VLSI IMPLEMENTATION OF DIGITAL FILTERS

by

SREENIVASACHAR SUNDER
B.E., 1985 and M.E., 1987
Anna University, Madras, India

A DISSERTATION SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the Department of
Electrical and Computer Engineering

We accept this dissertation as conforming
to the required standard

Dr. A. Antoniou, Supervisor, Dept. of Elect. & Comp. Eng.

Dr. F. El-Guibaly, Co-Supervisor, Dept. of Elect. & Comp. Eng.

Dr. N. J. Dimopoulos, Department Member, Dept. of Elect. & Comp. Eng.

Dr. R. Vahldieck, Graduate Advisor, Dept. of Elect. & Comp. Eng.

Dr. D. M. Miller, Outside Member, Dept. of Computer Science

Dr. M. A. Sid-Ahmed, External Examiner, University of Windsor

Supervisors: Dr. A. Antoniou and Dr. F. El-Guibaly

ABSTRACT

In this thesis we describe a method of mapping one-dimensional and multidimensional filter algorithms onto systolic architectures using the z-domain approach. In this approach the filter algorithm is first transformed into its corresponding z-domain equivalent and recursive expressions similar to single assignment codes are derived using Horner's rule or other polynomial evaluation techniques. By obtaining different recursive expressions, different systolic structures can be derived. The characteristics of these structures can easily be deduced from the recursive expressions. The multidimensional filters derived are modular and hierarchical, i.e., the three-dimensional structures are obtained from the two-dimensional ones which are in turn obtained from one-dimensional structures.

In considering the design of any array processor, it is important to consider the design of the processing elements involved. The most important and demanding operation in these elements is the multiplication. Four different multipliers are designed in which the number of operations required to produce the desired result is reduced. The reduced number of operations along with the advantages of very-large-scale integration technology in terms of increased device density and faster switching make these multipliers potential candidates in high-speed signal processing applications. The first multiplier is an area-efficient multiplier that uses approximately 50% of the area of a full parallel multiplier. In this multiplier only the units yielding the most significant part of the product are used. In addition, a correction unit is incorporated to minimize the error resulting from circumventing the use of units yielding the least significant part of the product. The second multiplier is based on the modified octal Booth algorithm in which four-bit segments of the multiplier are scanned and corresponding operations effected on the multiplicand. The third multiplier is a diminished-1 multiplier that finds application in the Fermat number-theoretic transform. In this multiplier the

use of a translator is circumvented and a novel technique for translation is incorporated in the multiplier structure. The fourth multiplier is one that performs an inner-product operation without the use of an accumulator thereby resulting in increased speed and reduced area.

Finally, we discuss the VLSI implementations of three of the multipliers mentioned above, a second-order digital filter, and a single processing element that can be used as a basic unit in designing one-dimensional and multidimensional digital filters. Some associated problems in digital-filter structures, viz., the quantization and overflow limit-cycle oscillations, have been taken into consideration and ways have been suggested for their elimination.

Examiners:

Dr. A. Antoniou, Supervisor, Dept. of Elect. & Comp. Eng.

Dr. F. El-Guibaly, Co-Supervisor, Dept. of Elect. & Comp. Eng.

Dr. N. J. Dimopoulos, Department Member, Dept. of Elect. & Comp. Eng.

Dr. R. Vahldieck, Graduate Advisor, Dept. of Elect. & Comp. Eng.

Dr. D. M. Miller, Outside Member, Dept. of Computer Science

Dr. M. A. Sid-Ahmed, External Examiner, University of Windsor

Table of Contents

List of Tables

List of Figures

Abbreviations

ACM	Accumulator-multiplier
AFA	AND gate followed by a full adder
AHA	AND gate followed by a half adder
ASR	AND gate followed by a shift register
CAD	Computer aided design
CMC	Canadian Microelectronics Corporation
CMOS	Complementary metal-oxide semiconductor
CMOS3DLM	Northern Telecom Electronics 3-micron single-polysilicon, double-level metal, p-well CMOS process
CMOS4S	Northern Telecom Electronics 1.2-micron double-polysilicon, double-level metal, twin-well CMOS process
CSA	Carry-save adder
CSF	Cascaded second-order filter
DFT	Discrete Fourier transform
DG	Dependence graph
DSM	Design scale micron
FA	Full adder
FIFO	First-in-first-out
FNT	Fermat number-theoretic transform
GIC	Generalized immittance convertor
IC	Integrated circuit
LSB	Least significant bit
MAC	Multiplier-accumulator
M-D	Multidimensional
MSB	Most significant bit
NFA	NAND gate followed by a full adder
NHA	NAND gate followed by a half adder
NMM	Non-additive multiplicative module
NTE	Northern Telecom Electronics
NTT	Number theoretic transform
OMB	Octal modified Booth algorithm
PE	Processing element
PGA	Pin grid array
PLA	Programmable logic array
PSD	Power spectral density
QMB	Quarternary modified Booth algorithm
RHA	A half adder followed by a 1-bit register

RIA	Regular iterative algorithm
RX	An EXCLUSIVE-OR gate followed by a 1-bit register
SFG	Signal flow graph
SR	Shift register
SSF	Single second-order filter
SUF	Speed-up factor
SVD	Singular value decomposition
TMS	Test mode select
VLSI	Very-large-scale integration
WDF	Wave digital filter

Acknowledgments

I wish to express my gratitude to my supervisors, Dr. A. Antoniou and Dr. F. El-Guibaly of the Department of Electrical and Computer Engineering, for their encouragement, guidance, and advice during the course of this research and for their help in the preparation of this thesis.

The assistance rendered by the Canadian Microelectronics Corporation in the fabrication of VLSI chips for this project is fully acknowledged. I am grateful to Dr. D. M. Miller of the Department of Computer Science in this regard.

Financial assistance received from Dr. A. Antoniou and Dr. F. El-Guibaly (through the Natural Sciences and Engineering Research Council of Canada and the Micronet, National Centres of Excellence Program) is gratefully acknowledged.

I am grateful to my parents for making it possible for me to become what I am and to get as far as I have. My wife, Rama, has contributed to this thesis in many intangible ways for which I wish to record my very sincere gratitude.

Dedication

श्रीरामः शरणं समस्तजगतां
रामं विना का गती
रामेण प्रतिहन्यते कलिमलं
रामाय कार्यं नमः।
रामात् त्रस्यति कालभीमभुजगो
रामस्य सर्वं वशे
रामे भक्तिरखण्डिता भवतु मे
राम त्वमेव आश्रयः॥

Unto Lord Shriramachandra I dedicate this thesis

Chapter 1

Introduction

The increasing demand for processing speed and overall system performance in modern signal and image processing applications necessitates a specialized computing technology. The availability of low-cost, high-density, high-speed very-large-scale integration (VLSI) devices and emerging computer-aided design (CAD) facilities presage a major breakthrough in the design and application of massively parallel processors. In particular, VLSI microelectronics technology has inspired many innovative designs in array processor architectures. This trend has now become a major focus of attention in governments, industries, and universities. In the last decade, there has been a dramatic worldwide growth in research and development on the systematic mapping of various signal and image processing applications onto VLSI architectures.

Modern signal and image processing technology depends critically on the device and architectural innovations of the computing hardware. Sequential systems are inadequate for real-time processing systems; the additional computational capability available through VLSI concurrent array processors will become a necessity. In most real-time digital signal processing applications, general-purpose parallel computers cannot offer satisfactory processing speed due to severe demands imposed by system overhead. Therefore, special-purpose array processors will become the only appealing alternative. Let us consider a real-time application in order to substantiate the above claim. In digital video processing, it is usual to

carry out operations like filtering of images that have 512×512 pixels per frame. Assuming that there are 24 frames arriving per second and that 10 operations per pixel are carried out, the number of operations per second to be carried out would be approximately 10^7. With the present technology for the general-purpose processors, it is difficult to achieve this speed due to the classic memory access bottleneck problems and system overhead.

Current parallel computers can be put into three structural classes: vector processors, shared memory systems, and array processors [1]-[2]. The first two classes belong to the general-purpose computer domain. The development of these systems requires a complicated design of control units and optimized schemes for allocation of machine resources. The third class, however, belongs to the domain of special-purpose computers. The design of such systems requires a broad knowledge of the relationship between parallel-computing hardware and software structures. It is this class of arrays that offers a promising solution to meet real-time processing requirements.

1.1 VLSI array processors

A solution to meet real-time signal processing requirements is to use special-purpose array processors and to maximize the processing concurrency by either pipeline or parallel processing or both. An efficient system can be achieved if the array entails a balanced distribution of processor work load while observing the requirement of data locality, i.e., short communication paths. These properties of load distribution and information flow serve as guidelines to the designer of VLSI arrays. One such special-purpose VLSI array is the systolic array which entails a massive amount of concurrency. In the following section we shall discuss the concept of systolic arrays and their suitability as special-purpose computers for digital signal processing applications.

1.2 Systolic arrays

A systolic system consists of a set of regularly interconnected processors, each capable of performing a set of operations [3]. In a systolic array, data flows between processors in a rhythmic fashion, passing through many processing elements (PEs) before it returns to memory, much as blood circulates from the heart through the vascular systems and back to the heart. In this fashion data extracted from memory is used by many processors in a parallel and/or pipelined fashion thereby improving memory utilization. The major factors for adopting systolic arrays for special-purpose processors are:

1. Simple and regular design
2. Concurrency and local communication
3. Suitability for compute-bound applications

In integrated-circuit technology, the cost of components is dropping dramatically whereas the cost of design grows with the complexity of the system. Special-purpose systems are seldom produced in large quantities and in such cases part costs are less important than design costs. As a consequence, the design cost of special-purpose systems must be relatively small for them to be more attractive than general-purpose systems. Moreover, if a special-purpose system design is composed of a few types of simple PEs that are used repetitively with simple interfaces, great savings in terms of cost can be achieved. Furthermore, simple and regular systems are likely to be modular and therefore adaptable to various performance goals.

An important factor in the speed of a computing system is the use of concurrency. For special-purpose systems, concurrency depends on the underlying algorithms employed. Massive parallelism can be achieved if the algorithm is formulated such that high degrees of pipelining and multiprocessing can be introduced.

Systolic arrays are designed for compute-bound problems that are based on regular recurrence equations [4]-[10]. Consequently, they have been used in the areas of digital filtering, image and speech processing, and matrix algebra to name a few applications [4]-[14]. Several types of systolic arrays have been proposed depending on the type of structure employed, e.g., linear, triangular or hexagonal.

In an effort to obtain systolic structures for compute-bound problems, several mapping methodologies have been proposed to map algorithms directly onto systolic arrays to obtain maximum concurrency by using pipelining and parallel processing. In the following section we shall review some of the topics that are relevant to this thesis.

1.3 Review of previous work

1.3.1 Mapping methodologies

Many signal processing algorithms can be expressed as a set of iterative statements and such algorithms are called regular iterative algorithms (RIA) [5]. The common characteristic of many of the proposed methodologies for mapping RIAs onto iterative arrays is the use of a transformational approach that involves transforming the algorithm descriptions to iterative statements that are amenable to VLSI implementation. Distinct transformational systems for systolic design can be characterized by the manner in which the algorithms are described, the type of formal models used, and the type of transformations used.

In the methodology proposed by Lam and Mostow [15], an algorithm obtained by software transformations from a high-level specification, which results in segments of code executed repeatedly with a regular pattern of data accesses, is mapped onto a systolic design described by a structure and a driver. The structure describes the hardware PEs (which are functionally equivalent to the code segments), interconnections, and input-output ports. The driver defines data streams in terms of the original variables in the algorithm.

In [16], an algebraic representation is derived from the mathematical representation of the algorithm. The canonical algebraic representation consists of two expressions of the types (a) $\mathbf{v} = \mathbf{A}\mathbf{v} + \mathbf{b}x$, and (b) $y = \mathbf{c}^T\mathbf{v}$, where x represents the input, y represents the output, and $\mathbf{v}$ represents intermediate variables. The matrix $\mathbf{A}$ and the column vectors $\mathbf{b}$ and $\mathbf{c}$ represent the delays between the intermediate variables and each entry is either 0 or z^{-k}, where z is the complex variable in the z transform domain, which in the time domain represents a unit delay and k corresponds to the number of delays. Algebraic transformations are then applied to this representation. There are two major types of transformations, namely, retiming and k-slowing [16], that determine the distribution of delays and the latency periods of the systolic array. In this method, vector $\mathbf{v}$ is transformed to a vector $\mathbf{u} = \mathbf{D}\mathbf{v}$, where the matrix $\mathbf{D}$ is always a diagonal matrix whose diagonal elements are the delays. Because of this, the number of possible structures that can be obtained is limited.

In the method proposed by Moldovan [6]-[7], an algebraic model of the algorithm is derived from a set of recurrence relations, similar to those used in software compilers. This model consists of a structured set of indexed computational space where each node represents a set of computations. The algebraic representation of the algorithm is then transformed by local and global transformations. Local transformations are used to rewrite computations that are mapped into the functional and structural specifications of the PEs of the systolic architecture. Global transformations, composed of time and space transformations, are used to restructure the algorithm. They are chosen in such a way that the new algorithm has a set of dependencies that are amenable to VLSI implementation. Time transformations determine the execution time of the algorithm and the timing for data communications. Space transformations determine the interconnections and the directions of data flow.

An extension to the work by Moldovan was carried out by Miranker and Winkler [8]. In this method an algorithm is represented as either a mathematical

expression or a cyclic-loop program. The mathematical expressions are rewritten using the properties of the operators in an ad hoc manner. Theoretically this method can be applied to any algorithm although a systematic design seems possible only for those algorithms described by programs with loops.

In the method described by Capello and Steiglitz [9], starting from a set of recurrence equations describing the algorithm, a canonical representation is obtained by adjoining an index representing time to the definition of recurrence. Each index is associated with a dimension of a geometric space, where each point corresponds to a tuple of indices on which a set recurrences is defined. To each such point, a set of computations is associated, and its implementation is left unspecified. These computations are mapped directly into functional specifications of the PEs in the systolic architecture. From the geometric representation in conjunction with an ordering rule, the topology, the size of the architecture, and the timing are derived systematically. By selecting different geometric transformations, distinct representations and their corresponding architectures are derived.

In [4], a signal flow graph (SFG) representing an algorithm is first derived. The nodes of the SFG correspond to the functional description of the PEs of the architecture. Localization rules are then applied to derive a regular and temporally localized SFG. The localization procedure consists of selecting cut-sets of the SFG and reallocating scaled delays to edges leaving and entering each cut-set in such a way that at least one unit of time is allowed for communicating a signal between two nodes. Delays are combined with operational modules to obtain a full description of the operation of a basic systolic module. The resulting SFG can be mapped directly onto a systolic array by mapping basic modules into PEs and edges into interconnections. Timing and data movements are derived from the basic modules due to the localized spatial and temporal characteristics of the SFG.

Quinton [10] proposed a method based on expressing a problem as a set of uniform recurrence equations over a domain consisting of a set of index points.

In this method, given a system of n uniform recurrence equations defined over some domain $D \in Z^M$ and with some characteristic dependency vectors, a timing function that maps points of D onto time is found. This requires the identification of a convex space of feasible solutions from which one can be chosen heuristically. Such a space can be found from the knowledge of the dependency vectors and D (D can be thought of as the index set of the recurrences). Next, an allocation function is chosen, which projects D along some chosen direction such that two points in D with the same image under the timing function do not map onto the same point in space. Once the timing and allocation functions are known, the systolic array can be systematically generated.

In the method advanced by Cohen [17], starting from a mathematical expression involving subscripted variables, a new expression, where a well-defined shift operator is used to model displacements in time or shifts in space, is derived. Symbolic manipulation is used to transform the derived mathematical expression into equivalent ones by using the properties of the shift and functional operators in the expression.

In many of the approaches using matrix transformations, the number of structures possible is limited because of the restrictions in the number of feasible transformations. Moreover, especially in digital filtering, the complexity of the transformational approach increases as the dimension of the filter increases. The index space for a one-dimensional (1-D) filter is two-dimensional (2-D) and that of a 2-D filter is four-dimensional (4-D) and so on. As the dimension of the filter increases, firstly, it may be difficult to obtain a matrix representation of the problem and, secondly, the scheduling and projection vectors using the SFG method, for instance, become complicated.

In this thesis, we describe a method for mapping 1-D and multidimensional (M-D) digital filter algorithms onto systolic architectures using the z-domain approach. This method is more general than the one mentioned in [17] and easier than many of the methods that use the transformational approach mentioned

above. Any filter algorithm is first transformed into its corresponding z-domain equivalent. Different structures are obtained by reordering the summations and delays involved in the filter algorithm thereby circumventing the use of matrix transformation. The method mentioned in this thesis can be applied to obtain many advantageous structures that can satisfy a set of desirable or preset criteria such as latency, locality, and modularity.

1.3.2 Multiplier design

In considering the design of any array processor, it is important to consider the design of the PEs involved. The most important operation in any PE is multiplication. Currently, the multiplier area and time are still the dominant factors in determining the size and speed of operation of the system. In the design of multipliers discussed in the literature [18]-[25], a lot of effort has been directed towards increasing the speed of operation and decreasing the area by using the advantages of VLSI technology in terms of increased device density and faster switching. However, if multiplication algorithms are designed such that the number of operations required to produce the desired result is reduced then, together with the advantages of VLSI technology a great reduction in area and increase in speed of operation can be achieved simultaneously. In this thesis we describe, in addition, multiplier schemes that are suitable not only in the architectures proposed here but also in many other architectures used in signal processing applications.

1.3.3 Digital-filter implementation

As was mentioned earlier, the result of advances in VLSI fabrication technology has brought about a dramatic reduction in the cost of information processing. One area in which this effect is most pronounced is the field of real-time signal processing. In this area the continuous flow of data in conjunction with the complexity of many of the algorithms imposes severe computational demands that often cannot be satisfied by general-purpose machines or components. Sample

rates depend on the application, ranging from around 8 kHz for speech systems to tens and hundreds of MHz for real-time radar processors.

These demands can be met in principle by new system architectures which exploit some of the potential concurrency that is inherent in the underlying algorithms. VLSI technology offers the potential to implement such architectures. Through this technology we expect to see the implementation of powerful real-time signal processing algorithms that previously have been only of theoretical interest. However, the very advances in device technology that caused this revolution also bring a new challenge to product development of VLSI systems. Without advances in design methodology and tools, manufacturing capability and algorithm development will far exceed our capacity for system design [26].

Design of high sample rate nonrecursive filters has received considerable interest in the last two decades, both in the context of bit-parallel [27]-[28] as well as bit-serial implementations [29]-[31]. Through bit-parallel designs, implementation of high sample rate nonrecursive filter chips running up to 300 MHz has become possible. On the other hand, high sample rate recursive filters have not received much attention due to internal recursion or looping that negates the possibility of pipelining. Past efforts on high-speed recursive filter structures have been based on block filter structures, where a block of inputs is processed to generate a block of outputs, and the signals are processed in non-overlapping blocks [32]. Although many block recursive filter structures existed for a long time, they were quite complex to implement. Wave digital filters (WDFs), a class of recursive digital filters that are closely related to classical filter networks, have received considerable interest since these structures exhibit a desirable property that the frequency response of these filters is less sensitive to coefficient variation [33] and consequently various approaches to direct VLSI implementation of WDFs have been reported [34]-[36]. Though the WDFs are very easily amenable to VLSI implementation, they cannot be decomposed into modular PEs that can be used for any type of filtering, viz., lowpass, highpass, etc. In other words a lowpass

WDF structure cannot be used for other types of filtering unless special kinds of adaptors are used.

In the past, the effect of limit-cycle oscillations due to quantization has not been taken into consideration in the implementation of recursive, direct-form, digital-filter structures. As a consequence, the application of such recursive filters has been limited. In this thesis we also discuss the implementation of systolic digital-filter structures obtained in direct-form that is efficient in area and in which circuits to circumvent quantization and overflow limit cycles are incorporated.

1.4 Outline of thesis

This thesis is organized in three parts. The first part, comprising Chapters 2 and 3, deals with the mapping of 1-D and M-D digital-filter algorithms onto systolic arrays. The second part, Chapter 4, deals with four different multiplier structures that can be used in the systolic arrays proposed here, in particular, and in architectures used in digital signal processing, in general. The last part, comprising Chapters 5 and 6, deals with the VLSI implementation of some of the multipliers discussed in Chapter 4 and some of the second-order digital filter structures discussed in Chapter 2.

In Chapter 2, two approaches for the mapping of digital filter algorithms onto hardware are discussed. One is based on the SFG and the other on the z-domain characterization of the filter algorithm. In the z-domain approach, any filter algorithm is first transformed into its corresponding z-domain equivalent and by reordering the summation and delays in the transformed equation, several structures are obtained that satisfy the design criteria, such as latency, locality, and modularity. The inconvenience of using the SFG to obtain 1-D recursive filter structures and the efficacy of using the z-domain method are also discussed. We later use only the z-domain approach to derive systolic structures for digital filters that are modular with local data communications.

In Chapter 3, the z-domain approach is extended to the realization of systolic

structures for 2-D, 3-D, and M-D digital filters. All the filter structures obtained are modular and hierarchical. Techniques to circumvent some inherent problems in raster-scanned images, like line and frame wrap-around problems, are also considered. The chapter concludes with a comparison of the various structures.

In Chapter 4, four different multipliers are described. The first multiplier is an area-efficient multiplier that uses only about 50% of the area of a conventional full parallel multiplier. In most signal processing applications, both the input and the output word lengths of a system are the same. An $N \times N$ multiplier produces a product of $2N$ bits, of which only N bits are used. The multiplier designed here avoids the use of all the redundant cells which yield the N bits that are truncated. A correction unit is incorporated that reduces the concomitant error.

The second multiplier is based on the modified octal Booth algorithm. In this algorithm, four bit-segments of the multiplier are scanned and the corresponding operations effected on the multiplicand. In this method, however, a non-trivial multiplication of a number by three is present which is effected as an addition of the number in question with a left-shifted version of the number. This involves an extra delay as a result of this addition. In order to improve the speed of operation we advance a multiplier based on the octal modified Booth algorithm in which the results of the non-trivial operation are precomputed using an external carry look-ahead adder thus avoiding the extra delay.

The third multiplier finds application in the Fermat number-theoretic transform. Here, the numbers, which are integers, are represented in diminished-1 representation. Hitherto diminished-1 multipliers have used translators to convert numbers from their diminished-1 representation to the corresponding binary value. In the multiplier proposed, the use of a translator is circumvented and a novel technique to incorporate this operation of translation in the multiplier structure is described. As a consequence, the area is reduced and the speed of operation of the multiplier is increased.

The conventional way to obtain an inner product is to use a multiplier in con-

junction with an accumulator where each multiplication is followed by an addition of the number stored in the accumulator. The output is obtained when the final pair of numbers are multiplied and added to the result stored in the accumulator. In Chapter 4, we propose a structure that performs an inner product that circumvents the use of an accumulator thereby resulting in increased speed and reduced area.

In Chapter 5, we describe the VLSI implementation of three of the multiplier structures discussed in Chapter 4. These chips have been simulated using SILOS and implemented in 1.2μ CMOS4S technology.

In Chapter 6, we deal with the VLSI implementation of a second-order recursive filter and a single PE proposed in Chapter 2. In the implementation of the second-order digital filter, an iterative multiplier structure has been incorporated which significantly reduces the silicon area of the chip. In both designs, viz., the second-order digital filter and the systolic PE, units to eliminate both quantization and overflow limit cycles have been incorporated. A comparison of the second-order filter built as a single unit and the second-order filter built using a cascade of the PEs has also been carried out in terms of roundoff noise and area $\times$ time complexity.

Conclusions and suggestions for further research are given in Chapter 7.

Chapter 2

Mapping methodology for one-dimensional digital filters

2.1 Introduction

As has been mentioned earlier, there are several mapping methodologies for the mapping of digital-filter algorithms onto hardware. However, we have chosen the signal flow graph (SFG) approach to compare our approach with since it involves matrix transformations that are typical of most of the other approaches. In addition, the SFG serves as a tool for data flow analysis of the underlying algorithm.

In this chapter, several systolic architectures for 1-D nonrecursive and recursive digital filters using the SFG and z-domain approaches are derived. It is shown that the SFG approach is not effective in the sense that it lacks the versatility of the z-domain approach. As a particular application of the z-domain approach, systolic structures suitable for second-order digital filters, decimators, and interpolators are derived.

2.2 Signal flow graph approach

Parallel implementations of an algorithm can be obtained using two approaches viz., vectorizing a sequential algorithm and using recursive equations and sin-

gle assignment codes. Vectorizing compilers process a source code written as a sequential code to generate machine instructions that can be executed in parallel. However, a vectorizing compiler does not rewrite the source code to utilize the inherent concurrent parallelism. There are languages like OCCAM [4] developed for parallel machines that describe a concurrent computing system as a set of independent processes that use locally defined variables and communicate via predefined channels. However, tools are needed to define the concurrency and parallelism within an algorithm before coding it using languages designed for parallel machines.

