrical and Computer Engineering, MA3.8.1, Toronto, Canada, September 1992.

[18] R.J. Higqins, "Digital Signal Processing in VLSI", Prentice Hall,Engle-Wood Cliffs, J.J., 1990.

[19] H.V. Sorensen, C.S. Burrus and D.L. Jones, "A New Efficient Algorithms for Computing a Few DFT Points", IEEE ISCAS, 1988.

[20] M.H. Lee, "On Computing 2-D Systolic Algorithm for Discrete Cosine Transform", IEEE Trans. on Circuits and Systems, Vol.37, No.10, pp.1321-1323, October, 1990.

[21] A. Antola, R. Negrini and N. Scarabottolo, " Arrays for Discrete Fourier Transform", EIRASIP, Signal Processing IV, pp.915-918, North Holland, 1988.

[22] N.I. Cho, S.U. Lee, " A Fast 4 × 4 DCT Algorithm for the Recursive 2-D DCT", IEEE Trans. on Signal Processing, Vol.40, No.9, pp.2166-2172., September 1992.

[23] C. Chakrabarti and J. JaJa, "VLSI Architectures for Multidimensional Transforms", IEEE Trans. on Computers, Vol.40, No.9, pp.1053-1057, September, 1991.

[24] J. Martin, " Future Developments in Communications", Prentice-Hall, Inc.,Englewood Cliffs, NJ, 1986.

[25] K. Hwang and F.A. Briggs, "Computer Architecture and Parallel Processing", McGraw-Hill Book Company, 1984.

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

[27] H.T. Kung and et al, "VLSI Systems and Computations", Computer Science Press, 1981.

[28] S.Y. Kung, "On Supercomputing with Systolic/wavefront Array Processors", Proceedings of IEEE, pp.867-884, July 1984.

[29] H.T. Kung and C. E. Leiserson, "Systolic Arrays (for VLSI)", Addison-Wesley, 1980.

[30] H.T. Kung, "Systolic Algorithms", Academic Press, 1984.

[31] H.T. Kung, "Special Purpose Devices for Signal and Image Processing: an Opportunity in Very Large Scale Integration(VLSI)", Proceedings of SPIE, Real-time Signal Processing, SPIE, 1980.

[32] G. Bongiovanni, "Two VLSI Structures for the Discrete Fourier Transform", IEEE Trans. on Computers, C-32(8), August 1983.

[33] C.D. Thompson, "Fourier Transforms in VLSI", IEEE Trans. on Computers, C-32(11), November 1983.

[34] J.A.B. Fortes and B.W. Wah, "Systolic Arrays: A Survey of Seven Projects", IEEE Computer, July 1987.

[35] P. Quinton, "The Systematic design of Systolic Arrays", Automata Networks, Chapter 9, 1988.

[36] D.I. Moldovan, "On the design of algorithms for VLSI systolic arrays", Proc. of IEEE, 71(1), January 1983.

[37] S.K. Rao an T. Kailath, "Regular iterative Algorithms and their Implementation on Processor Arrays", Proc. of IEEE, 76(3):259-269, March 1988.

[38] W. Steenaart, J.Y. Zhang and Y. Tunca,"Mapping Recursive Algorithms onto Array Architectures", In E. De Prettere, editor, Algorithms and Parallel VLSI Architectures, 1991.

[39] W. Steenaart and J.Y. Zhang, "Mapping Recursive Algorithms onto Systolic Arrays", In Proceedings of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, May 1991.

[40] M.T. Heideman, D.H. Johnson and X.S. Burrus, "Gauss and the History of the Fast Fourier Transform", IEEE ASSP, vol.1, No.4, pp.14-21, 1984.

[41] J.W. Cooley and J.W. Tuckey, "An Algorithm for the Machine Computation of Complex Fourier Series", Math. Comput., 19, pp.297-301, 1965.

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

[43] C.M. Rader, "Discrete Fourier Transforms When the Number of Data Samples is Prime,", Proc. IEEE, 56, pp.1107-1108, 1968.

[44] S. Winograd,"On computing the discrete Fourier transform", Mth.comput., 32, pp.175-199, 1978.

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

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

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

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