Dependence graphs (DGs) and SFGs are tools that describe the data flow in an algorithm which allow the hardware designer to study any underlying parallelism. A DG exhibits the parallelism in an algorithm in the form of a regularly repeating pattern of data flow. Through clever manipulation of the data-flow directions, an SFG is derived that can be used to obtain a parallel hardware structure to implement the algorithm.

Before we discuss DGs and SFGs, let us examine the concept of single assignment code [4]. Consider the FORTRAN code for a matrix-vector multiplication $\mathbf{c} = \mathbf{A}\mathbf{b}$ given by

```
      DO 10 N = 1,4
      C(N) = 0.0
      DO 10 K = 1,4
      C(N) = C(N) + A(N,K) * B(K)
10    CONTINUE
```

It can be seen that C(N) is overwritten many times to save storage space. Moreover, C(N) is evaluated after C(N-1) is evaluated. If such a code were to be implemented in hardware, it would result in a design that is inefficient in speed. Algorithms can be described in such a way that each variable is assigned only one value during the execution. This description is said to be in single assignment code form. The above FORTRAN code can be written in the single assignment code form as

```
      DO 10 N = 1,4
      C(N,1) = 0.0
      DO 10 K = 1,4
      C(N,K+1) = C(N,K) + A(N,K) * B(K)
10    CONTINUE
```

Each element in C is assigned only one value. The algorithm is now described in a 2-D index space and at each index point in the space the three variables A, B, and C are defined without any ambiguity.

A DG is a graph $G = [N, A]$ where N is a set consisting of nodes and A is a set comprising arcs or edges. A DG has one node for each of the variables in the algorithm and a directed arc from node $i \in N$ to node $j \in N$ if and only if the variable associated with node j is computed using the variable associated with node i in the algorithm. Since a single assignment code specifies an ordering among the computations in the algorithm, a DG can be thought of as a graph that shows such an ordering among the computations [4]. In other words, a DG is a graphical representation of the single assignment code. A valid processor array structure for implementing the algorithm is obtained by designating one PE for each point in the DG. This, however, leads to inefficient utilization of the PEs, since each PE is active only once during the course of execution of the algorithm. In order to improve the utilization of PEs, it is often desirable to map the nodes of the DG onto a smaller number of PEs. To achieve this it is useful to map the DG onto an intermediate form which is the SFG. An SFG is a graph that shows the dependence of the computations that occur in an algorithm and consists of processing nodes, communicating edges, and delays. In general a node is often denoted by a circle representing an arithmetic or logic operation performed with a zero delay such as multiplication or addition. An edge denotes a dependence relation and possibly a delay. A complete SFG description includes both functional and structural parts. The functional description defines the behaviour within a node, whereas the structural description specifies the interconnection between the nodes. Since an SFG comprises nodes and edges, it is reasonable to think that

all the nodes are connected by edges in some orderly manner. If the DG of the algorithm at hand has an inherent regularity then the nodes and the edges are also connected in a regular fashion. We shall use the SFG approach and the z-domain approach for the design of digital-filter structures and show how the SFG approach becomes unsuitable as the dimensionality of the filter increases.

2.3 One-dimensional nonrecursive digital filters

A nonrecursive filter is a discrete-time system in which any sample $y(n)$ of the output sequence $\{y(n)\}$ is explicitly determined as a weighted sum of a finite number of samples of the input sequence $\{x(n)\}$. Mathematically, a nonrecursive filter can be represented as [37]

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k) \quad n \geq 0 \tag{2.1}$$

where N is the length of the filter. In order to facilitate the derivation of architectures in which all PEs are followed by a storage unit, (2.1) is modified to

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-1-k) \tag{2.2}$$

or equivalently,

$$y(n+1) = \sum_{k=0}^{N-1} h(k)x(n-k) \tag{2.3}$$

The DG of (2.3) is shown in Fig. 2.1 where both the inputs and the weights of the filter are transmitted or broadcast throughout the index space (n, k).

All digital-filter algorithms have an inherent regularity and parallelism within them and it is known that a regular iterative algorithm can always be expressed in a single assignment code form [5].

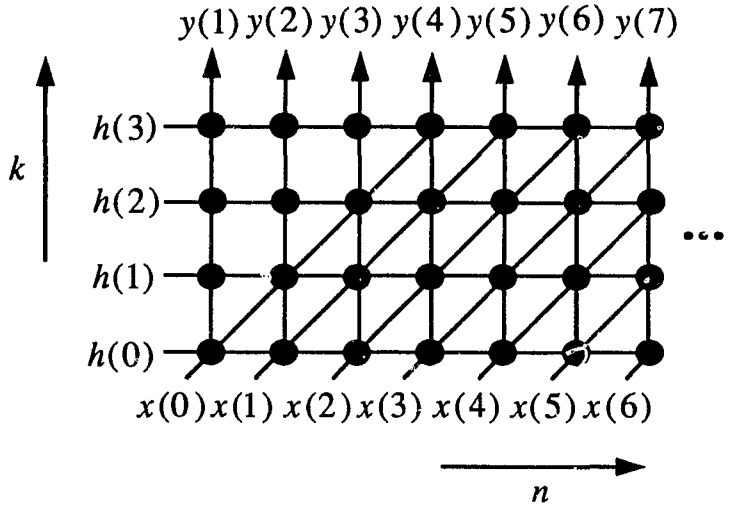


Figure 2.1: Dependence graph of (2.3).

The first step in the systolic implementation of (2.3) is to convert it to a single assignment code as

$$\begin{aligned} y_n &= \sum_{k=0}^{N-1} h_0^k x_n^k \qquad (2.4) \\ x_n^k &= x_{n-1}^{k-1}, \quad x_n^0 = x(n), \quad x_0^k = 0 \quad \text{for} \quad k > 0 \\ h_0^k &= h(k) \\ y(n+1) &= y_n, \quad y(0) = 0 \end{aligned}$$

for $n,\ k\ =\ 0,\ 1,\ \cdots,\ N-1$. The SFG of (2.4) is shown in Fig. 2.2 for a filter of length four, where (n, k) represents the index space.

Figure 2.2 also shows the scheduling vector $\vec{s}$ and the projection vector $\vec{d}$. The choice of $\vec{d}$ determines the array configuration. All nodes lying along a straight line parallel to $\vec{d}$ are assigned to one PE. The projection vector is always in the direction of the extremal ray which results in systolic architectures of lower dimension than the SFG [10]. The choice of $\vec{s}$ is such that the following two conditions are satisfied [4]

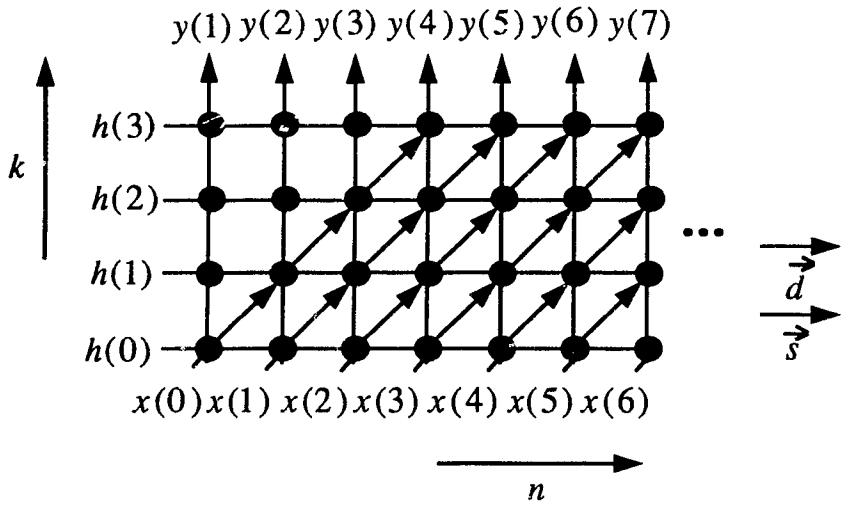


Figure 2.2: Signal flow graph of (2.4).

$$\vec{s}^T \cdot \vec{e} \geq 0 \quad \forall \vec{e} \tag{2.5}$$

$$\vec{s}^T \cdot \vec{d} \neq 0 \tag{2.6}$$

where $\vec{e}$ represents any arc in the SFG. Equation (2.5) states that all dependence arcs flow in the same direction across the hyperplanes which ensures causality in any scheduling scheme. Equation (2.6) states that hyperplanes cannot be parallel to the projection vector which ensures that the nodes on a hyperplane will not be projected onto the same processor. However, it can be seen in Fig. 2.2 that $\vec{s}^T \cdot \vec{e} = 0$ for the vertical arcs are involved in the generation of $y(n)$. Thus mapping this SFG onto an array will lead to the semi-systolic structure of Fig. 2.3(a) where each block represents a PE of the type depicted in Fig. 2.3(b). It is evident from the SFG that all the partial sums for each output have to be added simultaneously.

A different systolic structure is obtained if (2.3) is converted into the single assignment code

$$y_n^k = y_n^{k+1} + h_0^k x_n^k \tag{2.7}$$

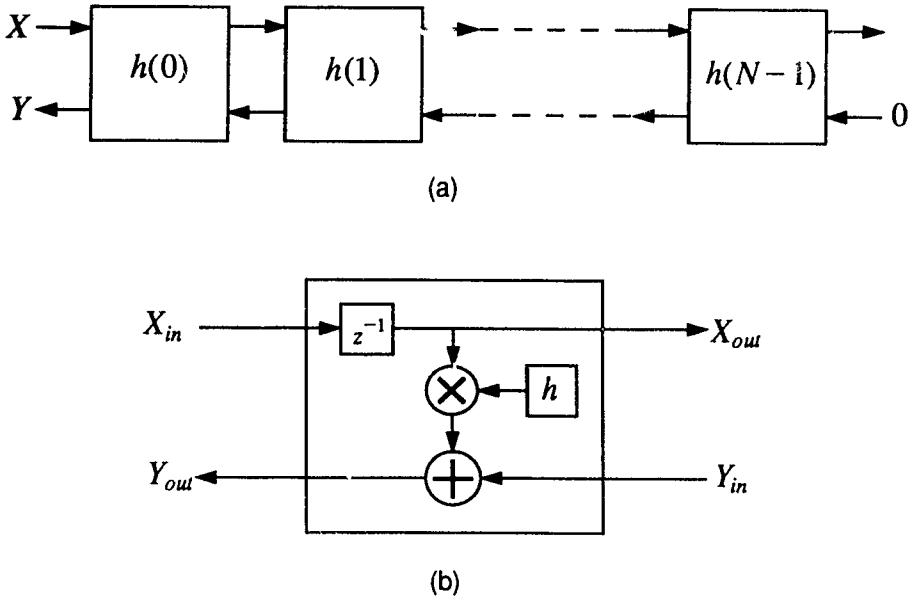


Figure 2.3: Mapping of the signal flow graph of (2.3) onto a systolic array. (a) The systolic structure. (b) Details of PE involved.

$$\begin{aligned} x_n^k &= x(n-k) \\ h_0^k &= h(k) \\ y(n+1) &= y_n^0, \quad y_n^N = 0 \end{aligned}$$

for $n,\ k\ =\ 0,\ 1,\ \cdots,\ N-1$. The SFG of (2.7) is shown in Fig. 2.4.

Evidently, the input data $x(n)$ as well as the filter weights $h(n)$ are broadcast. Choosing the projection vector $\vec{d} = [1\ 0]^T$ and the scheduling vector $\vec{s} = [1\ -1]^T$, we obtain the systolic architecture of Fig. 2.5. Figure 2.5(a) shows the mapping of the above equation onto a systolic architecture while Fig. 2.5(b) shows the details of each PE. The z^{-1} block in the figure represents a unit delay.

By introducing a delay of N sampling periods, (2.1) can be written as

$$y(n+N) = \sum_{k=0}^{N-1} h(k)x(n-k) \tag{2.8}$$

A different systolic structure is obtained if (2.8) is converted into the single as-

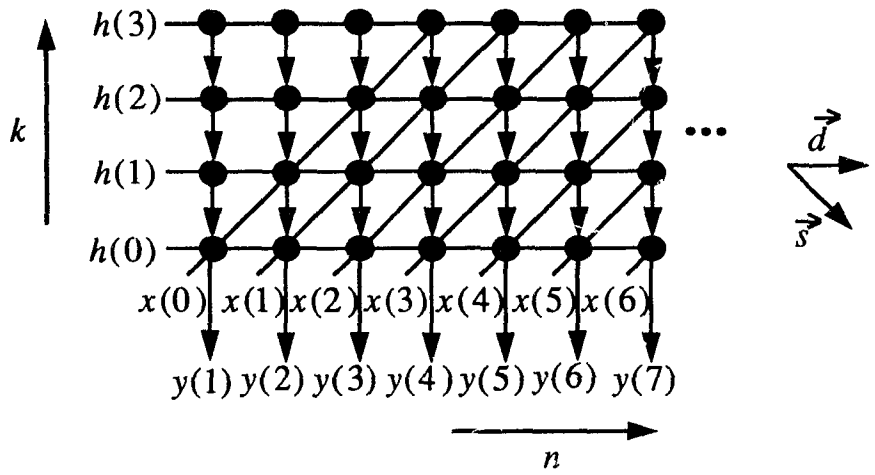


Figure 2.4: Signal flow graph of (2.7) in which filter inputs are broadcast and outputs are propagated.

signment code

$$y_n^k = y_n^{k-1} + h_0^k x_n^k \tag{2.9}$$

where

$$\begin{aligned} x_n^k &= x_{n-1}^{k-1}, \quad x_n^0 = x(n), \quad x_0^k = 0, \quad \text{for} \quad k > 0 \\ h_0^k &= h(k) \\ y(n+N) &= y_n^{N-1}, \quad y_n^{-1} = 0 \end{aligned}$$

The modified SFG is shown in Fig. 2.6 for a filter of length four. We see that all signal communications are restricted to nearest neighbours. The figure also shows the scheduling vector $\vec{s}$ and the projection vector $\vec{d}$, where the hyperplanes (dotted lines) represent different time instants. Choosing a projection vector $\vec{d} = [1\ 0]^T$ and the scheduling vector $\vec{s} = [1\ 1]^T$, we obtain the systolic structure shown in Fig. 2.7.

Some systolic arrays and pipelined computing structures have local memory to store data for use in subsequent operations. The local memory is usually in

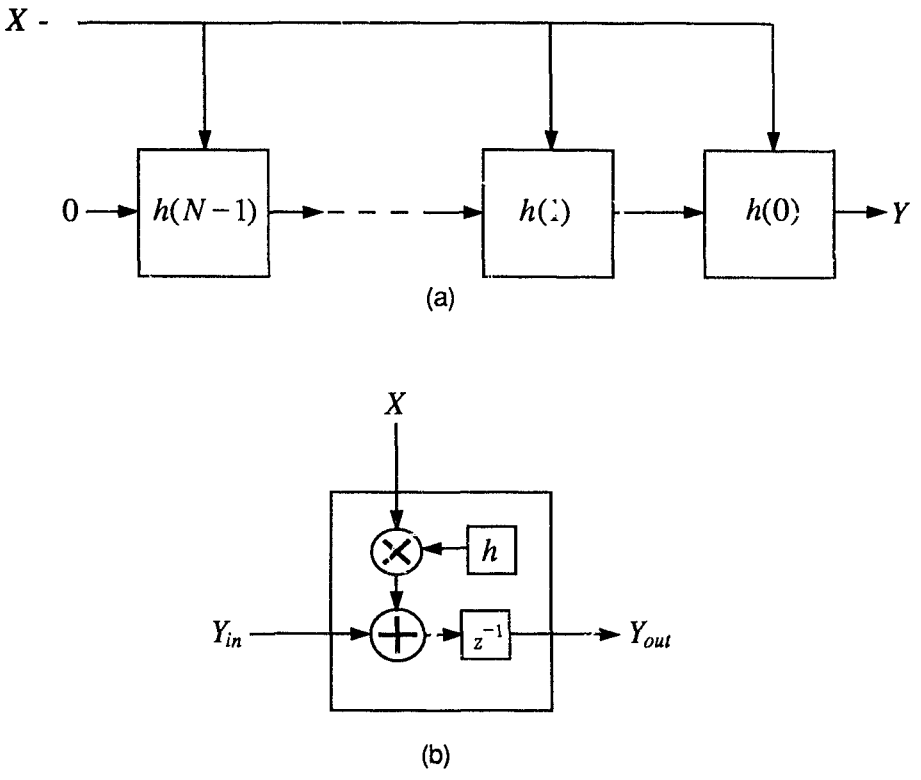


Figure 2.5: Mapping of the signal flow graph of (2.4) onto a systolic array (a) The systolic structure. (b) Details of PE involved.

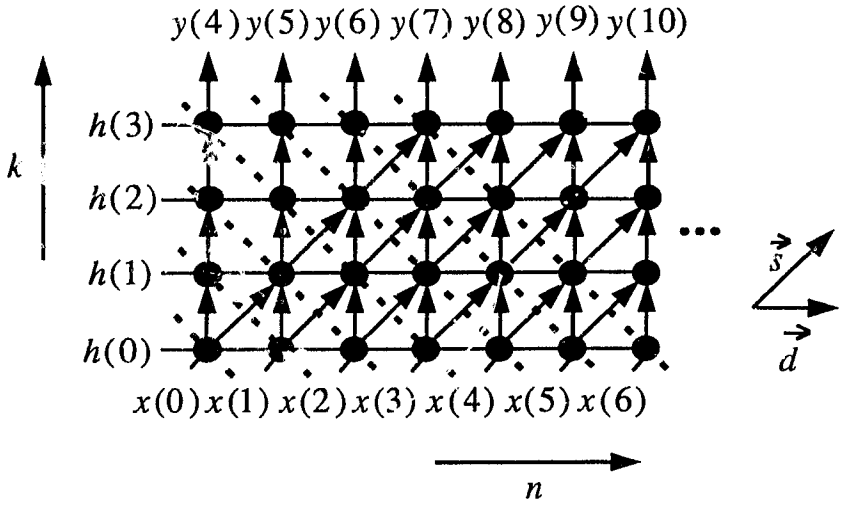


Figure 2.6: Signal flow graph of (2.9) in which filter inputs and outputs are propagated.

the form of first-in-first-out (FIFO) elements of linearly or geometrically increasing sizes [38]. It would be interesting to see if such architectures can support nonrecursive filtering. Below we show that this indeed is the case [39].

Equation (2.8) can be expressed as

$$y_n^k = y_n^{k-1} + p_{n-k}^k \tag{2.10}$$

where

$$p_n^k = p_{n-1}^k, \quad x_n^k = x_n^{k-1}, \quad x_n^0 = x(n),$$
$$h_n^k = h_{n-1}^k, \quad h_0^k = h(k), \quad p_{n-k}^k = h_{n-k}^k \, x_{n-k}^k$$

for all $n, k \geq 0$. The input samples are propagated vertically as shown in Fig. 2.8 and the partial products for a particular output are evaluated at the same instant. This can be seen by examining the node activities at each hyperplane. Choosing the projection vector $\vec{d} = [1\ 0]^T$ and a scheduling vector $\vec{s} = [1\ 1]^T$, we get the systolic array illustrated in Fig. 2.9 for a filter of length four. The zeroth PE in Fig. 2.9 can be represented by

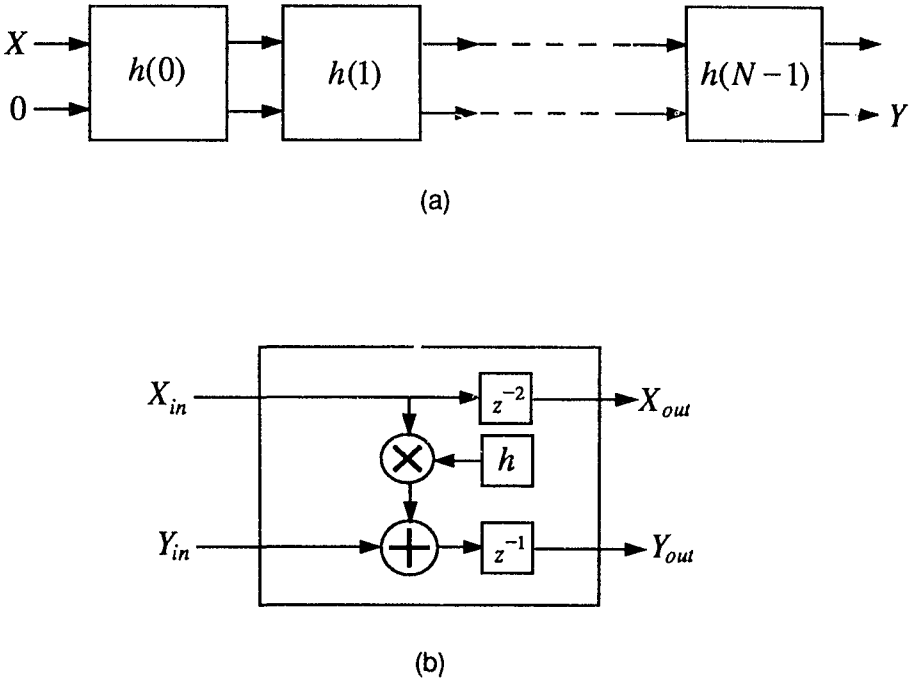


Figure 2.7: Mapping of (2.9) onto a systolic architecture. (a) The systolic array. (b) Details of the PE involved.

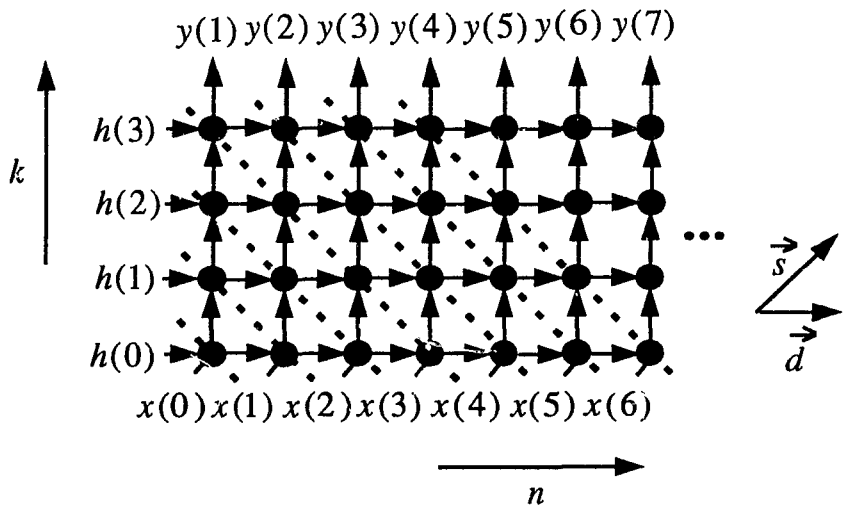


Figure 2.8: Signal flow graph of the modified form of Fig. 2.6 showing redirection of the filter inputs and the formation of partial products at the hyperplanes.

$$y_n^0 = y_n^{-1} + p_{n-1}^0, \quad y_n^{-1} = 0,$$
$$p_{n-1}^0 = h(0)x_{n-1}^0, \quad x_n^1 = x_n^0$$

On the other hand, the kth PE for $k = 1, 2, \cdots, N-1$, can be represented by

$$y_n^k = y_n^{k-1} + p_{n-k}^k, \quad p_{n-k}^k = p_{n-k+1}^k,$$
$$p_{n-k+1}^k = p_{n-k+2}^k, \quad \cdots, \quad p_{n-2}^k = p_{n-1}^k,$$
$$p_{n-1}^k = h(k)x_{n-1}^k, \quad x_n^{k+1} = x_n^k$$

At any time instant, the PEs of the array produce all the partial products for a particular output sample and each PE stores one of the filter weights. In each PE, there is a FIFO memory whose size increases linearly with the position of the PE. Thus the topmost PE will be associated with a FIFO memory of length $N-1$. A triangular systolic architecture based on Fig. 2.8 is shown in Fig. 2.10 for a filter of length four. As can be seen from the figure, the systolic array has three different types of cells.

In the following section we shall describe the z-domain approach and explain

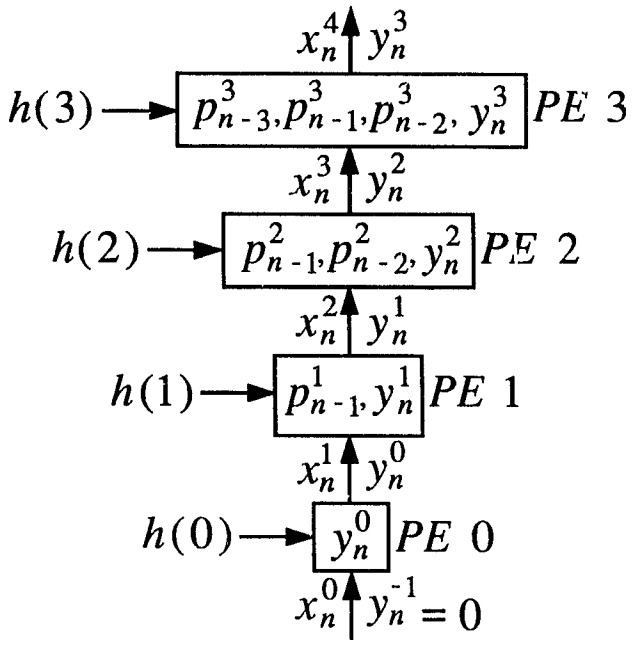


Figure 2.9: Systolic realization of the signal flow graph of Fig. 2.8

how the filter structures derived using the SFG method can be obtained using the z-domain approach.

2.4 z-domain approach

For any mathematical expression or algorithm involving subscripted variables, which conceptually represent data sequenced in time or space, a new expression in terms of a well-defined shift operator can be used to model displacements in time or shifts in space. Symbolic manipulations can then be used to transform the derived algorithm into equivalent ones by using the properties of the shift and the functional operators in the expression. The z-domain approach for mapping digital filter algorithms onto systolic hardware can be described as follows:

1. Any given filter algorithm is first transformed into its z-domain equivalent.

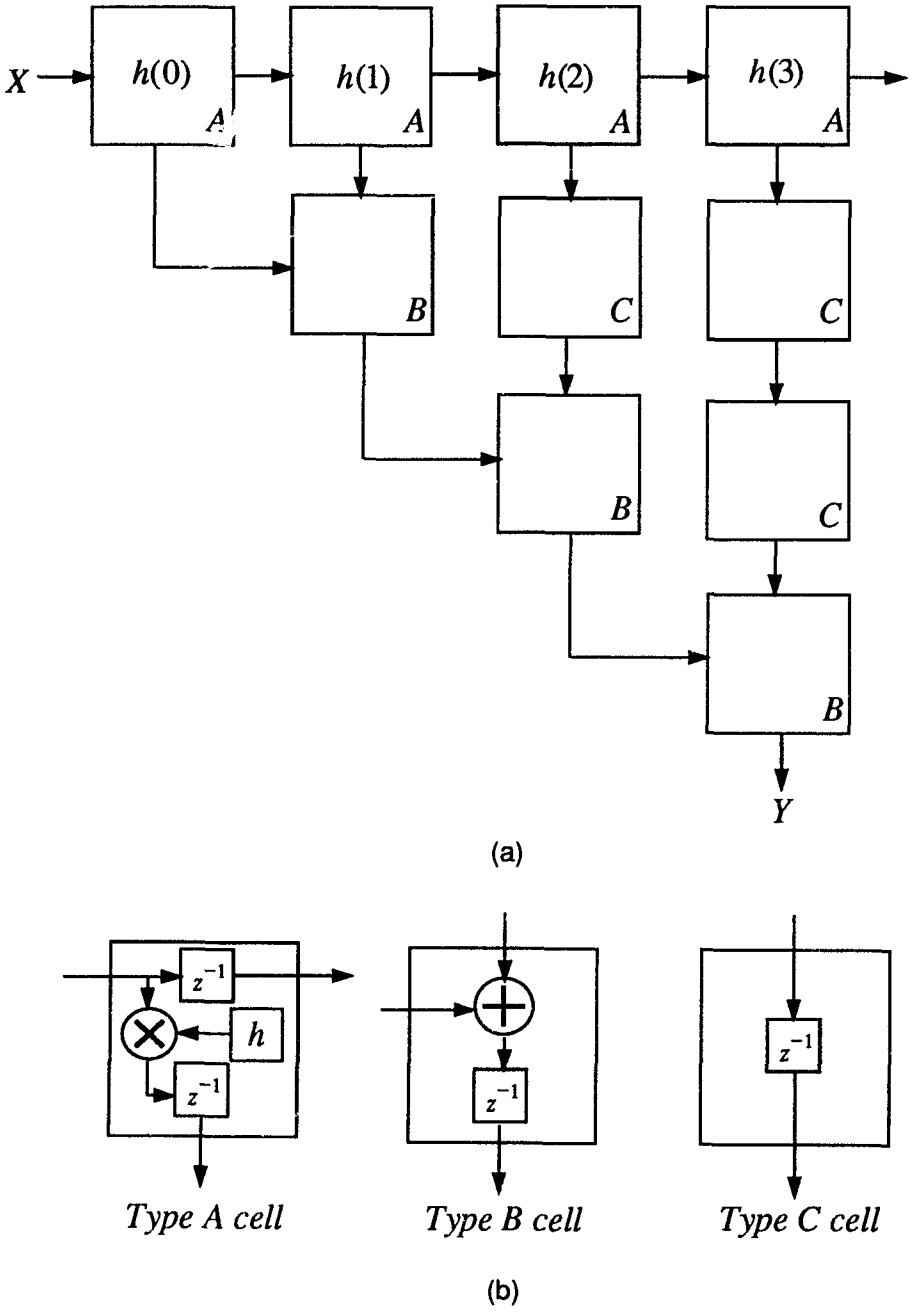


Figure 2.10: Systolic realization of the signal flow graph of Fig. 2.8.

2. By using Horner's rule or other polynomial evaluation techniques, a set of recursive expressions is derived.

3. From the recursive expressions, the structures of the processing element and the systolic array are obtained.

4. By reordering the shift and functional operators in the filter equations, different recursive expressions and, consequently, different systolic structures can be derived.

To illustrate the approach, (2.2) can be written in the z-domain as

$$\begin{aligned} Y &= \sum_{k=0}^{N-1} h(k) z^{-(k+1)} X \\ &= h(0)Xz^{-1} + h(1)Xz^{-2} + \cdots + h(N-1)Xz^{-N} \\ &= h(0)X_0 + h(1)X_1 + \cdots + h(N-1)X_{N-1} \end{aligned} \tag{2.11}$$

where

$$\begin{aligned} X_k &= z^{-1}X_{k-1} \qquad 1 \leq k \leq N-1 \\ X_0 &= z^{-1}X \end{aligned} \tag{2.12}$$

with $Y \equiv Y(z)$ and $X \equiv X(z)$.

Alternatively (2.11) can be rewritten as a set of recursions

$$\begin{aligned} Y_i &= h_i X_i + Y_{i+1} \qquad 0 \leq i \leq N-1 \\ Y_N &= 0 \\ Y &= Y_0 \end{aligned} \tag{2.13}$$

Equation (2.11) shows the transformed filter equation and (2.13) shows the recursive expression derived from (2.11). The mapping of (2.11) onto a linear systolic structure is the same as that of Fig. 2.3 and the PE structure can easily be obtained from (2.13). It can be seen from (2.12) that the input data is propagated

from one PE to the next. The addition of all the partial products is performed simultaneously.

The characteristics of the systolic structure obtained using the z-domain approach can easily be deduced from the recursive expressions by noting the following aspects:

1. Each iteration is assigned a PE to perform it.
2. The number of PEs is determined by the number of iterations required to produce the final output.
3. PE complexity and computational load is dictated by the nature of the recursive expressions.
4. Chip area is determined by the number and complexity of each of the PEs.
5. Communication requirements are determined by studying the source of the data used on the right-hand side of each recursive expression.
6. Buffer size is determined by the sum of all the powers of z^{-1} in each term on the right-hand side of each recursive expression.
7. The processing rate is determined from the computational load and complexity of the PEs.
8. System latency is determined by the highest power of z^{-1} in the recursive expression that deals with the desired output.

Processing rate is defined as the rate at which data are processed by each PE and may be equal to the reciprocal of the multiply time and/or the add time in each PE depending on the actual implementation. Latency is defined as the time elapsed between the application of the first sample of the input and the appearance of the first sample of the output. In a systolic structure, it is not possible to feed an

input and obtain the corresponding output during the same cycle. As a result, the minimum latency in these structures is one sampling period.

We could arrive at the systolic architecture of Fig. 2.5 using the z-domain approach by rewriting (2.2) as

$$Y = z^{-1}(h(0)X + z^{-1}(h(1)X + \cdots + z^{-1}(h(N-1)X)\cdots)) \tag{2.14}$$

The above equation can be expressed as a recursive relation of the form

$$Y_i = z^{-1}[h(i)X + Y_{i+1}] \quad \text{for } i = N-1,\ N-2\ ,\cdots,\ 0 \tag{2.15}$$

where

$$Y_N = 0$$
$$Y = Y_0$$

The mapping of (2.14) onto a systolic architecture results in the same structure as that shown in Fig. 2.5. Processor i receives X and Y_{i+1} as inputs and evaluates Y_i after one sampling period. It can be seen from (2.15) that the components of the PE which effect the operations in (2.15) comprise a multiplier, an adder, and a delay element.

Using the z-domain approach, the systolic structure of Fig. 2.7 can be obtained by writing the filter equation as

$$\begin{aligned} z^{-(N-1)}Y &= \sum_{k=0}^{N-1} z^{-(N-1-k)}h(k)z^{-(2k+1)}X \\ &= z^{-1}(h(N-1)X_{N-1} + z^{-1}(h(N-2)X_{N-2} \\ &\quad + \cdots + z^{-1}(h(0)X_0)\cdots)) \end{aligned} \tag{2.16}$$

where $X_k = z^{-2}X_{k-1}$, for $k = 1,\ 2,\ \cdots,\ N-1$ and $X_0 = X$. Equation (2.16) can be expressed using recursive relations as

$$Y_i = z^{-1}[h(i)X_i + Y_{i-1}] \quad 0 \le i \le N-1 \tag{2.17}$$
$$Y_{-1} = 0$$
$$z^{-(N-1)}Y = Y_{N-1}$$

The mapping of (2.16) onto a systolic architecture is shown in Fig. 2.7(a) while the details of the PE is shown in Fig. 2.7(b).

The structure of Fig. 2.10 can also be obtained by writing (2.16) as

$$\begin{aligned} z^{-(N-1)}Y &= z^{-1}\left(\cdots z^{-1}\left(z^{-1}\left(h(0)X\right) + z^{-1}\left(h(1)Xz^{-1}\right)\right) + \cdots \right. \\ &\quad \left. + z^{-(N-1)}\left(h(N-1)Xz^{-(N-1)}\right)\right) \end{aligned} \tag{2.18}$$

A set of recursive relations can similarly be obtained for the above equation.

2.5 One-dimensional recursive digital filters

In [4] it is mentioned that many RIAs can be represented in terms of SFGs. For a nonrecursive filter, this is easily accomplished. A recursive filter can be thought of as a combination of two nonrecursive filters in which the output of one filter is fed to the other. Thus, a recursive filter equation is represented as an overlap of two SFGs, each representing one nonrecursive filter equation. Therefore, the resulting SFG is complex and it is difficult to obtain the projection and scheduling vectors. In addition, as will be shown in the following chapter, the complexity of the SFG approach increases as the dimension of the filter increases and it becomes very difficult to obtain systolic structures. As a result, the SFG is not an effective tool in the mapping of recursive digital filter algorithms onto hardware.

In this section, we use the z-domain approach for obtaining several systolic arrays for recursive filters that are modular with regular interconnections. Let us consider a two-input nonrecursive digital filter characterized by the equation

$$Y = \sum_{k=0}^{N-1} a(k)X_1 z^{-(k+1)} - \sum_{k=0}^{N-1} b(k)X_2 z^{-(k+1)} \tag{2.19}$$

where $Y \equiv Y(z)$, $X_1 \equiv X_1(z)$, $X_2 \equiv X_2(z)$ and N is the length of the filter. Again, the filter output is delayed deliberately by one sample to facilitate the derivation of structures in which all the PEs are followed by a storage unit. The

reason for choosing a two-input nonrecursive filter will become clear in the next chapter in the context of 2-D recursive filter implementations. A recursive filter can be obtained by suitably rearranging the coefficients and making $Y = X_2$.

Equation (2.19) can be written as

$$Y = \sum_{k=0}^{N-1} z^{-(k+1)}[a(k)X_1 - b(k)X_2] \tag{2.20}$$

If each term inside the square brackets is realized by a PE, then a structure is obtained in which the input signals $x_1(n)$ and $x_2(n)$ are broadcast to all the PEs, and the output of each PE is delayed by a different amount. The filter structure resulting from (2.20) is not systolic.

To overcome the use of unequal delays standard algebraic techniques are used. Applying Horner's rule to (2.20) yields

$$\begin{aligned} Y &= z^{-1}\Big[(a(0)X_1 - b(0)X_2) + z^{-1}\left[(a(1)X_1 - b(1)X_2)\right. \\ &\quad \left. + \cdots + z^{-1}\left[(a(N-1)X_1 - b(N-1)X_2)\right]\cdots\right]\Big] \end{aligned} \tag{2.21}$$

The above equation can be written as

$$\begin{aligned} Y_i &= z^{-1}\left[a(i)X_1 - b(i)X_2 + Y_{i+1}\right] \qquad 0 \leq i \leq N-1 \\ Y_N &= 0 \\ Y &= Y_0 \end{aligned} \tag{2.22}$$

The ith PE computes Y_i in (2.22) using two multiplications and two additions. Figure 2.11(a) shows the mapping of (2.21) onto a systolic structure while Fig. 2.11(b) describes the details of the PE involved. It can be seen that the structure is modular and the adder in each PE adds only three inputs at any time [37].

The input signals in the previous implementation are broadcast to all the PEs. We can obtain an implementation in which the input signals are propagated from

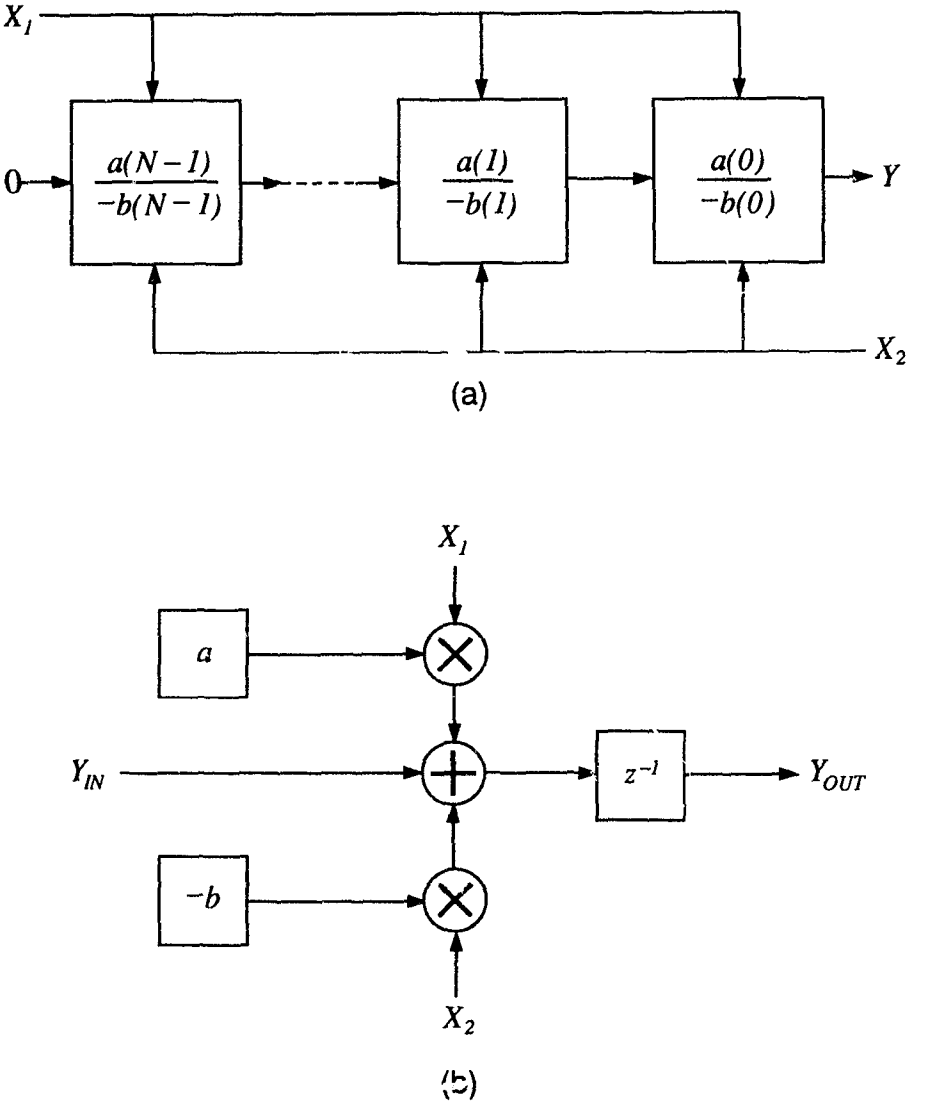


Figure 2.11: Mapping of (2.21) onto a systolic architecture. (a) The systolic array. (b) Details of PE involved.

one PE to the next by writing (2.19) as

$$Y = \sum_{k=0}^{N-1} \left[a(k)(X_1 z^{-(k+1)}) - b(k)(X_2 z^{-(k+1)}) \right] \tag{2.23}$$

If each term inside the square brackets is assigned to a PE, then the inputs to each PE are delayed by different amounts and the PE outputs are added simultaneously. The structure of the filter will contain unequal delay elements and the adder will be required to add more than two inputs at any time.

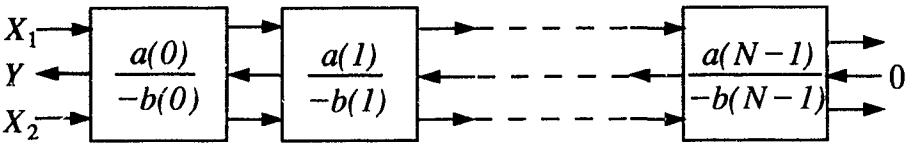
If we introduce intermediate variables X_{1k} and X_{2k} given by

$$\begin{aligned} X_{1k} &= z^{-1} X_{1(k-1)} \\ X_{2k} &= z^{-1} X_{2(k-1)} \end{aligned}$$

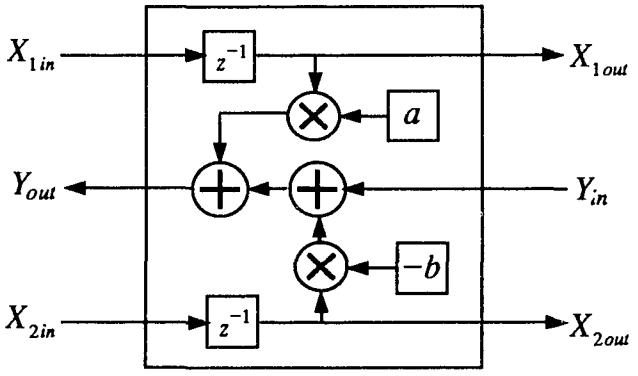
for $k = 1, 2, \cdots, N-1$ and $X_{10} = z^{-1} X_1$ and $X_{20} = z^{-1} X_2$, then (2.23) becomes

$$\begin{aligned} Y &= \left[a(0)(z^{-1} X_1) - b(0)(z^{-1} X_2) \right] \\ &\quad + \left[a(1) z^{-1} (z^{-1} X_1) - b(1) z^{-1} (z^{-1} X_2) \right] \\ &\quad + \cdots + \left[a(N-1) z^{-1} (z^{-(N-1)} X_1) - b(N-1) z^{-1} (z^{-(N-1)} X_2) \right] \\ &= [a(0) X_{10} - b(0) X_{20}] + [a(1) X_{11} - b(1) X_{21}] \\ &\quad + \cdots + \left[a(N-1) X_{1(N-1)} - b(N-1) X_{2(N-1)} \right] \end{aligned} \tag{2.24}$$

If each PE performs the operations inside each pair of square brackets, then from (2.24) it can be seen that the inputs are propagated from one PE to the next. Thus the delay registers are all of the same length. Although the delays in (2.24) are all equal, the filter output is produced after all the partial products have been added. Figure 2.12(a) shows the mapping of (2.24) while Fig. 2.12(b) shows the details of the PE involved. It can be seen from Fig. 2.12(b) that each PE has two adders and thus the output of the filter, for a length of N, will incur a delay of $2NT_a$, where T_a is the adder delay.



(a)



(b)

Figure 2.12: Mapping of (2.24) onto a systolic architecture. (a) The systolic array. (b) Details of PE involved.

We can obtain implementations in which both the input and output signals are pipelined. This scheme involves the use of more complex PEs. For the case when N is even, (2.19) can be written as

$$\begin{aligned} Y &= z^{-1}\{[a(0)X_{10} - b(0)X_{20} + a(1)X_{11} - b(1)X_{21}] \\ &\quad + z^{-1}\{[a(2)X_{11} - b(2)X_{21} + a(3)X_{12} - b(3)X_{22}] \\ &\quad + \cdots + z^{-1}\{[a(N-2)X_{1p} - b(N-2)X_{2p} \\ &\quad + a(N-1)X_{1q} - b(N-1)X_{2q}]\}\cdots\}\} \end{aligned} \tag{2.25}$$

where

$$\begin{aligned} X_{1k} &= z^{-1}X_{1(k-1)} \\ X_{2k} &= z^{-1}X_{2(k-1)} \end{aligned}$$

for $k = 1, 2, \cdots, p, q$, where $p = (N-2)/2$ and $q = N/2$, and $X_{10} = X_1$, and $X_{20} = X_2$. Equation (2.25) can be written as

$$\begin{aligned} Y_i &= z^{-1}\left[a(2i)X_{1i} - b(2i)X_{2i} + a(2i+1)X_{1(i+1)}\right. \\ &\quad \left. - b(2i+1)X_{2(i+1)} + Y_{i+1}\right], \quad 0 \le i \le N/2 - 1 \end{aligned} \tag{2.26}$$

$$\begin{aligned} Y_q &= 0 \\ Y &= Y_0 \end{aligned}$$

The ith PE is assigned to compute Y_i in (2.26). Each PE is required to perform four multiplications and four additions, and has three inputs and three outputs. The three inputs are the two signal inputs and the partial sum from the adjacent PE. The outputs are the two delayed signal inputs and the partial sum. The mapping of (2.26) onto a systolic structure is shown in Fig. 2.13(a) while Fig. 2.13(b) depicts the details of the PE involved.

Similarly we can derive systolic structure for the case when N is odd. The recursive relation can be written as

$$Y_i = a(2i)X_{1(i+1)} - b(2i)X_{2(i+1)} + z^{-1}\left[a(2i+1)X_{1(i+1)}\right.$$

$$-b(2i+1)X_{2(i+1)} + Y_{i+1}\Big] \qquad 0 \leq i \leq (N-3)/2 \tag{2.27}$$

$$Y_r = a(N-1)X_{1s} - b(N-1)X_{2s}$$

$$Y = Y_0$$

where

$$X_{1k} = X_{1(k-1)}$$

$$X_{2k} = X_{2(k-1)}$$

for $k = 0, 1, \cdots, r, s$, where $r = (N-1)/2$ and $s = (N+1)/2$. The mapping of (2.27) onto a PE is shown in Fig. 2.13(c).

Several systolic structures can be obtained by manipulating summations and delays in the filter equation. Below some systolic structures for second-order recursive filters, decimators, and interpolators are presented.

2.6 Examples

2.6.1 Second-order recursive filters

The traditional method of reducing roundoff noise in digital filters is to decompose an Nth-order into a cascade or parallel connections of second-order (and possibly first-order) sections. The idea is to separate closely spaced poles and zeros into different sections of the filter. These second-order sections are most often realized in the direct form.

The transfer function of a second-order recursive filter is given by

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \tag{2.28}$$

The above equation can be written in terms of the z transforms of the input and output as

$$Y = a_0 X z^{-1} + a_1 X z^{-2} + a_2 X z^{-3} - b_1 Y z^{-1} - b_2 Y z^{-2} \tag{2.29}$$

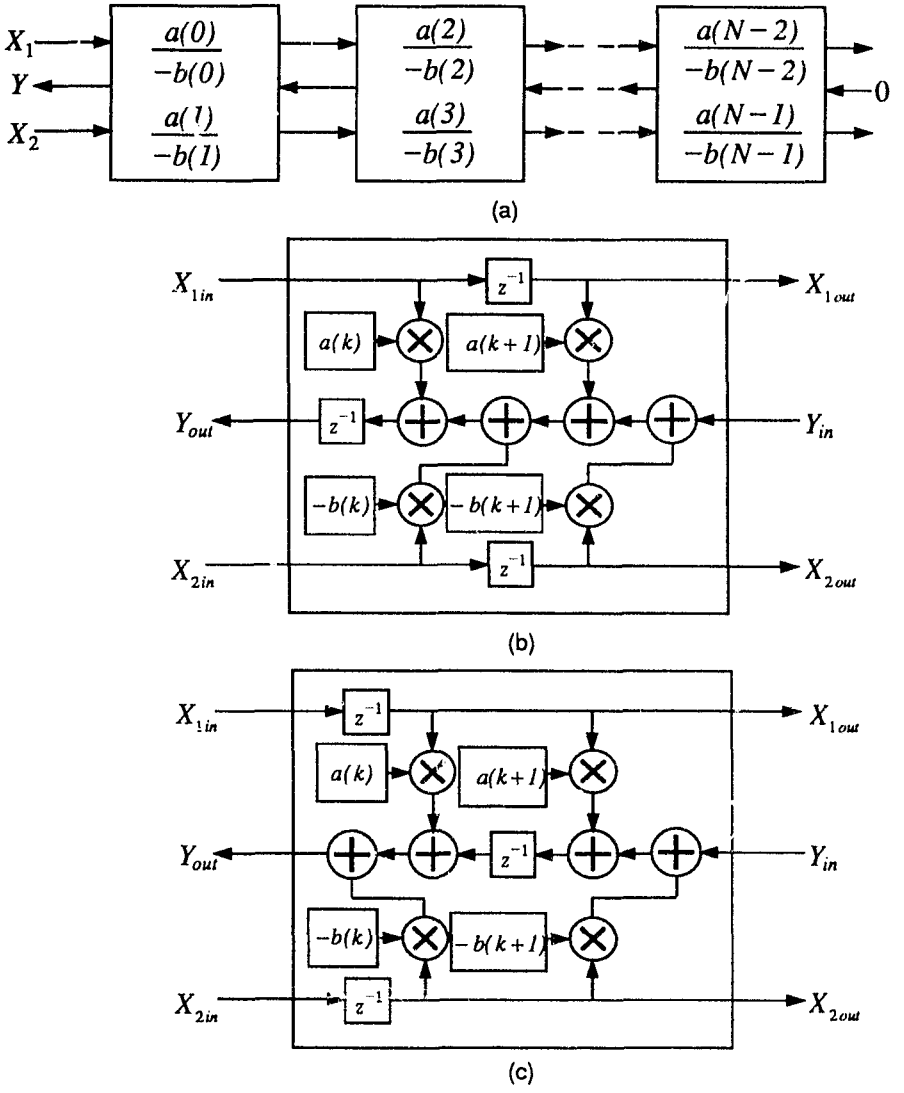


Figure 2.13: Mapping of (2.26) and (2.27) onto systolic architectures. (a) The systolic array. (b) Details of PE involved in mapping (2.26). (c) Details of PE involved in mapping (2.27).

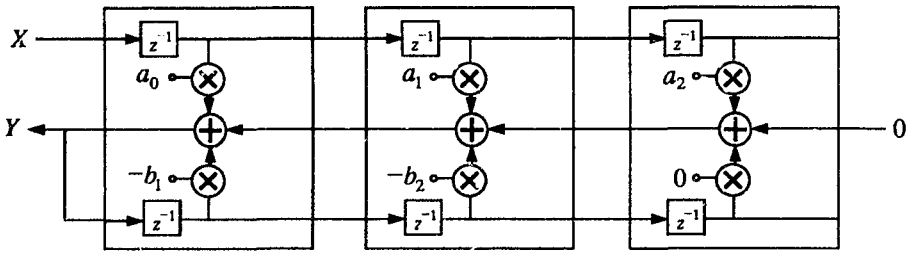


Figure 2.14: Mapping of (2.30) onto a systolic array.

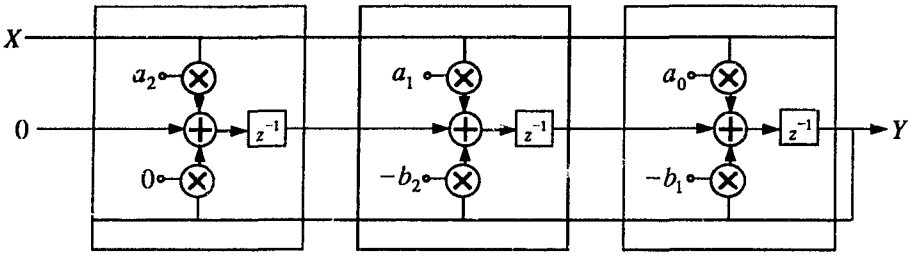


Figure 2.15: Mapping of (2.31) onto a systolic array.

where the output is delayed by one sample. By expressing (2.29) in a number of ways, several systolic filter structures can be obtained.

Equation (2.29) can also be written as

$$\begin{aligned} Y &= a_0X_0 + a_1X_1 + a_2X_2 - b_1Y_0 - b_2Y_1 \\ &= (a_0X_0 - b_1Y_0) + (a_1X_1 - b_2Y_1) + (a_2X_2) \end{aligned} \tag{2.30}$$

where the X_i's, Y_i's are defined as before. If the expression in each set of brackets is realized by a PE, the mapping of (2.30) yields the structure shown in Fig. 2.14 where all the PEs are of the same type.

By rewriting (2.29) as

$$Y = z^{-1}((a_0X - b_1Y) + z^{-1}((a_1X - b_2Y) + z^{-1}(a_2X))) \tag{2.31}$$

a systolic structure of Fig. 2.15 is obtained in which three PEs are used.

The recursive filter equation (2.29) can be written as

$$Y = z^{-1}\{(a_0X - b_1Y) + (a_1X_0 - b_2Y_0) + (a_2X_1)\} \tag{2.32}$$

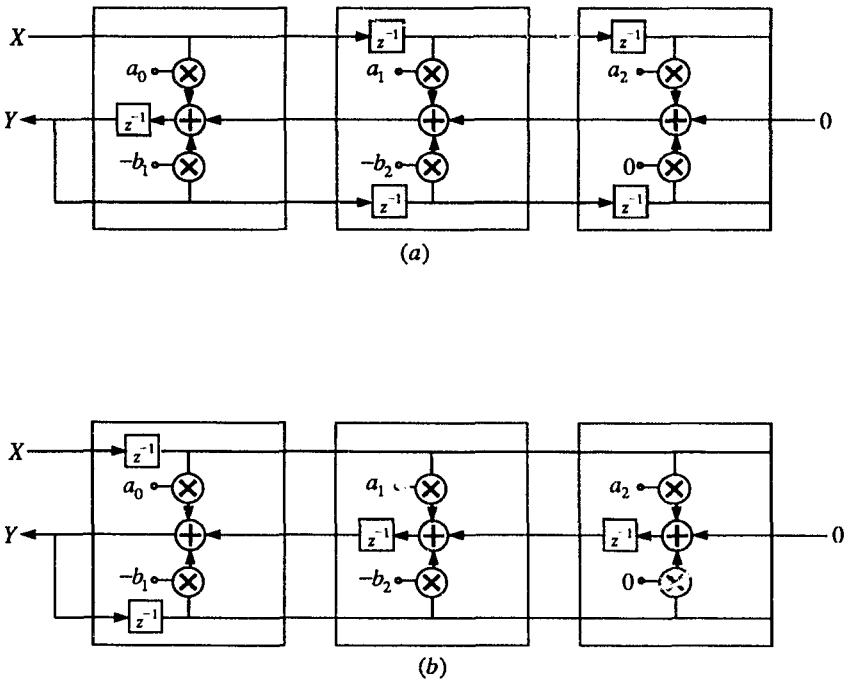


Figure 2.16: (a) Mapping of (2.32) onto a systolic array. (b) Mapping of (2.33) onto a systolic array.

and can be realized as shown in Fig. 2.16(a). As can be seen, a structure with two types of PEs is obtained. In the first PE, the signals $x(n)$ and $y(n)$ are broadcast and the partial product is pipelined. In the other PEs, the signals $x(n)$ and $y(n)$ are pipelined and the partial products are added asynchronously.

By rewriting (2.29) as

$$Y = (a_0X_0 - b_1Y_0) + z^{-1}\left\{(a_1X_0 - b_2Y_0) + z^{-1}(a_2X_0)\right\} \tag{2.33}$$

the types of PEs used in the structure of Fig. 2.16(a) can be interchanged as depicted in Fig. 2.16(b).

In all the above second-order filter structures, either the input signal or the output signal is pipelined, but not both. In what follows we obtain structures in which both the input and output signals are pipelined. By multiplying the

numerator and denominator of the transfer function given by (2.28) with a first-order polynomial of the type [40]

$$C(z) = 1 + c_1 z^{-1}$$

we get

$$H(z) = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2} + p_3 z^{-3}}{1 + q_1 z^{-1} + q_2 z^{-2} + q_3 z^{-3}} \tag{2.34}$$

where

$$\begin{aligned} p_0 &= a_0 \\ p_1 &= a_1 + a_0 c_1 \\ p_2 &= a_2 + c_1 a_1 \\ p_3 &= a_2 c_1 \\ q_1 &= b_1 + c_1 \\ q_2 &= b_2 + b_1 c_1 \\ q_3 &= b_2 c_1 \end{aligned}$$

To assure realizability, q_2 has to be equal to zero. Thus we require $c_1 = -b_2/b_1$ and $b_1 \neq 0$. To assure stability we require $\mid b_2/b_1 \mid < 1$.

We can now write (2.34) as

$$\begin{aligned} Y &= p_0 X z^{-1} + p_1 X z^{-2} + p_2 X z^{-3} + p_3 X z^{-4} - q_1 Y z^{-1} - q_3 Y z^{-3} \\ &= z^{-1} \left\{ (p_0 X_0 + p_1 X_1 - q_1 Y_0) + z^{-1} \left[(p_2 X_1 + p_3 X_2 - q_3 Y_1) \right] \right\} \end{aligned} \tag{2.35}$$

where the output is delayed by one sample. The mapping of (2.35) results in the structure shown in Fig. 2.17 where all the signals, including partial products, are pipelined.

Since there are four coefficients in the numerator of (2.34) we can introduce a latency of four sampling periods in order to pipeline all the signals in each PE.

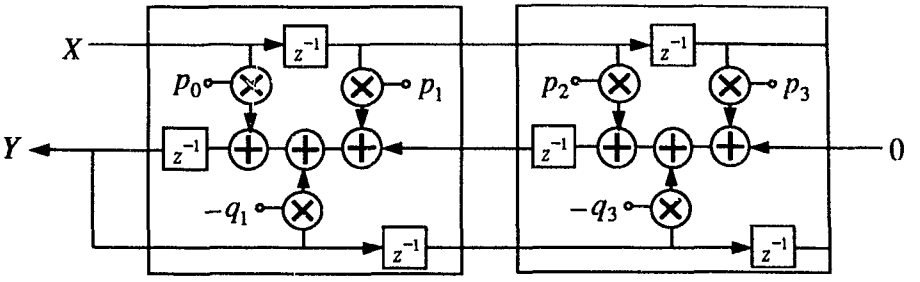


Figure 2.17: Mapping of (2.35) onto a systolic array.

We can write

$$\begin{aligned} Y &= p_0 X z^{-4} + p_1 X z^{-5} + p_2 X z^{-6} + p_3 X z^{-7} \\ &\quad -q_1 Y z^{-1} - q_3 Y z^{-3} \qquad (2.36) \\ &= z^{-1}(p_3 X_3 - q_1 Y_0 + z^{-1}(p_2 X_2 - q_3 Y_1 \\ &\quad +z^{-1}(p_1 X_1 + z^{-1}(p_0 X_0)))) \qquad (2.37) \end{aligned}$$

where

$$\begin{aligned} X_k &= z^{-2} X_{k-1}, \quad k = 1, 2, 3 \\ X_0 &= X \\ Y_1 &= z^{-1} Y_0 \\ Y &= Y_0 \end{aligned}$$

The mapping of (2.37) results in the structure reported in [40]. This structure has only one type of PE and all the signals as well as the partial products are pipelined. Each PE incorporates two multipliers, two adders, and three unit delays.

A variety of systolic structures obtained by manipulating the second-order filter equation are described in [41].

2.6.2 Decimators and Interpolators

In the following we apply the z-domain method to obtain systolic structures for multirate systems. In various applications, it is necessary or desirable to change

the effective sampling rate of a discrete-time signal. The process of increasing the sampling rate by an integer factor is called interpolation and reduction of the sampling rate by an integer factor is called decimation. These two operations may be combined to effect the change in sampling rate by a rational factor [42].

The transfer function of a general decimator is given by

$$H(z^{-1}) = \sum_{i=0}^{N-1} z^{-(i+1)} H_i(z^{-N}) \tag{2.38}$$

The above equation is slightly different from that mentioned in [43] since we have assumed that each PE has a storage element. Since the transfer function $H_i(z^{-N})$ represents an allpass filter, it can be expressed as a cascade of elementary allpass transfer functions $A_{i,k}(z^{-N})$, i.e.,

$$H_i(z^{-N}) = \prod_{k=1}^{K_i} A_{i,k}(z^{-N}) \tag{2.39}$$

$$A_{i,k} = \frac{a_{i,k} + z^{-N}}{1 + a_{i,k} z^{-N}} \tag{2.40}$$

For the case where $N = 3$ and $K_i = 3$, the transfer function for the decimator becomes

$$H(z^{-1}) = z^{-1}(H_0(z^{-3}) + z^{-1}(H_1(z^{-3}) + z^{-1}(H_2(z^{-3})))) \tag{2.41}$$

where

$$H_i(z^{-3}) = \left(\frac{a_{i,1} + z^{-3}}{1 + a_{i,1} z^{-3}}\right)\left(\frac{a_{i,2} + z^{-3}}{1 + a_{i,2} z^{-3}}\right)\left(\frac{a_{i,3} + z^{-3}}{1 + a_{i,3} z^{-3}}\right) \tag{2.42}$$

for $i = 0, 1, 2$.

The above allpass transfer functions can be realized using one of the several structures described earlier. Two such structures for decimators are shown in Figs. 2.18 and 2.19.

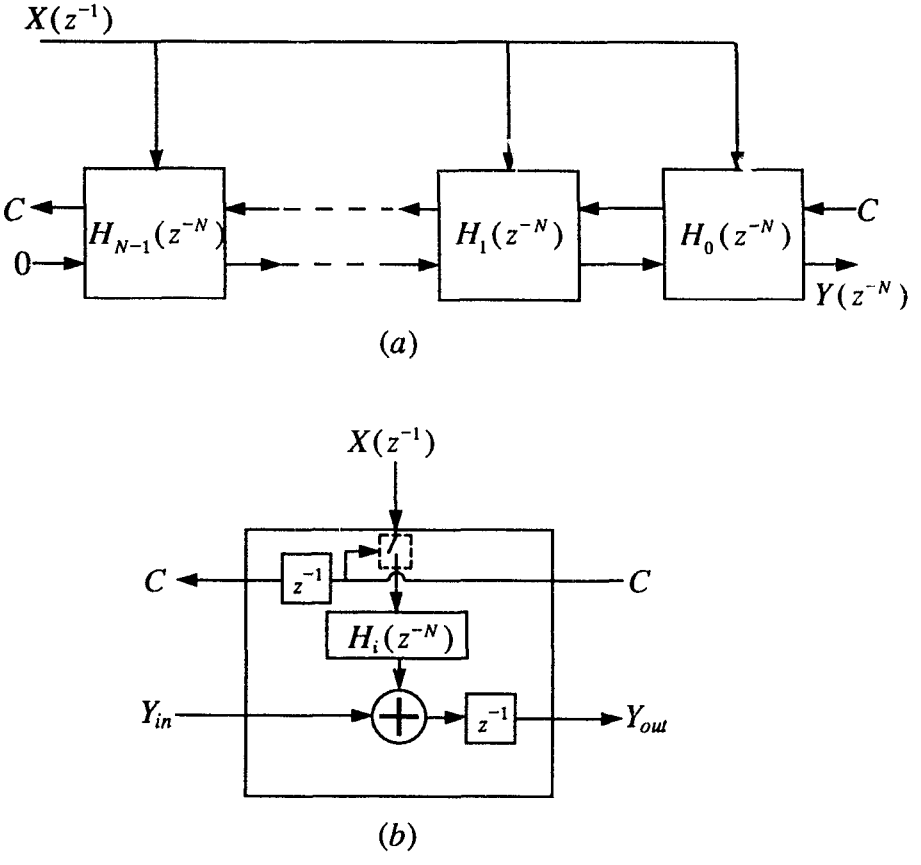
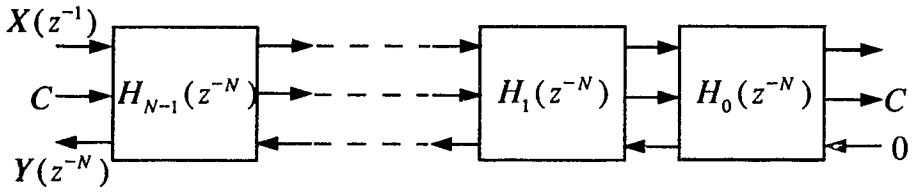
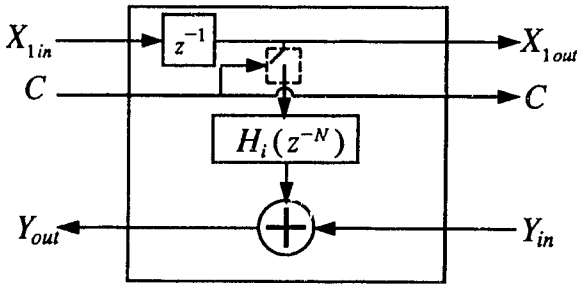


Figure 2.18: Systolic structure for a decimator. (a) The systolic array. (b) Details of PE involved.



(a)



(b)

Figure 2.19: An alternative systolic structure for a decimator. (a) The systolic array. (b) Details of PE involved.

Interpolator structures can be obtained by transposing the decimator structures [37]. Hence the above techniques can also be used to obtain systolic structures for interpolators.

2.7 Conclusions

In this chapter we have discussed two approaches for the derivation of systolic structures for digital filters. The SFG approach has been used to obtain 1-D nonrecursive filters, while the z-domain approach has been used to obtain both nonrecursive and recursive filters. The inconvenience of using the SFG approach for recursive filters has also been discussed. By reordering the filter equations in different ways, several structures can be obtained that are systolic. It must, however, be pointed out that the designer must evaluate the merits and demerits of the structures in terms of design constraints. It must be mentioned here that the structures developed in this chapter are suitable for bit-parallel realizations. However, the z-domain approach could also be used to obtain bit-serial realizations by rewriting the filter equations in terms of the bit representations of the signals involved.

In the next chapter we shall extend the z-domain approach to include the derivation of M-D digital filters. We also compare the various structures obtained on the basis of the speed, area, cycle period, and latency.

Chapter 3

Mapping methodology for multidimensional digital filters

3.1 Introduction

In this chapter, we extend the z-domain approach described in the previous chapter to the mapping of multidimensional (M-D) digital-filter algorithms onto systolic arrays.

It is well known that an M-D convolution can be evaluated as a series of 1-D convolutions. This fact has been used by Kung *et al.* in their design of 2-D convolvers [44]. Other M-D digital-filter realizations have been proposed in [12], and [45]-[47]. While the structures in [12] are systolic, those in [45]-[47] are irregular and are not, as a consequence, immediately suitable for VLSI implementation. In [12], the realizations have been obtained on the basis of a time-domain data-flow analysis and, consequently, only a few structures have been explored. Such a method of obtaining systolic structures does not allow the designer to obtain structures that meet desirable or preset criteria such as system latency, locality, modularity, and PE complexity. Below, we use the z-domain approach to systematically derive systolic structures for 2-D and 3-D digital filters that are modular and hierarchical and explain methods to extend the approach to obtain M-D digital filters.

3.2 Two-dimensional recursive filters

A first-quadrant 2-D recursive filter can be represented by the equation [48]

$$Y = \sum_{k_2=0}^{h-1}\sum_{k_1=0}^{w-1} a(k_1,\ k_2)Xz_1^{-k_1}\ z_2^{-k_2} - \sum_{k_2=0}^{h-1}\sum_{k_1=0}^{w-1} b(k_1,\ k_2)Yz_1^{-k_1}\ z_2^{-k_2} \tag{3.1}$$

where $X \equiv X(z_1,\ z_2)$, $Y \equiv Y(z_1,\ z_2)$, and $b(0,0) = 0$. This equation can also be written as

$$Y = \left[\sum_{k_1=0}^{w-1} a(k_1,\ 0)z_1^{-k_1}X - \sum_{k_1=1}^{w-1} b(k_1,\ 0)z_1^{-k_1}Y\right] + \sum_{k_2=1}^{h-1}\left[\sum_{k_1=0}^{w-1} a(k_1,\ k_2)z_1^{-k_1}z_2^{-k_2}X - \sum_{k_1=0}^{w-1} b(k_1,\ k_2)z_1^{-k_1}z_2^{-k_2}Y\right] \tag{3.2}$$

From (3.1) and (3.2) it can be seen that a 2-D recursive filter can be treated as either a combination of 2-D nonrecursive filters or 1-D recursive filters. In addition, nonrecursive filters can be obtained by setting the b coefficients to zero in the recursive filter equations.

In the following sections we derive different 2-D recursive structures in terms of 1-D recursive structures [49]. The 2-D structures to be obtained realize (3.2) except that the output is delayed by one sampling period. In effect, the structures realize the modified equation

$$Y = \left[\sum_{k_1=0}^{w-1} a(k_1,\ 0)z_1^{-(k_1+1)}X - \sum_{k_1=1}^{w-1} b(k_1,\ 0)z_1^{-k_1}Y\right] + \sum_{k_2=1}^{h-1}\left[\sum_{k_1=0}^{w-1} a(k_1,\ k_2)z_1^{-(k_1+1)}z_2^{-k_2}X - \sum_{k_1=0}^{w-1} b(k_1,\ k_2)z_1^{-(k_1+1)}z_1z_2^{-k_2}Y\right] \tag{3.3}$$

Scheme 1

Equation (3.3) can be written as

$$\begin{aligned} Y &= \left[\sum_{k_1=0}^{w-1}\left(a(k_1,\ 0)z_1^{-(k_1+1)}\right)X - \sum_{k_1=1}^{w-1}\left(b(k_1,\ 0)z_1^{-k_1}\right)Y\right] \\ &\quad + \sum_{k_2=1}^{h-1} z_1 z_2^{-k_2}\left[\left(\sum_{k_1=0}^{w-1} a(k_1,\ k_2)z_1^{-(k_1+1)}\right)(z_1^{-1}X)\right. \\ &\quad \left. - \left(\sum_{k_1=0}^{w-1} b(k_1,\ k_2)z_1^{-(k_1+1)}\right)Y\right] \\ &= [F(0)X - G(0)Y] + \sum_{k_2=1}^{h-1} z_1 z_2^{-k_2}\left[F(k_2)(z_1^{-1}X) - G(k_2)Y\right] \end{aligned} \tag{3.4}$$

where

$$F(k_2) = \sum_{k_1=0}^{w-1} a(k_1,\ k_2)z_1^{-(k_1+1)} \qquad \text{for } k_2 = 0,\ 1,\ \cdots,\ h-1$$

and

$$G(k_2) = \begin{cases} \sum_{k_1=1}^{w-1} b(k_1,\ 0)z_1^{-k_1} & \text{for } k_2 = 0 \\ \sum_{k_1=0}^{w-1} b(k_1,\ k_2)z_1^{-(k_1+1)} & \text{for } k_2 = 1,\ 2,\ \cdots,\ h-1 \end{cases}$$

Rewriting (3.4), we get

$$\begin{aligned} Y &= [F(0)X - G(0)Y] + z_1 z_2^{-1}\left([F(1)X' - G(1)Y]\right. \\ &\quad + z_2^{-1}\left([F(2)X' - G(2)Y]\right. \\ &\quad \left.\left. + \left(\cdots z_2^{-1}\left([F(h-1)X' - G(h-1)Y]\right)\cdots\right)\right)\right) \end{aligned}$$

where $X' \equiv z_1^{-1}X$. The above equation can be written as

$$Y_i = z_2^{-1}\left(F(i)X' - G(i)Y + Y_{i+1}\right), \quad 1 \le i \le h-1 \tag{3.5}$$

$$Y = F(0)X - G(0)Y + z_1 Y_1$$

$$Y_h = 0$$

The mapping of (3.5) onto a systolic architecture for a window of size 3×3 is shown in Fig. 3.1 where each of the parallel structures represents a 1-D filter of the type depicted in Fig. 2.11. Nonrecursive 2-D structures can easily be obtained from recursive structures by simply setting the feedback coefficients to zero, i.e., $b(k_1,\ k_2) = 0$ for $0 \leq k_1 \leq w-1$ and $0 \leq k_2 \leq h-1$.

There are some inherent problems to processing raster-scanned images, viz., line and frame wrap-around problems. Let us assume a picture of width W and height H, and a filter window of width w and height h. The last pixel of a line in a frame of image is followed by the first pixel of the next line. The first $w-1$ pixels of each line will contain contributions from the last $w-1$ pixels of the previous line. This phenomenon is known as line wrap-around. Because of this problem, the structure in [44] will produce $H(w-1)$ erroneous pixels.

The first $h-1$ lines of each frame will contain contributions from the last $h-1$ lines of the previous frame. This phenomenon is known as frame wrap-around. Again, in the design proposed in [44], $W(h-1)$ incorrect pixels are produced at the start of a new frame before the first correct results appear. Thus for a continuous processing of raster-scanned images, the total number of erroneous pixels produced per raster is equal to $H(w-1) + W(h-1) - (w-1)(h-1)$.

Line wrap-around for the structure shown in Fig. 3.1 can be eliminated by clearing all the storage elements within the PEs of each 1-D filter structure. However, this should be done after the reception of the last pixel of a given line and before the reception of the first pixel of the next line by any 1-D filter structure. In Fig. 3.1 the bottom 1-D filter structure has to be cleared prior to the clearing of the 1-D filter structures above it. This is due to the delay of the input to the 1-D filter structures above the bottommost 1-D filter structure. Frame wrap-around can be eliminated by clearing all the storage elements within the PEs of each 1-D filter as well as all the delay elements between adjacent 1-D filters. This should be done after the reception of the last pixel in the last line of a given frame and before the reception of the first pixel of the following frame.

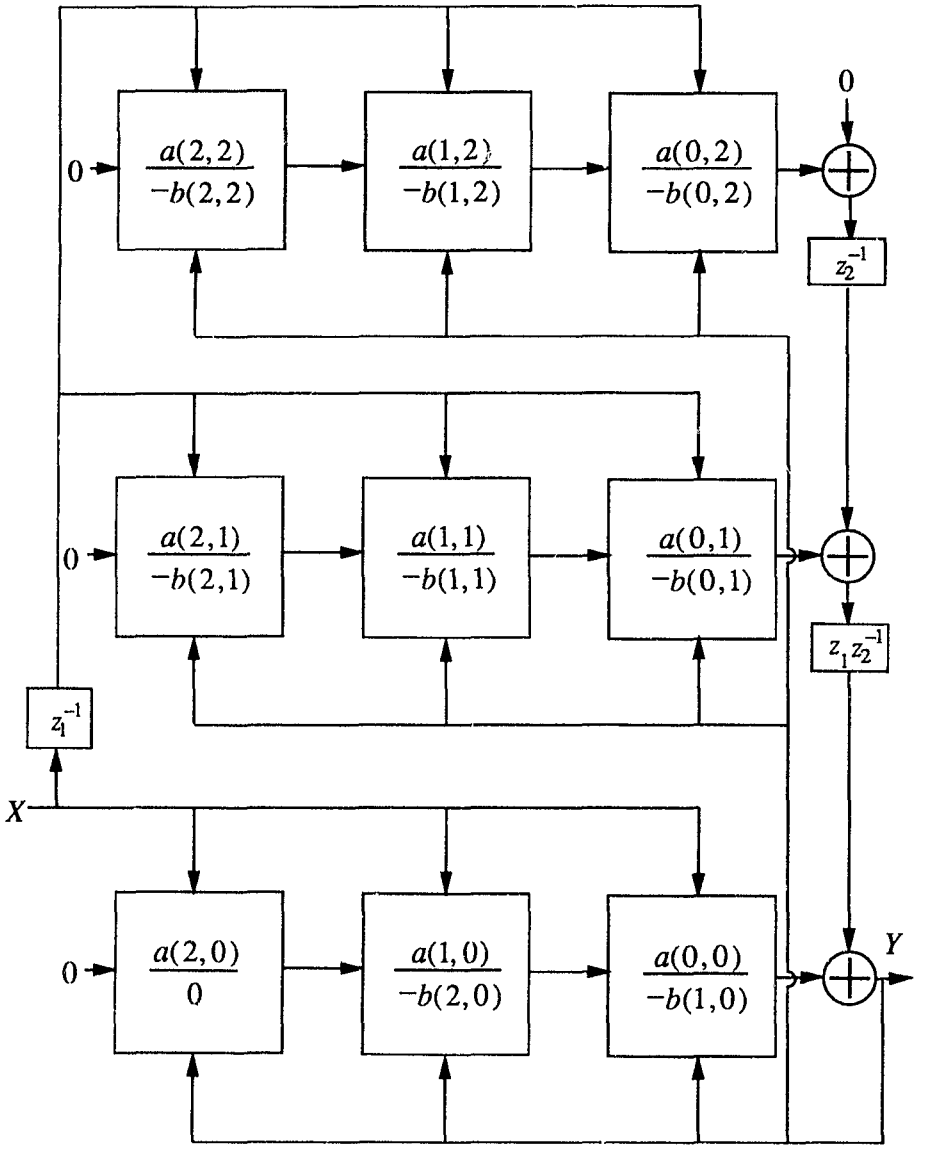


Figure 3.1: Systolic array for a 2-D recursive filter using Scheme 1 for a window of size 3×3.

Scheme 2

Equation (3.3) can be written as

$$\begin{aligned} Y &= \left[\sum_{k_1=0}^{w-1}\left(a(k_1,\ 0)z_1^{-(k_1+1)}\right)X - \sum_{k_1=1}^{w-1}\left(b(k_1,\ 0)z_1^{-k_1}\right)Y\right] \\ &\quad + \sum_{k_2=1}^{h-1}\left[\left(\sum_{k_1=0}^{w-1} a(k_1,\ k_2)z_1^{-(k_1+1)}\right)\left(z_2^{-k_2}X\right)\right. \\ &\quad \left. - \left(\sum_{k_1=0}^{w-1} b(k_1,\ k_2)z_1^{-(k_1+1)}\right)\left(z_1 z_2^{-k_2}Y\right)\right] \\ &= [F(0)X - G(0)Y] \\ &\quad + \sum_{k_2=1}^{h-1}\left[F(k_2)\left(z_2^{-k_2}X\right) - G(k_2)\left(z_1 z_2^{-k_2}Y\right)\right] \end{aligned} \tag{3.6}$$

Expanding the summations in the above equation and manipulating, we get

$$\begin{aligned} Y &= [F(0)X - G(0)Y] + [F(1)X_1 - G(1)Y_1] + [F(2)X_2 - G(2)Y_2] \\ &\quad + \cdots [F(h-1)X_{h-1} - G(h-1)Y_{h-1}] \end{aligned} \tag{3.7}$$

where

$$\begin{aligned} X_k &= z_2^{-1}X_{k-1} \\ Y_k &= z_2^{-1}Y_{k-1} \end{aligned}$$

for $k = 2, 3, \ldots, h-1$ and $X_1 = z_2^{-1}X$, $Y_1 = z_1 z_2^{-1}Y$. The mapping of (3.7) onto a systolic architecture for a window of size 3×3 is shown in Fig. 3.2. [1] Line and frame wrap-around problems can be avoided by using the techniques discussed in Scheme 1.

[1] This systolic architecture has been derived in [12], however, some errors in regard to the sizes of the registers have been corrected in this figure.

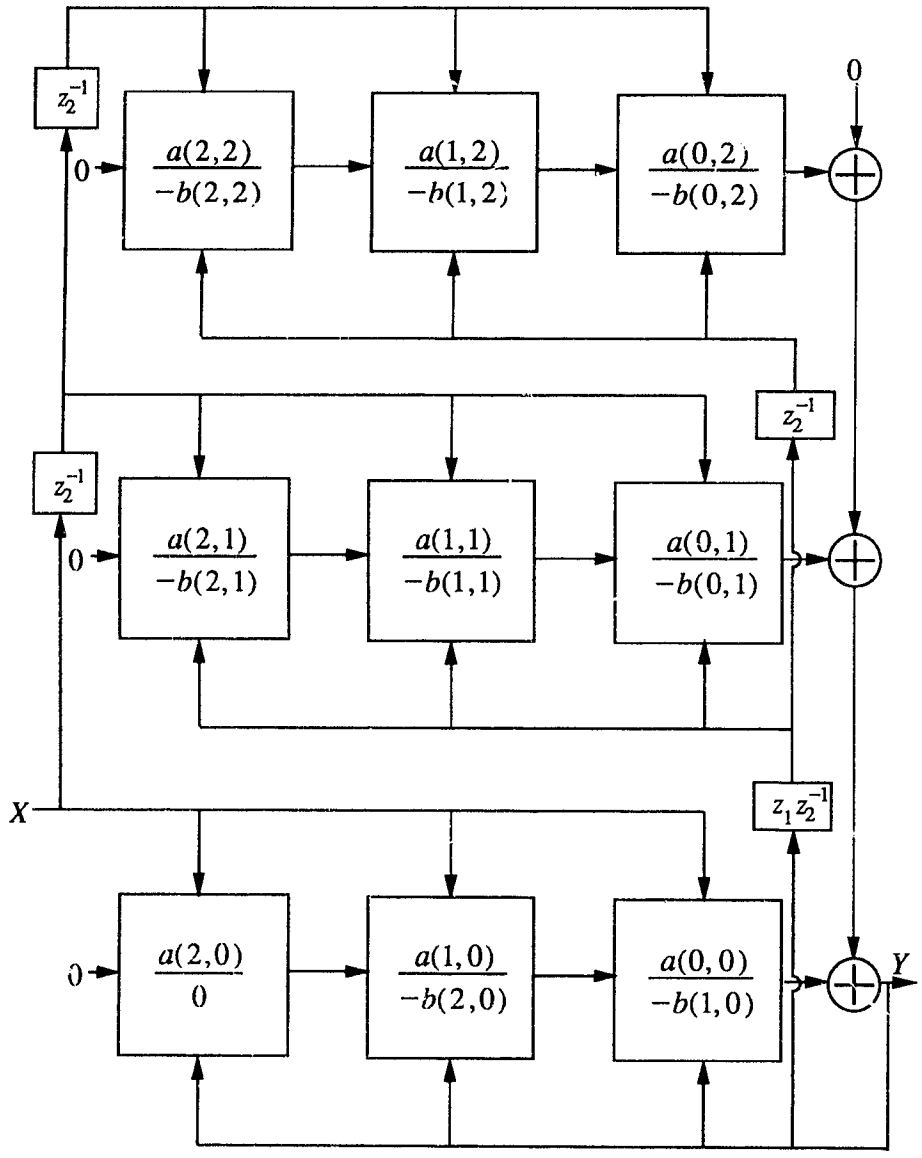


Figure 3.2: Systolic array for a 2-D recursive filter using Scheme 2 for a window of size 3×3.

Scheme 3

Equation (3.3), can be written as

$$\begin{aligned} Y &= \left[\sum_{k_1=0}^{w-1}\left(a(k_1,\ 0)z_1^{-(k_1+1)}\right)X - \sum_{k_1=1}^{w-1}\left(b(k_1,\ 0)z_1^{-k_1}\right)Y\right] \\ &\quad + \sum_{k_2=1}^{h-1} z_1^{(k_2+1)}z_2^{-k_2}\left[\left(\sum_{k_1=0}^{w-1} a(k_1,\ k_2)z_1^{-(k_1+1)}\right)\left(z_1^{-k_2}(z_1^{-1}X)\right)\right. \\ &\quad \left. - \left(\sum_{k_1=0}^{w-1} b(k_1,\ k_2)z_1^{-(k_1+1)}\right)\left(z_1^{-k_2}Y\right)\right] \\ &= [F(0)X - G(0)Y] + \sum_{k_2=1}^{h-1} z_1^{(k_2+1)}z_2^{-k_2}\left[F(k_2)\left(z_1^{-k_2}(z_1^{-1}X)\right)\right. \\ &\quad \left. -G(k_2)\left(z_1^{-k_2}Y\right)\right] \end{aligned} \tag{3.8}$$

Rewriting the above equation, we get

$$\begin{aligned} Y &= [F(0)X - G(0)Y] + z_1^2 z_2^{-1}\left([F(1)X_1 - G(1)Y_1]\right. \\ &\quad + z_1 z_2^{-1}\left([F(2)X_2 - G(2)Y_2]\right. \\ &\quad \left. + (\cdots z_1 z_2^{-1}\left([F(h-1)X_{h-1} \quad -G(h-1)Y_{h-1}]\right\}\cdots))\right) \end{aligned} \tag{3.9}$$

where

$$\begin{aligned} X_k &= z_1^{-1}X_{k-1} \\ Y_k &= z_1^{-1}Y_{k-1} \end{aligned}$$

for $k = 2, 3, \cdots, h-1$ and $X_1 = z_1^{-1}(z_1^{-1}X)$, $Y_1 = z_1^{-1}Y$. Equation (3.9) can be written as

$$\begin{aligned} S_i &= z_1 z_2^{-1}\left(F(i)X_i - G(i)Y_i + S_{i+1}\right) \qquad 1 \le i \le h-1 \\ S_h &= 0 \\ Y &= F(0)X - G(0)Y + z_1 S_1 \end{aligned} \tag{3.10}$$

Figure 3.3 shows a mapping of the (3.9) onto a systolic structure for a window of size 3×3. Line and frame wrap-around problems are circumvented by using the techniques discussed in Scheme 1.

3.3 Three-dimensional recursive filters

In a way similar to that used for 2-D filters, we can derive structures for 3-D recursive filters. A 3-D recursive filter whose output is delayed by one sample in the n_1 direction is characterized by the equation

$$\begin{aligned} Y = & \left[\sum_{k_1=0}^{N_1-1}\left(a(k_1,\ 0,\ 0)z_1^{-(k_1+1)}\right)X - \sum_{k_1=1}^{N_1-1}\left(b(k_1,\ 0,\ 0)z_1^{-k_1}\right)Y\right] \\ & + \sum_{k2=1}^{N_2-1} z_1 z_2^{-k_2}\left[\left(\sum_{k_1=0}^{N_1-1} a(k_1,\ k_2,\ 0)z_1^{-(k_1+1)}\right)(z_1^{-1}X)\right. \\ & \left. - \left(\sum_{k_1=0}^{N_1-1} b(k_1,\ k_2,\ 0)z_1^{-(k_1+1)}\right)Y\right] \\ & + \sum_{k_3=1}^{N_3-1} z_1 z_3^{-k_3}\left[\sum_{k_2=0}^{N_2-1} z_2^{-k_2}\left(\left(\sum_{k_1=0}^{N_1-1} a(k_1,\ k_2,\ k_3)z_1^{-(k_1+1)}\right)(z_1^{-1}X)\right.\right. \\ & \left.\left. - \left(\sum_{k_1=0}^{N_1-1} b(k_1,\ k_2,\ k_3)z_1^{-(k_1+1)}\right)Y\right)\right] \end{aligned} \tag{3.11}$$

Using the method described earlier, several modular structures for 3-D recursive filters can be derived. Fig. 3.4 shows the implementation of a 3-D recursive filter using a scheme similar to that of Scheme 1, for a window of size of $3 \times 3 \times 3$.

It can be seen that the realization schemes are hierarchical. That is, the 3-D filter structures are obtained from the 2-D ones, which are in turn obtained from the 1-D filter structures. The approach can be extended to the derivation of M-D systolic structures.

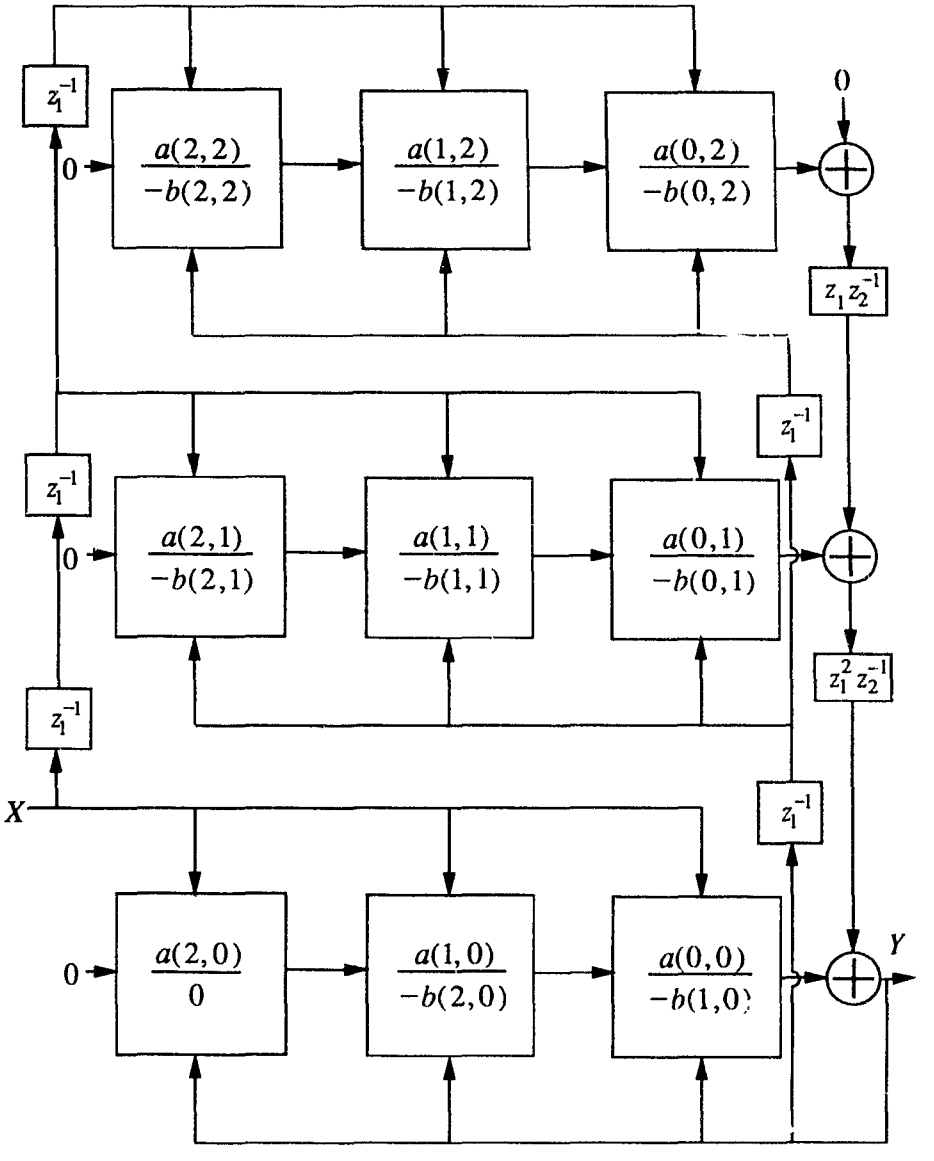
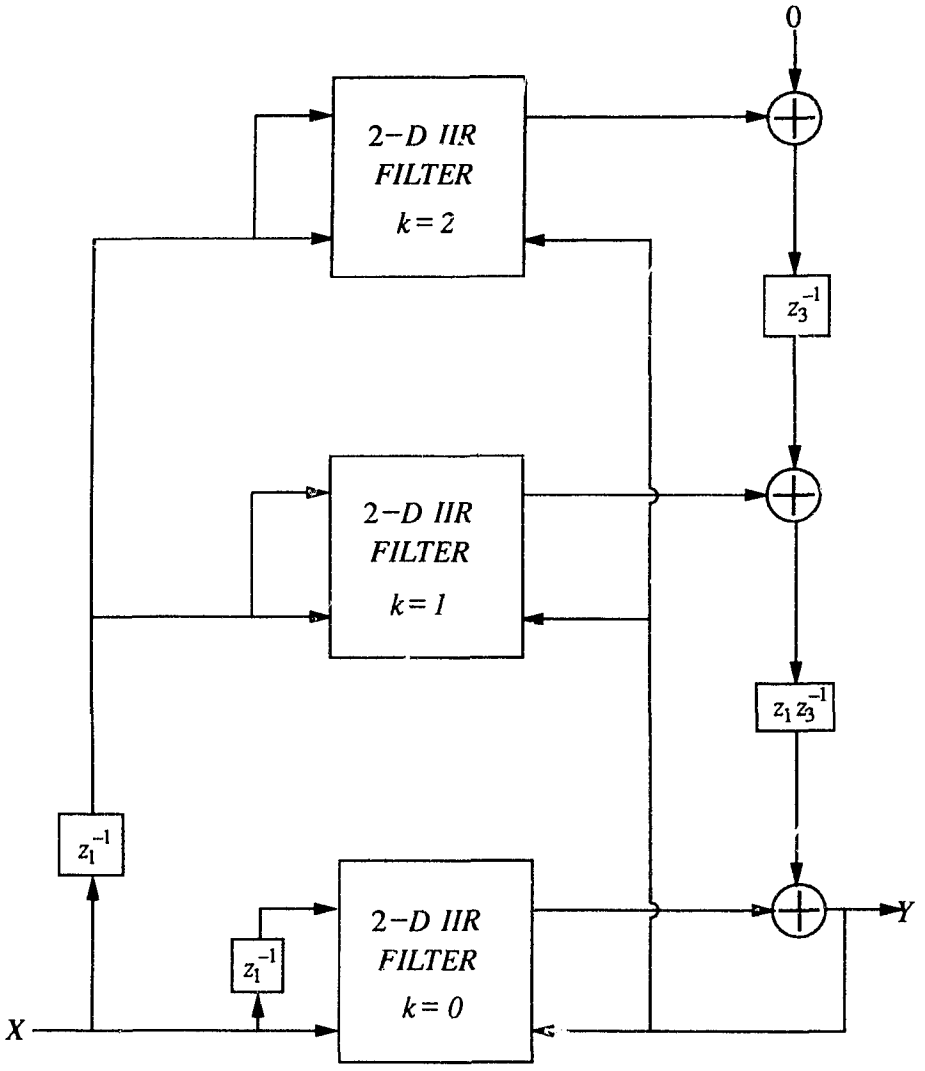
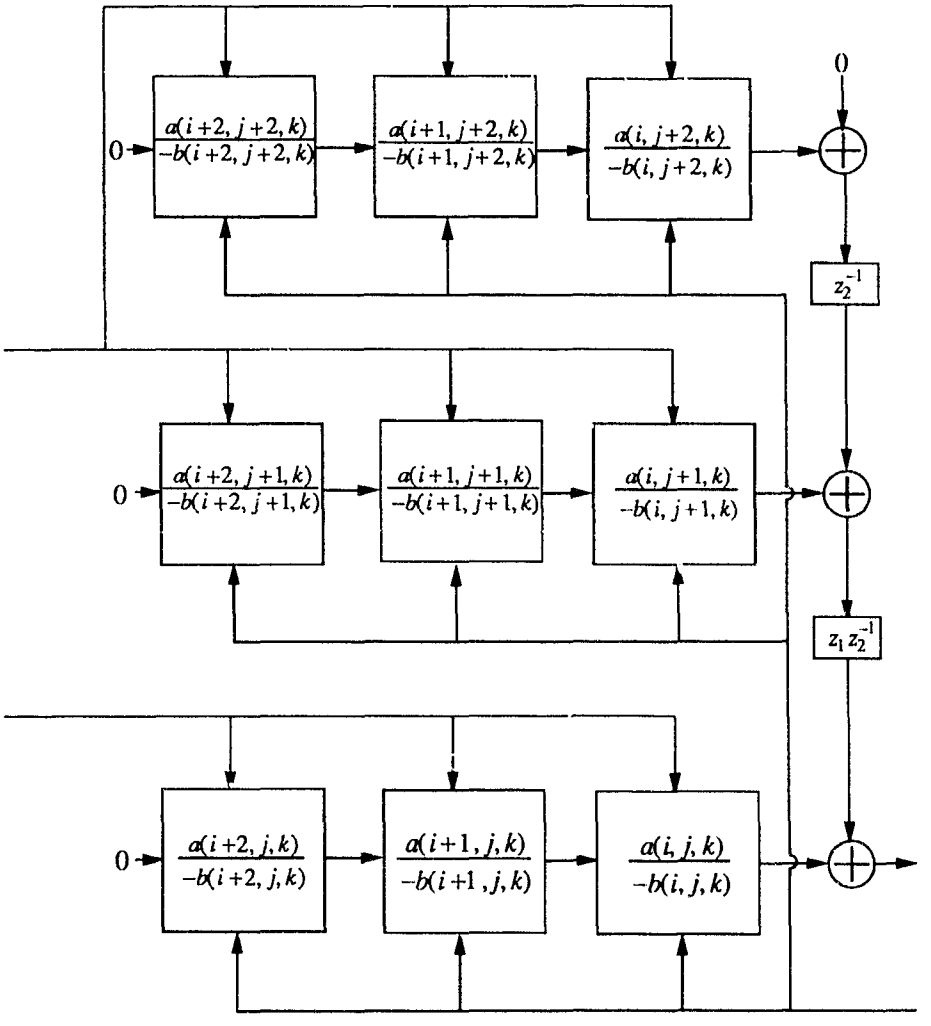


Figure 3.3: Systolic array for a 2-D recursive filter using Scheme 3 for a window of size 3×3.


0
2−D IIR
FILTER
k = 2
z_3^{-1}
2−D IIR
FILTER
k = 1
$z_1 z_3^{-1}$
z_1^{-1}
2−D IIR
FILTER
k = 0
z_1^{-1}
X
Y

(*a*)



(*b*)

Figure 3.4: Systolic array for a 3-D recursive filter using scheme similar to that of Scheme 1 for a window of size $3 \times 3 \times 3$. (a) The systolic array. (b) Details of the PE involved.

3.4 Conclusions

A mapping methodology for the derivation of systolic structures based on the z-domain representation described in Chapter 2 has been extended to the derivation of systolic structures for M-D digital filters. All the systolic structures derived produce a new output for each input sample thereby making the input/output data rates equal to the processing rate. Furthermore, the latency in these structures is one sampling period, which is the lowest that can be achieved in a systolic structure, as was stated above. Error analysis of all the structures described here can be carried out using the technique outlined in [50].

The z-domain approach described leads to hierarchical designs, i.e. the overall structure of a 3-D filter is obtained independently of the underlying structure of the 2-D filter employed, and the 2-D filter is obtained independently of the 1-D filters. Consequently, the 2-D and 3-D structures possess the characteristics of the 1-D structure.

Some associated problems with raster-scanned images, viz., line and frame wrap-around, have been taken into consideration and ways have been suggested for their elimination.

Chapter 4

Design of efficient multipliers

4.1 Introduction

In considering the design of any array processor, it is important to consider the design of the PEs involved. The most important operation in any PE is multiplication. Currently, the multiplier area and time are still the dominant factors in determining the size and speed of operation of the system [51]. In the design of multipliers developed in the past, a lot of effort has been directed towards increasing the speed of operation and decreasing the area by using the advantages of VLSI technology in terms of increased device density and faster switching. However, if multiplication algorithms are designed such that the number of operations required to produce the desired result is reduced then, together with the advantages of VLSI technology a great reduction in area and increase in speed of operation can be achieved simultaneously.

The type of multiplication scheme is dictated by the needs of the application in question. For instance, in digital filters, magnitude truncation is a very important operation that circumvents quantization limit-cycle oscillations. Magnitude truncation is effected by truncating a number of the least significant bits (LSBs) of the multiplication result. Multipliers used in such an application can be designed to incorporate this operation efficiently without generating the bits that are truncated. Convolutions and correlations performed using number theo-

retic transforms, such as the Fermat number-theoretic transform, make use of a diminished-1 multiplier operating under modulo arithmetic. In this chapter several efficient multiplier schemes are described that are used in digital filters, and more generally in signal processing applications. In addition, these schemes are compared with the conventional ones in terms of speed and area.

4.2 Area-efficient parallel multiplier

Two N-bit numbers to be multiplied, A and B, and their product P, can be represented as

$$A = \sum_{i=0}^{N-1} a_i 2^{i-N} \tag{4.1}$$

$$B = \sum_{i=0}^{N-1} b_i 2^{i-N} \tag{4.2}$$

and

$$P = \sum_{i=0}^{2N-1} p_i 2^{i-2N} \tag{4.3}$$

respectively, where $a_i,\ b_i,\ p_i \in \{0, 1\}$.

In a standard $N \times N$ parallel multiplier, the N^2 partial products are generated simultaneously and then added by an array of full adders as shown in Fig. 4.1 for an 8×8 multiplier. The partial sums propagate diagonally in the south-east direction along lines of equal binary weight. The carry signals propagate downwards along increasing binary weight directions. The delay through the parallel multiplier is due to the carry propagation through the adder array and the carry-ripple adder at the bottom of Fig. 4.1(a). In practice the carry-ripple adder is replaced by a carry look-ahead adder.

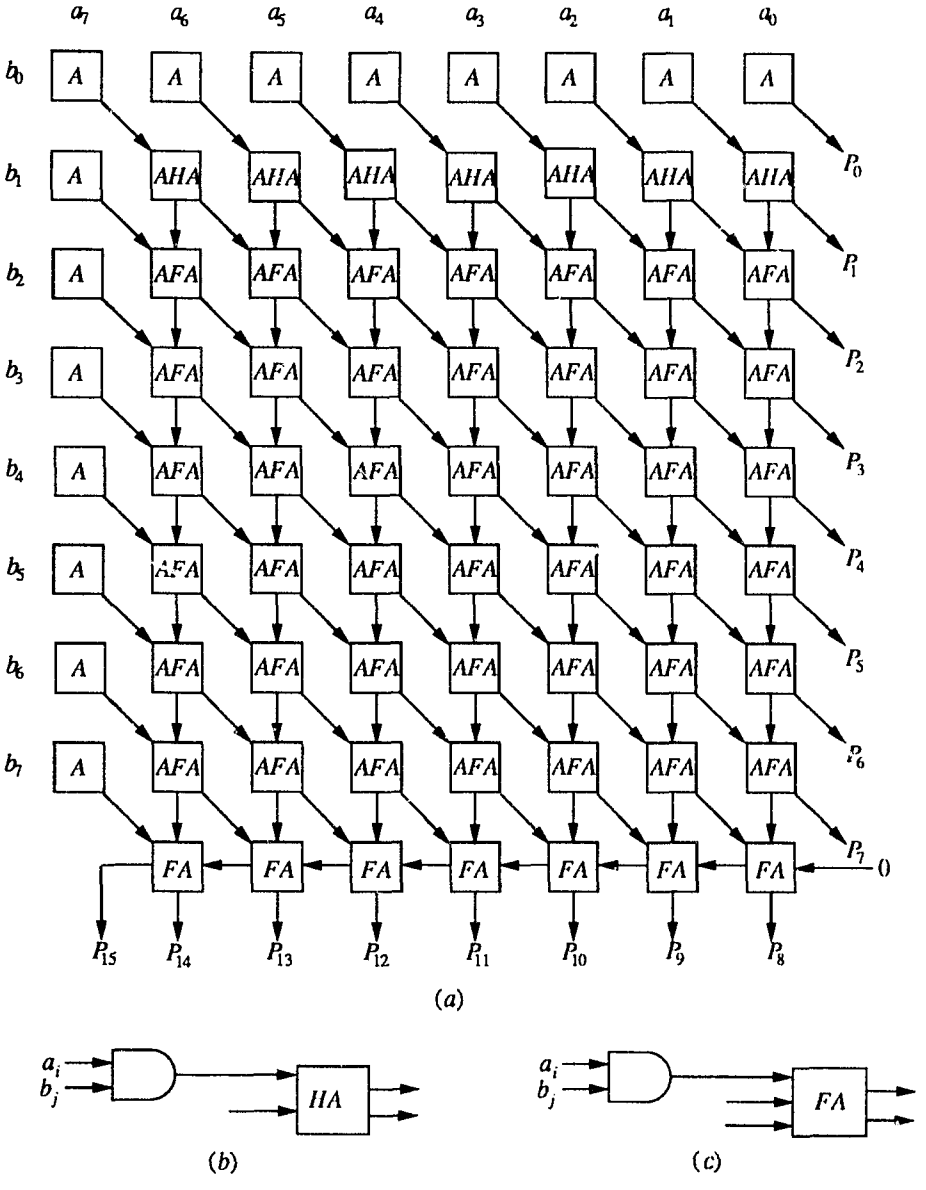


Figure 4.1: An 8×8 multiplication using parallel multiplier where A, HA and FA are the AND, half-adder and full-adder cells, respectively. (a) Multiplier block diagram. (b) Details of AHA cell. (c) Details of AFA cell.

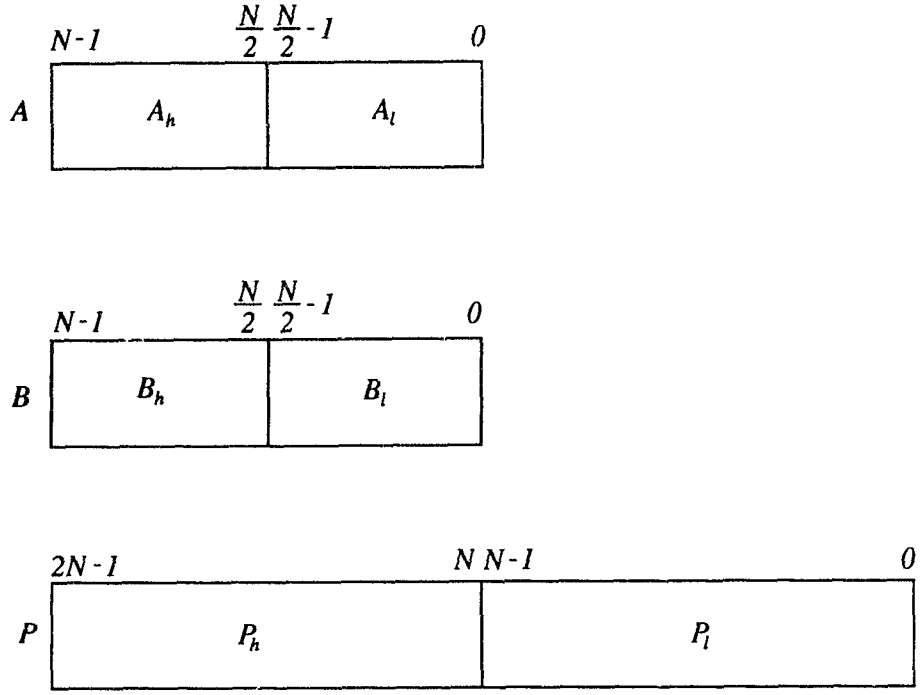


Figure 4.2: Representation of A, B, and P in terms of their most and least significant parts.

The product can be represented by the sum of two parts, P_h and P_l, representing the most and the least significant parts, respectively, i.e.,

$$P = P_h + P_l \tag{4.4}$$

where P_h and P_l are N bits long each and N is assumed to be even. Figure 4.2 shows the representation of the three numbers A, B, and P in terms of their most significant and least significant parts.

The most and least significant parts of P are given by

$$P_h = \sum_{i=N}^{2N-1} p_i \, 2^{i-2N} \tag{4.5}$$

and

$$P_l = \sum_{i=0}^{N-1} p_i \, 2^{i-2N} \tag{4.6}$$

respectively. The product can be written in terms of the most and least significant parts of A and B as

$$P = A_h B_h + A_h B_l + A_l B_h + A_l B_l \tag{4.7}$$

Figure 4.3 shows the manner in which the four terms of (4.7) are generated and Fig. 4.4 shows the various sections of a multiplier generating P_h and P_l. The shaded region contains cells that produce P_l. If the N LSBs of the product are truncated (or rounded), segment P_l is discarded and, in effect, approximately about half of the adder cells and, in turn, about 50% of the chip area are not used effectively.

In the light of the above discussion, we now attempt to develop a truncated multiplier that generates a product whose value is approximately equal to P_h without using the cells in the shaded part of Fig. 4.4. Figure 4.5 shows the details of an 8×8 truncated multiplier.

Comparison of Figs. 4.1 and 4.5 reveals that the truncated multiplier yields a product which is not exactly equal to P_h. To remedy this, we perform a probabilistic analysis of the resulting multiplication error and determining the expected value of the error which can be used as a bias term to improve the accuracy of the multiplier.

4.2.1 Error analysis

From Fig. 4.1 it can be seen that all the partial products are generated by performing an AND operation on the bits of the two operands. Let p_m be the probability that any multiplicand or multiplier bit is 1. The probability $p(i,j)$ that the partial product at column i and row j is 1 is equal to p_m^2. It must be mentioned that i increases from right to left and j increases from top to bottom as can be noted from Fig. 4.1.

Let $p_s(i,j)$ denote the probability that the sum bit of the adder at the ith column and jth row is one and, similarly, let $p_c(i,j)$ denote the probability that

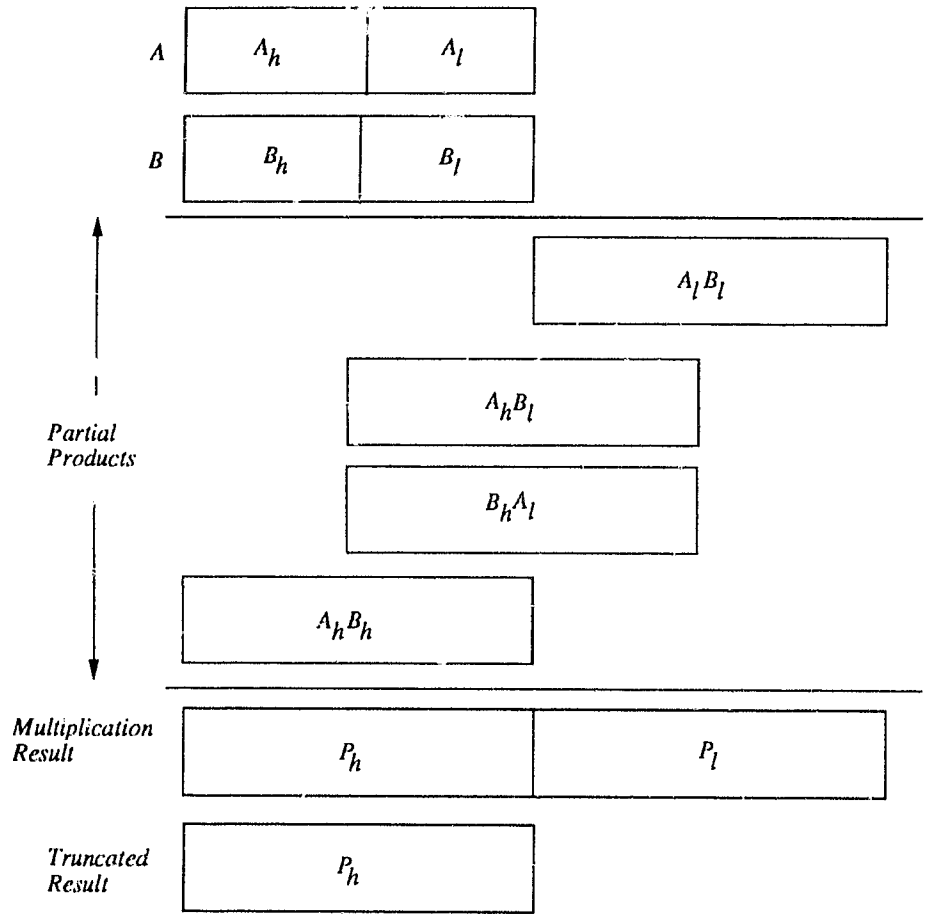


Figure 4.3: Details of generation of P. The partial results are placed horizontally according to their binary weights.

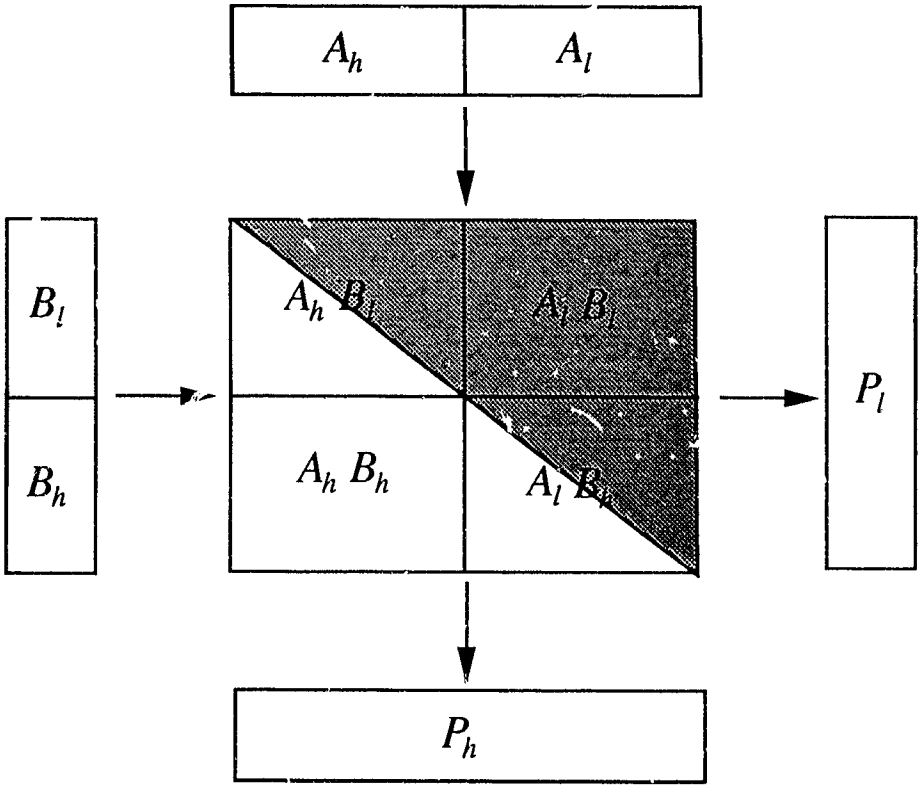


Figure 4.4: Sections in the parallel multiplier generating the four terms of P. The shaded region represents the cells that generate discarded results due to truncation.

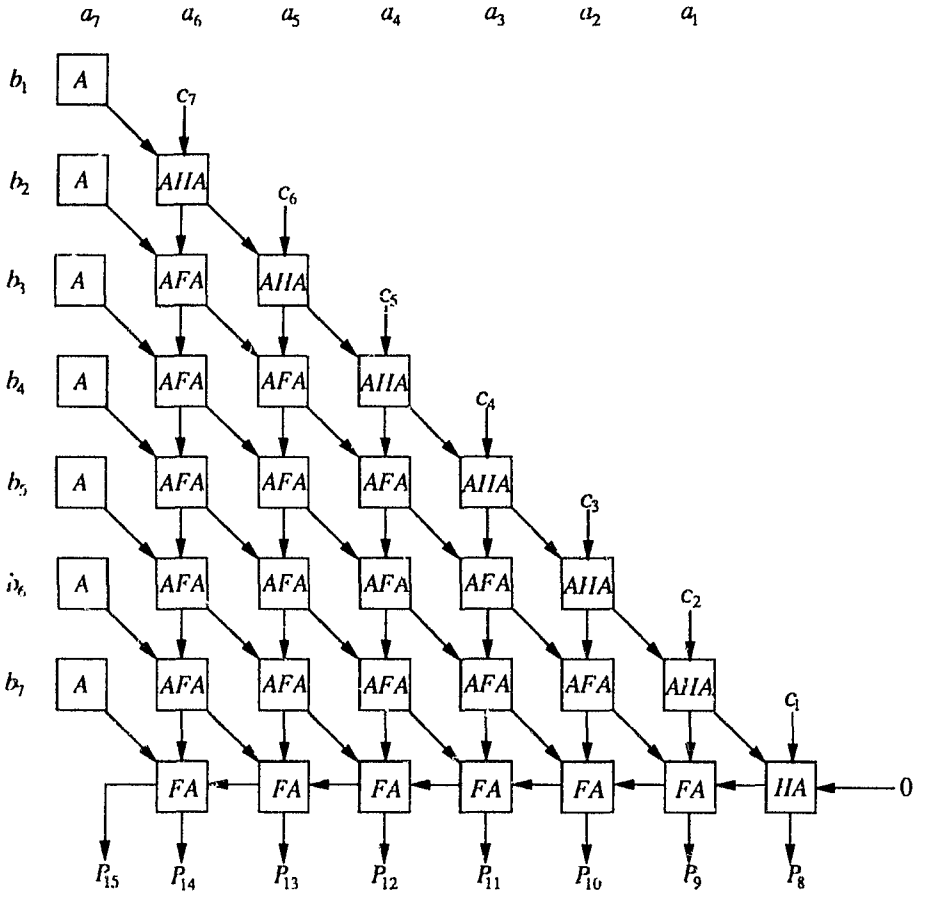


Figure 4.5: A truncated multiplier for an 8×8 bit multiplication.

the carry bit is one. For a parallel multiplier we have

$$\begin{aligned} p_c(i,j) &= p(i,j)\ p(i+1,j-1) \\ p_s(i,j) &= p(i,j)\ q(i+1,j-1) + q(i,j)\ p(i+1,j-1) \end{aligned} \tag{4.8}$$

for $0 \leq i \leq N-2$, $j = 1$, with $q(i,j) = 1 - p(i,j)$, and

$$\begin{aligned} p_c(i,j) &= p(i,j)\ p_s(i+1,j-1)\ p_c(i,j) \\ &\quad + p(i,j)\ p_s(i+1,j-1)\ q_c(i,j) \\ &\quad + p(i,j)\ q_s(i+1,j-1)\ p_c(i,j) \\ &\quad + q(i,j)\ p_s(i+1,j-1)\ p_c(i,j) \\ p_s(i,j) &= p(i,j)\ p_s(i+1,j-1)\ p_c(i,j) \\ &\quad + p(i,j)\ q_s(i+1,j-1)\ q_c(i,j) \\ &\quad + q(i,j)\ q_s(i+1,j-1)\ p_c(i,j) \\ &\quad + q(i,j)\ p_s(i+1,j-1)\ q_c(i,j) \end{aligned} \tag{4.9}$$

for $0 \leq i \leq N-2$, $2 \leq j \leq N-1$, with $q_c(i,j) = 1 - p_c(i,j)$ and $q_s(i,j) = 1 - p_s(i,j)$.

It can be seen from Fig. 4.1 that the expected error in the product of the truncated multiplier is equal to the value of the output carry bits from the cells at position (i,j) where $i+j = N-1$, $0 \leq i \leq N-2$, and $1 \leq j \leq N-1$. The weight of all the bits at the diagonal is equal to 2^{-N}. It is thus evident that the expected error ϵ' lies within the limits $0 \leq \epsilon \leq N-1$, where $\epsilon = \epsilon'/2^{-N}$. If we now calculate the expected value of the error, we can bias the result generated to yield a product with minimum error.

For brevity we shall call the cells at position (i,j) such that $i+j = N-1$ as the diagonal cells. The probability that all the input carry bits at the diagonal cells are zero is

$$p_d(0) = \prod_{i=0}^{N-2} q_c(i, N-1-i) = p_\phi \tag{4.10}$$

The probability that one of the input carry bits at the diagonal cells is one is

$$p_d(1) = \sum_{i=0}^{N-2} \frac{p_c(i, N-1-i)}{q_c(i, N-1-i)} p_\phi \tag{4.11}$$

The probability that two of the input carry bits at the diagonal cells are one is

$$p_d(2) = \sum_{i=0}^{N-3} \sum_{j=i+1}^{N-2} \frac{p_c(i, N-1-i)}{q_c(i, N-1-i)} \quad \frac{p_c(j, N-1-j)}{q_c(j, N-1-j)} p_\phi$$

Similarly, we can derive the probability that all the input carry bits on the diagonal cells are one as

$$p_d(N-1) = \sum_{i=0}^{N-2-(N-2)} \sum_{j=i+1}^{N-2-(N-3)} \cdots \sum_{s=r+1}^{N-2} \frac{p_c(i, N-1-i)}{q_c(N, N-1-i)} \times \frac{p_c(j, N-1-j)}{q_c(j, N-1-j)} \cdots \frac{p_c(s, N-1-s)}{q_c(s, N-1-s)} p_\phi \tag{4.12}$$

and after some manipulation

$$p_d(N-1) = \prod_{i=0}^{N-2} p_c(i, N-1-i) \tag{4.13}$$

The expected value and the variance of the error can now be written as

$$\mu = E\{\epsilon\} = \sum_{i=0}^{N-1} i \; p_d(i) \tag{4.14}$$

and

$$\sigma^2 = E\{(\epsilon - E\{\epsilon\})^2\} = \sum_{i=0}^{N-1} (i-\mu)^2 \; p_d(i) \tag{4.15}$$

respectively. Figures 4.6 and 4.7 show the variation of the expected error and the standard deviation with the size of the multiplier. On the basis of (4.14), an

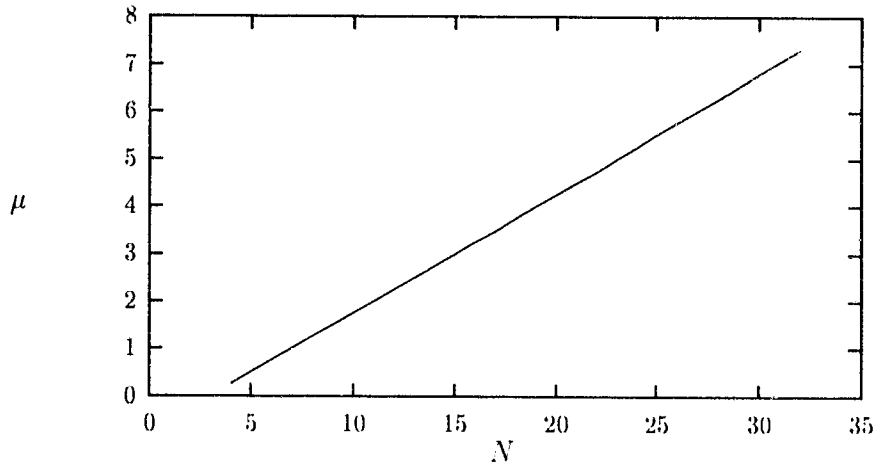


Figure 4.6: Variation of the expected value of the error with N.

error-correction unit can be designed that can generate an appropriate bias which can be added to the output. Instead of employing a separate correction unit, the correction bits $c_1 - c_7$ in Fig. 4.5 can be added to the adder cells located at the diagonal of the multiplier. The rationale for this technique is that the value of the correction bits cannot exceed $(N-1)2^{-N}$ and thus a maximum of $(N-1)$ bits can be added to the input of the multiplier.

4.2.2 Increase of speed in the truncated multiplier

The speed of the truncated multiplier can be increased by using the concept of non-additive multiplicative modules (NMM) [52]. The truncated multiplier can be partitioned as shown in Fig. 4.8. It can be seen from the figure that A_hB_h, A_hB_l and A_lB_h are all generated simultaneously. For an $N \times N$ truncated multiplier, only $N/2$ adder stage delays are incurred before the final addition as opposed to N adder stage delays when the multiplier is implemented without using the NMM technique.

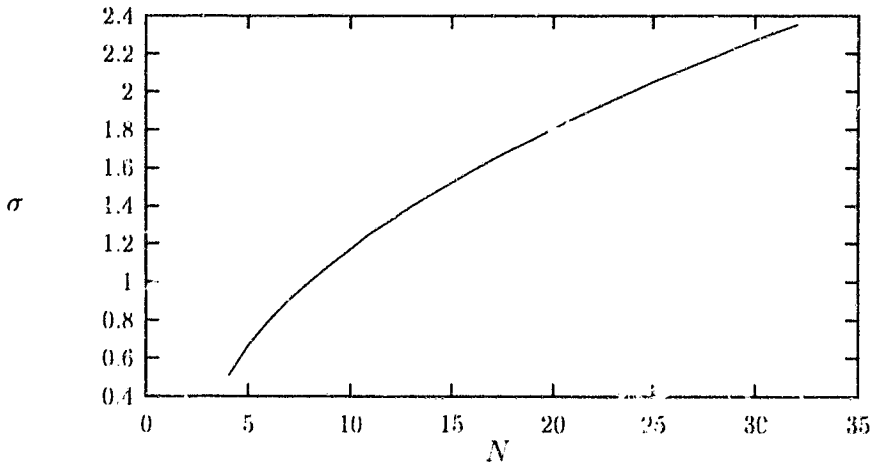


Figure 4.7: Variation of the standard deviation of the error with N.

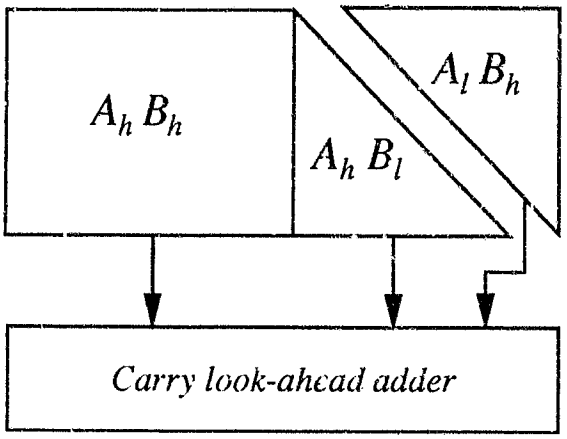


Figure 4.8: NMM technique to increase the speed of the truncated multiplier.

4.2.3 Reduction of area in the truncated multiplier

Let A_a, A_h, and A_f be the areas of an AND gate, a half adder, and a full adder, respectively. The area of a full parallel multiplier is given by

$$A_{fm} = N^2 A_a + (N-1)A_h + (N-1)^2 A_f \tag{4.16}$$

On the other hand, the maximum area of a truncated multiplier is given by

$$A_{tm} = \frac{1}{2}N(N-1)(A_a + A_f) \tag{4.17}$$

If we assume that the area of a half adder and a full adder are, respectively, four and eleven times that of an AND gate, the area of a truncated multiplier (for $N > 4$) is approximately 52% of that of the full multiplier (we make an assumption that the area of the multiplier is equal to the sum of the areas of individual components). Figure 4.9 shows the variation of the ratio of the area of a truncated multiplier to that of a full multiplier with the size of the multiplier. In Chapter 5, we shall design these multipliers to corroborate the theoretical results.

If we let $A_h = \xi A_f$ where $0 < \xi < 0.5$, then the ratio of the areas of the truncated multiplier and the full multiplier can be written as

$$\frac{A_{tm}}{A_{fm}} = \frac{0.5N(N-1)(1 + A_f/A_a)}{N^2 + [N^2 + (\xi - 2)N + \xi + 1]A_f/A_a} \tag{4.18}$$

Thus a family of curves of A_{tm}/A_{fm} versus the size of the multiplier for different values of A_f/A_a can be obtained.

4.2.4 Two's complement multiplication

Two's complement numbers can be handled in a way similar to the sign and magnitude multipliers by using the scheme mentioned in [53]. The multiplication

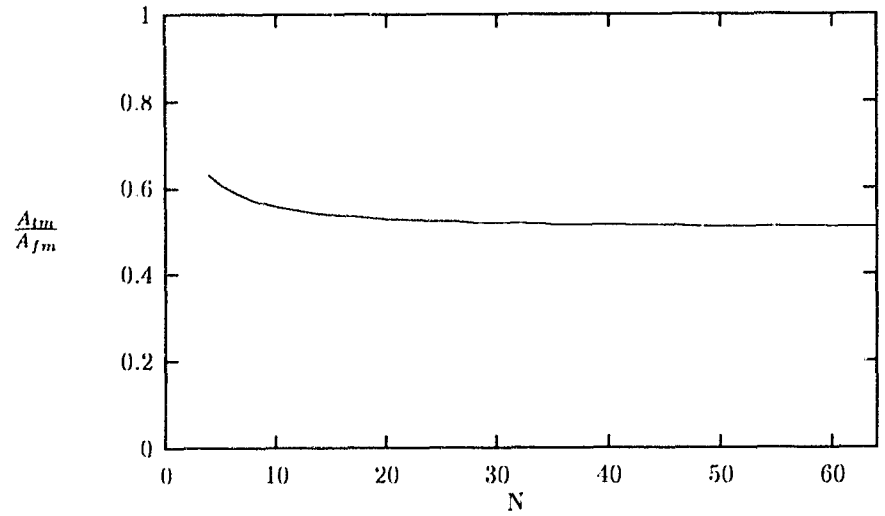


Figure 4.9: Variation of the ratio of the area of a truncated multiplier to that of a full multiplier with N.

of two numbers in two's complement representation is written as

$$P = \left(-a_{N-1} + \sum_{i=0}^{N-2} 2^{i-N+1} a_i\right)\left(-b_{N-1} + \sum_{j=0}^{N-2} 2^{j-N+1} b_j\right) \tag{4.19}$$

$$= \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} 2^{i+j-2N+2} a_i b_j + a_{N-1}b_{N-1}$$

$$-a_{N-1}\sum_{j=0}^{N-2} 2^{j-N+1} b_j - b_{N-1}\sum_{i=0}^{N-2} 2^{i-N+1} a_i \tag{4.20}$$

$$= \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} 2^{i+j-2N+2} a_i b_j + a_{N-1}b_{N-1}$$

$$+\left(\sum_{j=0}^{N-2} 2^{j}\overline{a_{N-1}b_j} + \sum_{i=0}^{N-2} 2^{i}\overline{b_{N-1}a_i}\right) 2^{-(N-1)}$$

$$+2^{-(N-2)} - 2 \tag{4.21}$$

From the above equation it can be seen that the multiplication of two numbers can be written in a form involving only positive partial products. Figure 4.10 shows

the parallel multiplier array for the two's complement multiplication. Evidently, a truncated two's complement multiplier can be readily designed by a structure similar to that in Fig. 4.5. An error analysis can be carried out using the approach in Section 4.2.1.

4.2.5 Pipelined truncated multiplier

In applications where a multiplier is operating on a stream of input data, pipelining can be used to increase the throughput.

It is readily observed from Fig. 4.1 that the data flow is vertical from one row to the next, and there are no horizontal connections between the cells except in the last row where the carry signals propagate through all the cells. Ignoring the last row for the moment, an attractive pipelining strategy is to place a register at the output of each cell in the multiplier. The stage delay would be equal to the delay of a 1-bit full adder plus the delay in the register. This, however, is not the case in the last row of the array, since the data flow is horizontal. One solution is to use a carry look-ahead adder to replace the last row of cells. However, this type of adder does not offer a structural regularity which is compatible with the rest of the array [18]. Besides, as the size of the multiplier increases, the delay through the carry look-ahead adder increases, making it the slowest stage of the pipelined architecture.

The pipeline stages after the last row of adders is shown in Fig. 4.11. In order to understand the pipelining in the truncated multiplier we have shown in Fig. 4.12 the partial products generated in a 4×4 multiplier. Only the partial products to the left of the vertical line are retained. A pipelined version of the truncated multiplier using the above pipelining strategy is shown in Fig. 4.13 for an 8×8 multiplication.

There are three major factors contributing to the increase in the size of the pipelined multiplier over the nonpipelined one, viz., 1) the area used by the pipeline registers, 2) the area used by the clock buffers, and 3) the overhead

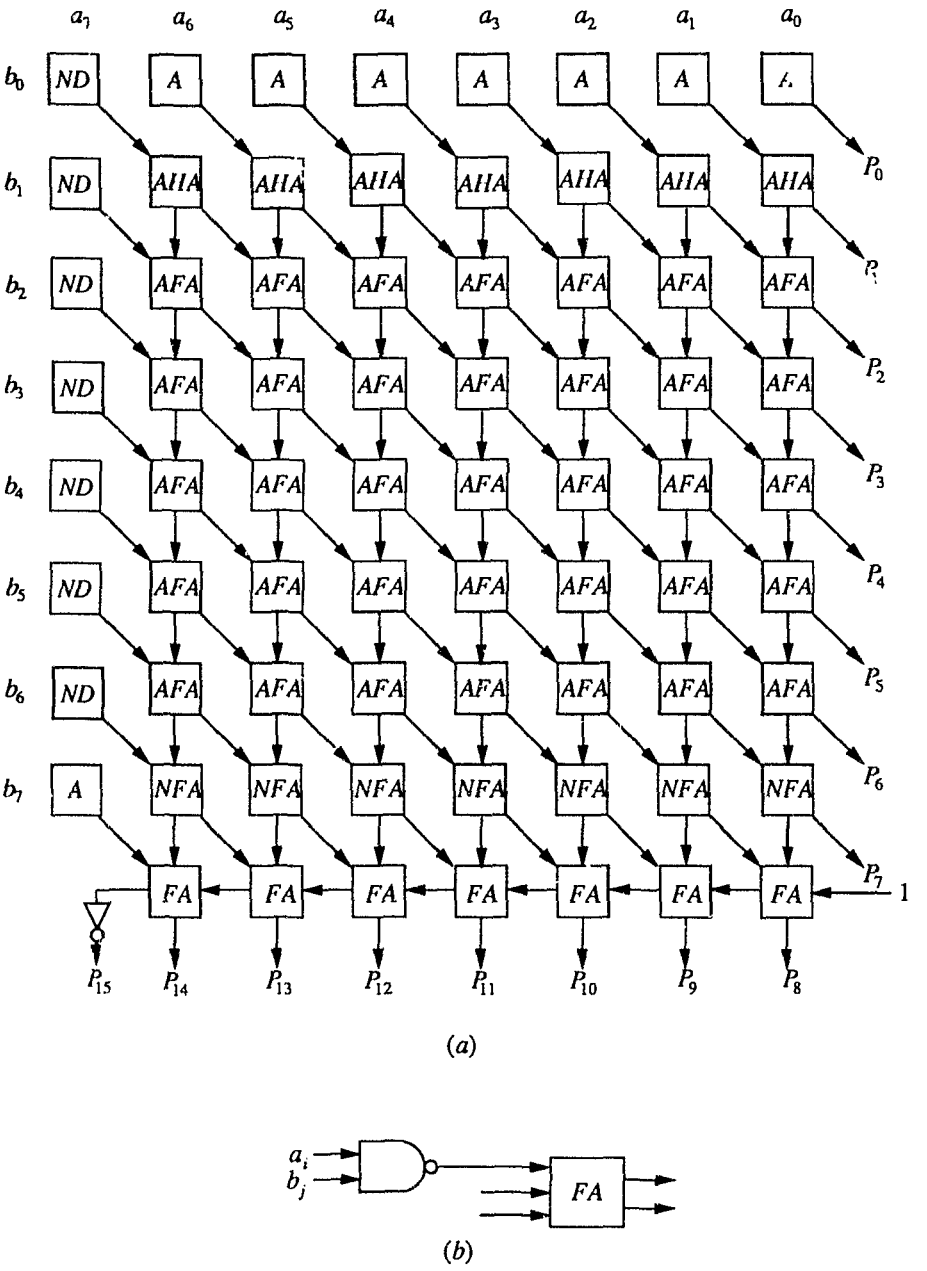


Figure 4.10: An 8×8 two's complement multiplication using parallel multiplier where ND is a NAND gate cell. (a) Multiplier block diagram. (b) Details of NFA cell.

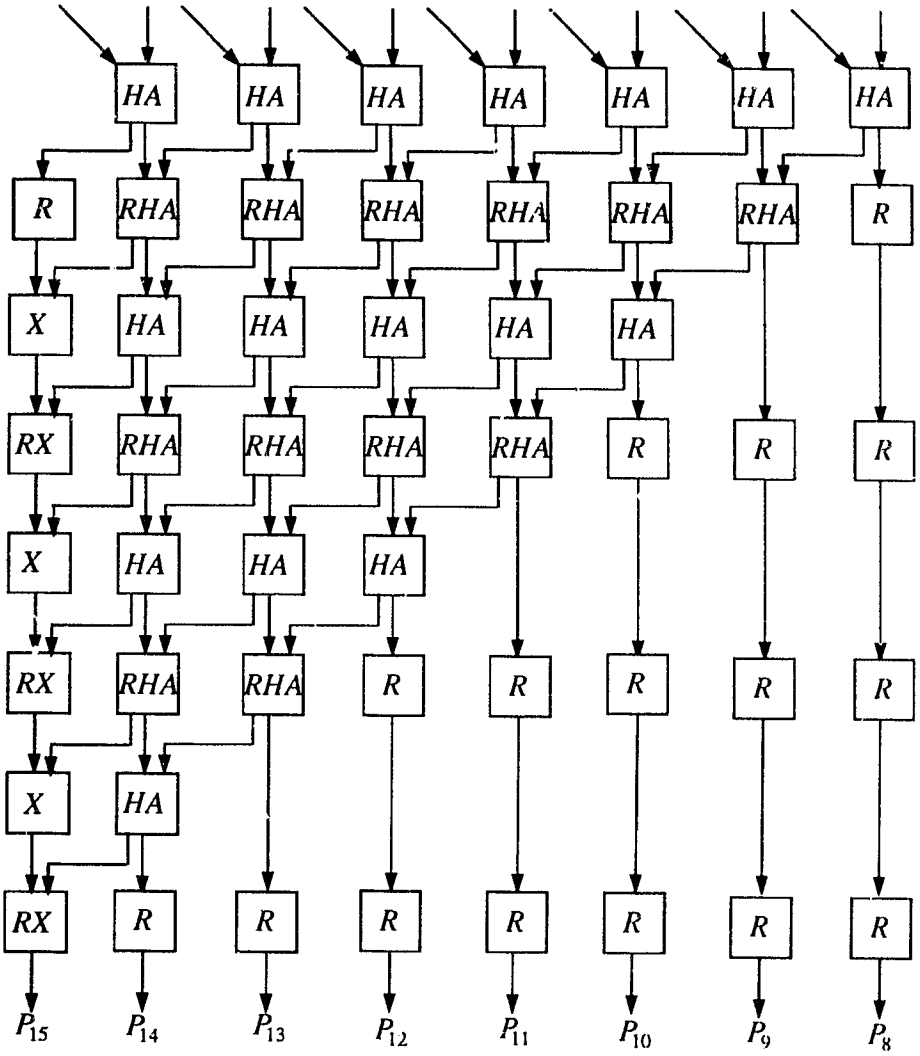


Figure 4.11: Pipelining technique for the last row of full-adder cells. R: 1-bit register, RHA: half adder followed by a 1-bit register, X: Exclusive-OR gate, RX: Exclusive-OR gate followed by a 1-bit register.

a_3	a_2	a_1	a_0				
b_3	b_2	b_1	b_0				
				a_3b_0	a_2b_0	a_1b_0	a_0b_0
			a_3b_1	a_2b_1	a_1b_1	a_0b_1	
		a_3b_2	a_2b_2	a_1b_2	a_0b_2		
	a_3b_3	a_2b_3	a_1b_3	a_0b_3			
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Figure 4.12: Partial products for a 4×4 multiplication.

area for clock and control routing and distribution. All these factors are directly affected by the degree of pipelining.

4.2.6 Application of the truncated multiplier in digital filters

In this section, we design a digital filter first using the truncated two's complement multiplier and then using the full parallel two's complement multiplier assuming that output noise power-spectral density (PSD) is the same in the two cases. We then compare the chip areas required. To this end, we design a second-order Butterworth wave digital filter based on the GIC configuration, shown in Fig. 4.11 [37].

The multiplier values m_1 and m_2 are found to be -0.4142136 and -0.4142136, respectively. The output PSD of the filter is given by

$$S_o(e^{j\omega}) = \left(|H_L(e^{j\omega})|^2 + 2|H_E(e^{j\omega})|^2\right) S_e(e^{j\omega}) \tag{4.22}$$

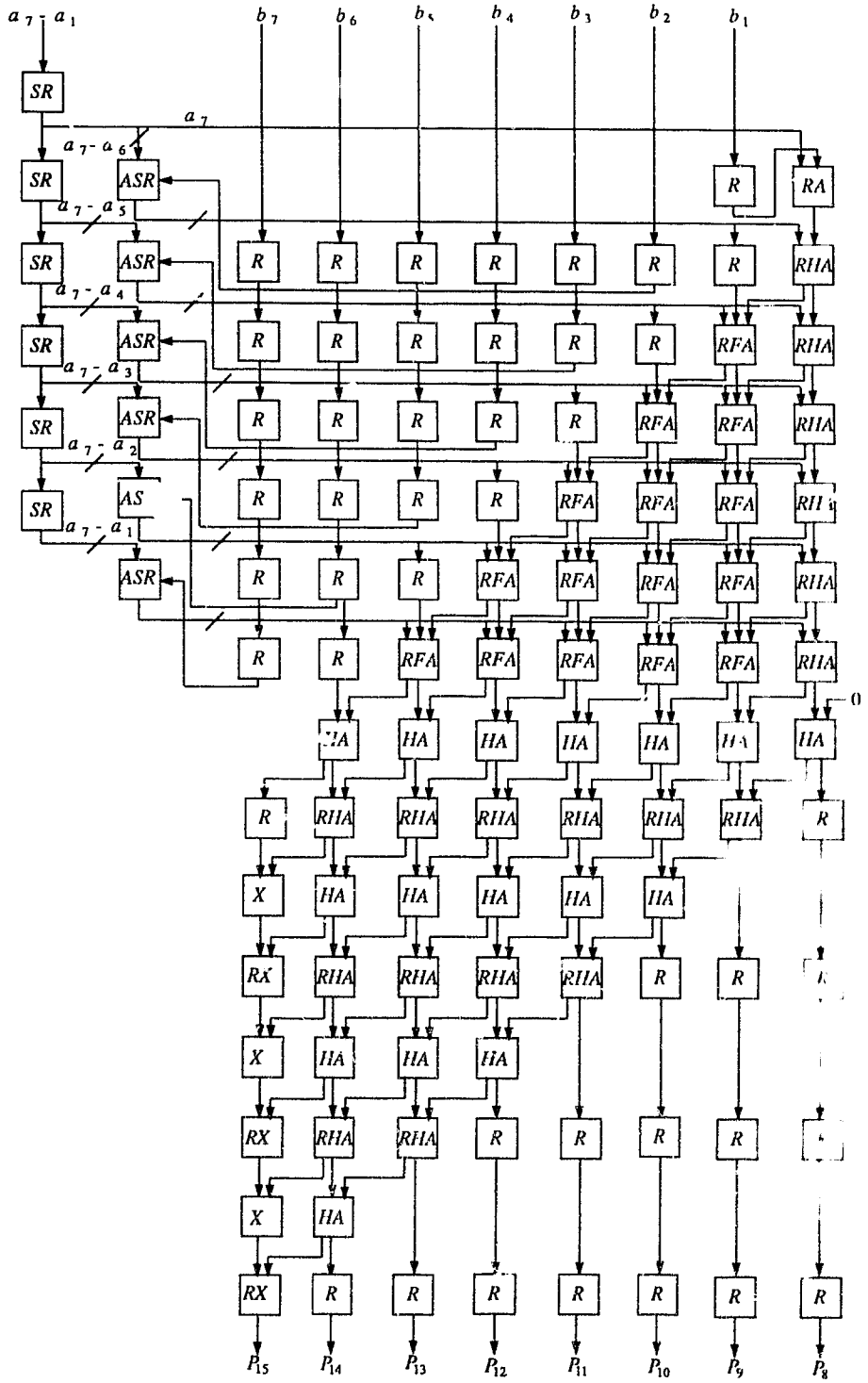


Figure 4.13: An 8×8 fully pipelined truncated multiplier. SR: 7-bit shift register, ASR: A set of AND gates followed by a register.

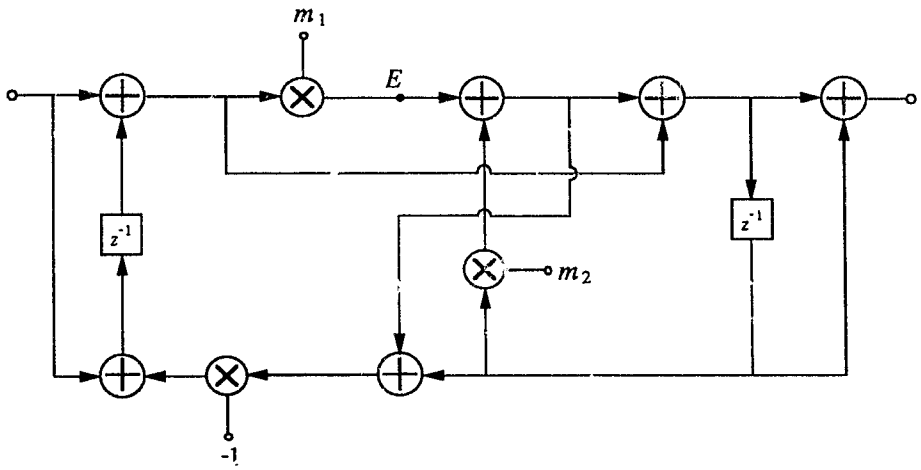


Figure 4.14: A lowpass GIC wave digital filter.

where

$$H_D(e^{j\omega}) = \frac{(1+m_1)(z+1)^2}{z^2+(m_1-m_2)z+1+m_1+m_2}$$

$$H_E(e^{j\omega}) = \frac{(z+1)(z-1)}{z^2+(m_1-m_2)z+1+m_1+m_2}$$

are the transfer functions from the filter input and point E to the output, respectively, and $S_e(e^{j\omega})$ is the PSD of the error of the multiplier used.

For a two's complement truncated multiplier, the average value of the error in a 12×12 multiplier is confined to the two LSBs. Thus a 12×12 truncated multiplier should produce approximately the same roundoff noise as a 10×10 standard multiplier. This, indeed, is the case as can be seen in Fig. 4.15.

On using (4.16) and (4.17) (incorporating the areas of NAND gates instead of AND gates for two's complement multiplication) we obtain the area of the truncated multiplier to be about 81 percent of that of a standard multiplier. This represents a substantial saving since in typical applications the digital-filter structure comprises many multipliers.

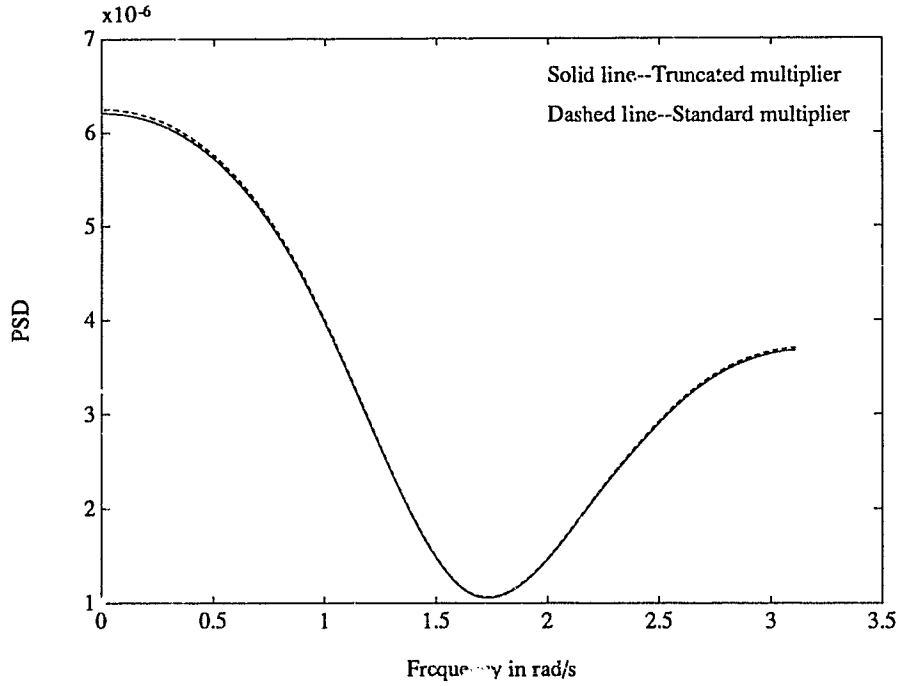


Figure 4.15: Variation of the output noise PSD of a lowpass GIC wave digital filter using standard and truncated multipliers.

4.2.7 Quasi-serial truncated multiplier

The quasi-serial multiplier [54] generates the bits of the product sequentially from the LSB to the MSB. Each bit is computed by counting the number of ones in the corresponding column of the summand matrix and adding the previous carry bits. This idea can be extended to obtain quasi-serial truncated multiplier. Let the multiplicand A, and the multiplier B, be expressed as in (4.1) and (4.2). The product of the truncated multiplier can be written as

$$P = \sum_{i=N}^{2N-2} s_i 2^{i-2N} \tag{4.23}$$

where

$$s_i = \sum_{k=0}^{i} a_k b_{i-k} \tag{4.24}$$

with $a_i,\ b_i = 0$, for $i < 1$ or $i > N-1$. The product of the truncated multiplier can be obtained from a set of recursions given by

$$\begin{aligned} p_i &= (s_i + c_{i-1}) \bmod 2 \\ c_i &= (s_i + c_{i-1} - p_i)/2 \\ c_{N-1} &= 0 \end{aligned} \tag{4.25}$$

for $i = N, \cdots, 2N-1$.

The above recursions sum the s_i term defined by (4.24), and reduce the sum for each column of the summand matrix to a product bit p_i and a carry value c_i.

The algorithm defined by (4.24) and (4.25) can be directly implemented with two $(N-1)$-bit shift registers, an $(N-1)$-bit register, a $\log_2 N$ add-shift register, an $(N-1)$-bit input counter and $(N-1)$ two-input AND gates, as shown in Fig. 4.16.

To initiate the multiplication, the multiplicand is entered into the $(N-1)$-bit shift register and the multiplier is stored in the $(N-1)$-bit register and after the

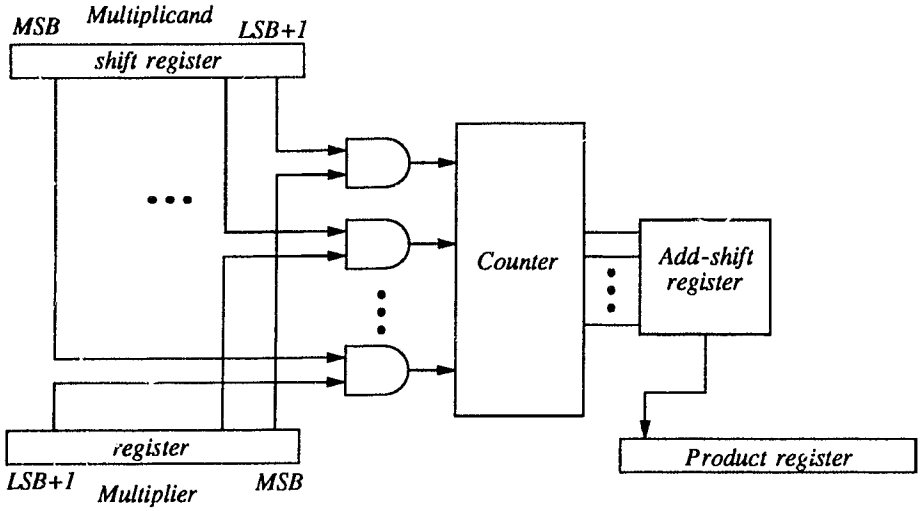


Figure 4.16: Quasi-serial truncated multiplier block diagram.

first cycle a zero is shifted into the MSB of the shift register. During every cycle, starting from the first, the number of ones output by the AND gates are counted using the counter, which are then added to the contents of the add-shift register. The contents of the add-shift register are shifted out as the bits of the product into the product register. After a total of N cycles the complete operation of multiplication is performed.

In order to carry out a two's complement truncated multiplication using the quasi-serial multiplier the same structure mentioned above can be used, except that after the first cycle the MSB is retained in the most significant position of the shift register [55], the topmost AND gate in Fig. 4.16 has to be replaced by a NAND gate and a one has to be stored in the add-shift register before commencing the multiplication.

4.3 Multiplier based on the modified octal Booth algorithm

In the conventional modified Booth algorithm three-bit strings of the multiplier are scanned and appropriate operations are carried out on the multiplicand [56]. This scheme generates exactly $N/2$ rows of partial products where N is the number of bits in the multiplier. Conventional multipliers that use the modified Booth algorithm, which we shall henceforth call the quaternary modified Booth (QMB) algorithm, add the partial products in a stage-by-stage fashion. In the method advanced in [19], the partial products generated by the multiplexers are added concurrently thereby increasing the speed of the multiplier.

In another version of the modified Booth algorithm four-bit strings instead of three-bit strings of the multiplier are scanned thus producing only $N/3$ rows of partial products. In this way, the number of adder stages is reduced and the speed of the multiplier is increased compared to that of the multiplier using the QMB algorithm. In using this version of the modified Booth algorithm, which we shall henceforth call the octal modified Booth (OMB) algorithm , there is a nontrivial operation, namely, a multiplication of the multiplicand by three [51]. This can be performed by first shifting the multiplicand left by one bit and then adding the result to the multiplicand. We will call multipliers based on the QMB and OMB algorithms as QMB and OMB multipliers, respectively, for brevity.

4.3.1 The QMB multiplier

The multiplier can be written as

$$B = \sum_{i=even}^{N} (-2b_{i+1} + b_i + b_{i-1})2^i = \sum_{i=0}^{N/2} Q_i 4^i \tag{4.26}$$

where

$$Q_i = -2b_{2i+1} + b_{2i} + b_{2i-1}$$

with $b_{-1} = 0$ and $Q_i \in \{-2, -1, 0, +1, +2\}$. The product of the multiplication can be written as

$$P = \sum_{i=0}^{N/2} AQ_i 4^i \tag{4.27}$$

where A is the multiplicand.

An encoder accepts three-bit strings of the multiplier as input and outputs the appropriate control signal. The control signals generated by the encoder are Z, ADD, $2ADD$, $2SUB$, SUB, and NEG. Z is the signal for which the multiplexer modifies the multiplicand to output zero. ADD and $2ADD$ are signals for which the multiplexer produces the multiplicand and twice the multiplicand, respectively. The SUB and $2SUB$ control signals allow the multiplexer to generate the complement and complement of twice the multiplicand, respectively. Finally, NEG generates a 0 or a 1 depending upon whether the multiplicand generates a positive or a complemented number. The truth table of the encoder along with the mathematical operations effected by each three-bit sequence of the multiplicand is shown in Table 4.1.

Subtraction can be carried out using 2's complement addition. This involves adding one to the complement of the multiplicand generated by the NEG control signal at the LSB for SUB and $2SUB$ operations. The extra one is generated by the encoding logic. Figure 4.17 shows the schematic for an unsigned 8×8 multiplication. Each multiplexer is ten bits long; eight bits are for the multiplicand and two extra bits at the most significant position are for the sign of the multiplicand and to accommodate the left-shift operations $2ADD$ and $2SUB$.

The operation of the multiplier shown in Fig. 4.17 is illustrated in Fig. 4.18. It must be noted that the input to all the multiplexers is the multiplicand. On the basis of the control signal generated by the encoder, the multiplicand is correspondingly modified by the multiplexer. The first three bits (including the extra bit b_{-1}) are input to encoder 1. A corresponding control signal is given to the input of the multiplexer which, in turn, generates a modified multiplicand that

Multiplier bits			Encoder output						Multiplexer operation
b_{2i+1}	b_{2i}	b_{2i-1}	Z	ADD	$2ADD$	$2SUB$	SUB	NEG	
0	0	0	1	0	0	0	0	0	$+0$
0	0	1	0	1	0	0	0	0	$+A$
0	1	0	0	1	0	0	0	0	$+A$
0	1	1	0	0	1	0	0	0	$+2A$
1	0	0	0	0	0	1	0	1	$-2A$
1	0	1	0	0	0	0	1	1	$-A$
1	1	0	0	0	0	0	1	1	$-A$
1	1	1	1	0	0	0	0	0	-0

Table 4.1: The truth table of the encoder along with the mathematical operations effected by the various three-bit sequences of the multiplier.

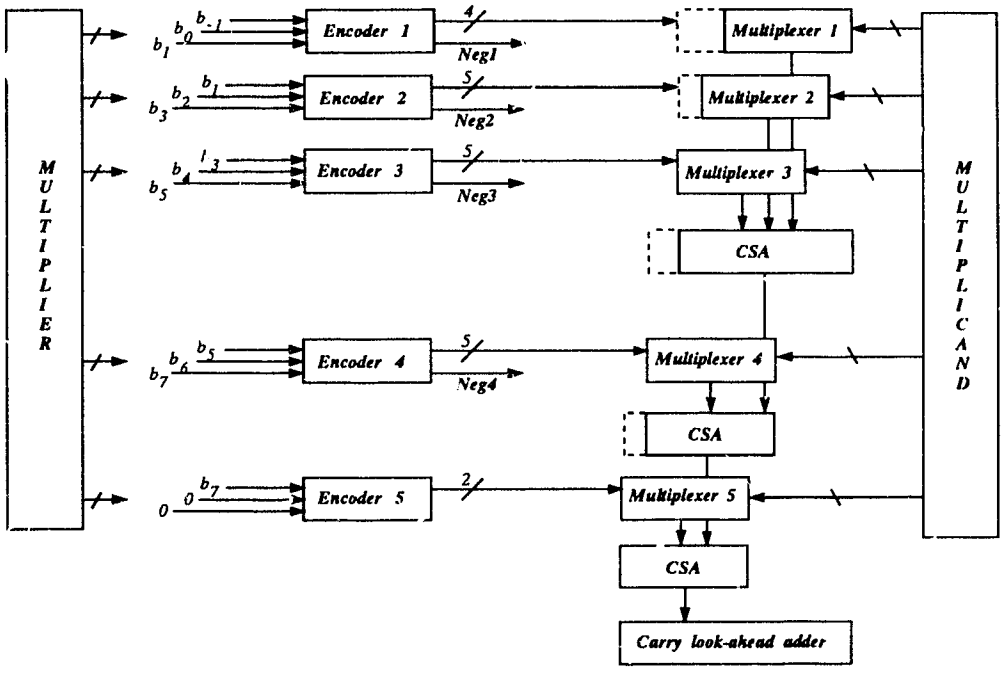


Figure 4.17: An 8×8 conventional CMB multiplier. CSA: Carry-save adder.

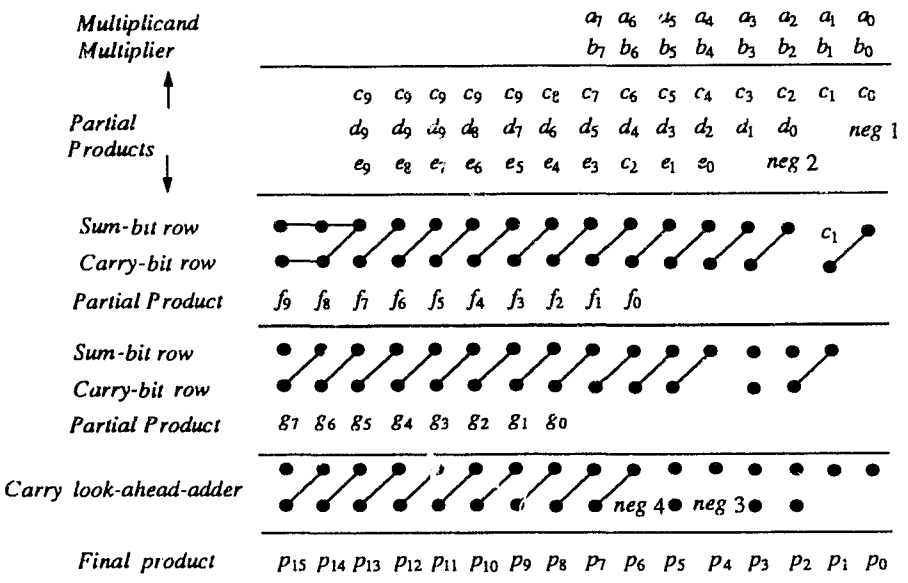


Figure 4.18: Operation of an 8×8 conventional QMB multiplier.

forms the first row of partial products, shown in Fig. 4.18 as c_i. The MSB of this row of partial products is extended by four bits. The next three bits of the multiplier are input to the encoder and a control signal causes the multiplexer to generate the second partial products, d_i, the MSB of which is extended by two bits. The third encoder-multiplexer combination generates the third ten-bit partial product, e_i. The above three partial products are added to produce a row of sum bits and a row of carry bits. It must be mentioned that in Fig. 4.18 a diagonal line joining two dots refers to the sum and carry bits generated by the addition of three bits at that column and a horizontal line joining two dots refers to the extension of the bit from right to left. As can be seen from the figure the sum and the carry rows are sign extended. The sum and carry rows are added to the f_i to produce two rows of bits and so on. In this way, a stage-by-stage addition of the partial products is carried out. The final two rows of sum bits and carry bits are added using a carry look-ahead adder.

Since the dummy bit, b_{-1}, at the least-significant position is always zero, the input to the first encoder can only have sequences 000, 010, 100, and 110 of the

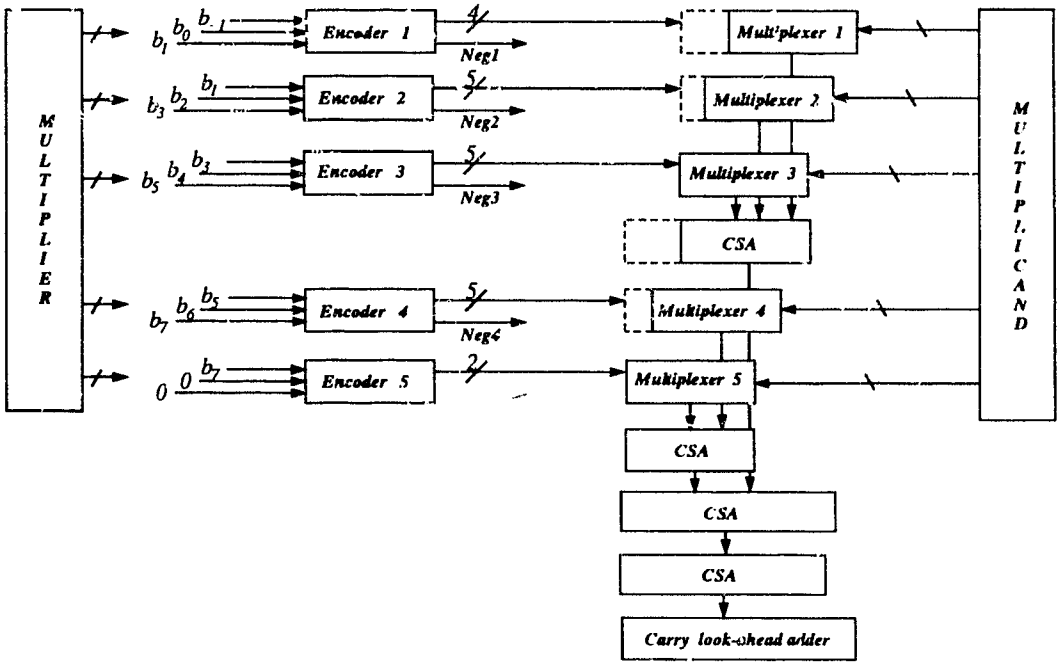


Figure 4.19: An 8×8 parallel QMB multiplier.

multiplier. Thus only the logic necessary to encode these sequences is required. Also since the multiplier is prefixed by two zeros, the input to the last encoder can only have sequences 000 and 001 of the multiplier which correspond to the Z and ADD operations. Two AND gates will be sufficient to implement these sequences.

A parallel implementation of a QMB multiplier was developed in [19]. In this implementation the outputs of a group of multiplexers are added simultaneously. The implementation of an unsigned 8×8 multiplication using this technique is shown in Fig. 4.19.

The operation of this multiplier can be explained using Fig. 4.18. The addition of the partial products c_i, d_i, and e_i can be carried out at the same time as the addition of the partial products f_i and g_i. Each adder produces two rows of bits to be added, one for the sum bits and the other for the carry bits. The reduction from a four-row summand matrix to the final two rows is achieved by using either

the Wallace [57] or the Dadda [58] tree-reduction techniques. Thus there are, in total, three stages of adder and one accelerated carry adder stage. It must be mentioned that there is no difference between the conventional and parallel QMB multipliers in terms of speed and hardware for an 8×8 multiplication.

4.3.2 The OMB multiplier

In the previous section, we saw that the QMB multipliers require four adder stages. It is possible, however, to reduce the adder stage delays by using techniques that reduce the summand matrix size. This can be achieved by using the OMB algorithm. The multiplier can be written as [51]

$$B = \sum_{i=0}^{N/3} O_i 2^{3i} = \sum_{i=0}^{N/3} O_i 8^i \tag{4.28}$$

where

$$O_i = -4b_{3i+2} + 2b_{3i+1} + b_{3i} + b_{3i-1},$$

with $b_{-1} = 0$ and $O_i \in \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$. The product can now be written as

$$P = \sum_{i=0}^{N/3} AO_i 8^i \tag{4.29}$$

where A is the multiplicand. The control signals generated by the encoder using the OMB algorithm are Z, ADD, $2ADD$, $3ADD$, $4ADD$, $4SUB$, $3SUB$, $2SUB$, SUB, and NEG. The control signals are similar to those mentioned for the QMB multiplier. The truth table of the encoder along with the the mathematical operations effected by each four-bit sequence of the multiplicand is shown in Table 4.2.

Figure 4.20 shows an 8×8 OMB multiplier and the operation of this multiplier is illustrated in Fig. 4.21. Here each multiplexer is 11 bits long; 8 bits

Multiplier bits			Encoder Output											Multiplexer operation
b_{3i+2}	b_{3i+1}	b_{3i}	b_{3i-1}	Z	ADD	$2ADD$	$2AUB$	$4ADD$	$4SUB$	$3SUB$	$2UUB$	SUB	NEG	
0	0	0	0	1	0	0	0	0	0	0	0	0	0	$+0$
0	0	0	1	0	1	0	0	0	0	0	0	0	0	$+A$
0	0	1	0	0	1	0	0	0	0	0	0	0	0	$+A$
0	0	1	1	0	0	1	0	0	0	0	0	0	0	$+2A$
0	1	0	0	0	0	1	0	0	0	0	0	0	0	$+2A$
0	1	0	1	0	0	0	1	0	0	0	0	0	0	$+3A$
0	1	1	0	0	0	0	1	0	0	0	0	0	0	$+3A$
0	1	1	1	0	0	0	0	1	0	0	0	0	0	$+4A$
1	0	0	0	0	0	0	0	0	1	0	0	0	1	$-4A$
1	0	0	1	0	0	0	0	0	0	1	0	0	1	$-3A$
1	0	1	0	0	0	0	0	0	0	1	0	0	1	$-3A$
1	0	1	1	0	0	0	0	0	0	0	1	0	1	$-2A$
1	1	0	0	0	0	0	0	0	0	0	1	0	1	$-2A$
1	1	0	1	0	0	0	0	0	0	0	0	1	1	$-A$
1	1	1	0	0	0	0	0	0	0	0	0	1	1	$-A$
1	1	1	1	1	0	0	0	0	0	0	0	0	0	-0

Table 4.2: The truth table of the encoder along with the mathematical operations effected by the various four-bit sequences of the multiplier.

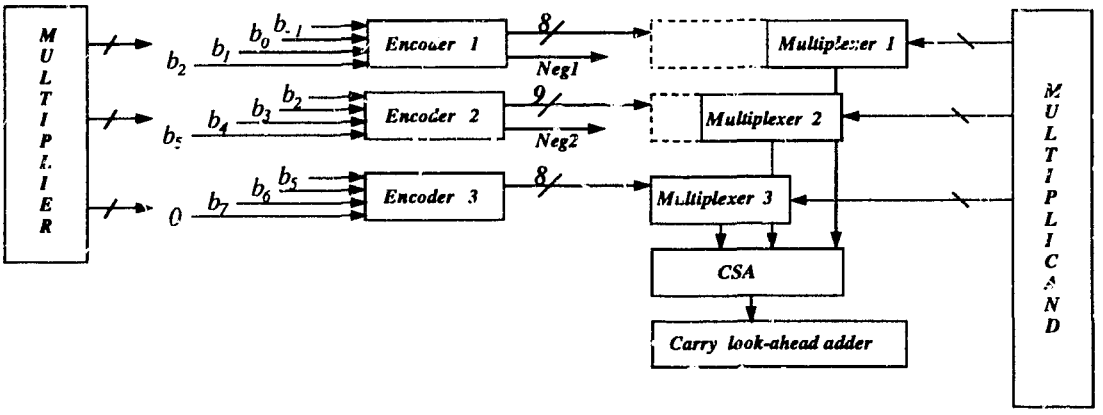


Figure 4.20. An 8×8 OMB multiplier.

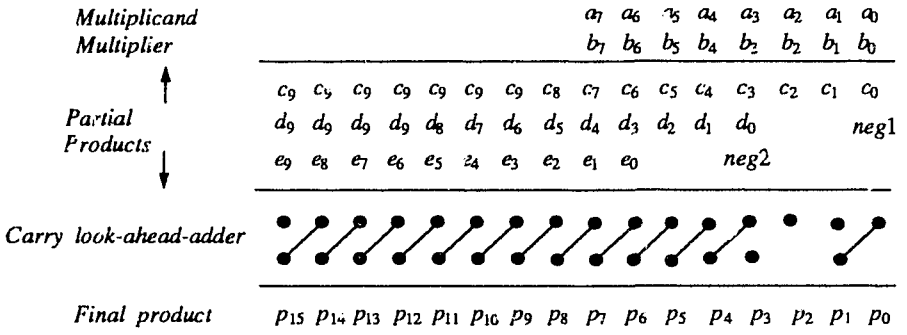


Figure 4.21: Operation of an 8×8 OMB multiplier.

are for the multiplicand and the three extra bits at the MSBs are for the sign of the multiplicand and to accommodate the left-shift for $2ADD$, $2SUB$, $3ADD$, $4ADD$, $3SUB$, and $2SUB$ operations. The operation of three times the multiplicand is not trivial and is accomplished by adding twice the multiplicand with the multiplicand.

Since the encoder for the OMB multiplier scans four-bit segments of the multiplier with one overlap bit, only three encoders and three multiplexers are required to perform an 8×8 multiplication. The outputs from the multiplexers are combined using an adder stage. This adder stage generates a carry word and a sum

word which are then added with a carry look-ahead adder.

There can be some hardware savings in the encoders and the multiplexers. Since the dummy bit at the least-significant position is always zero, the input to the first encoder can only have sequences 0000, 0010, 0100, 0110, 1000, 1010, 1100, and 1110. Thus only the logic necessary to encode these sequences is required. Also since the multiplier is prefixed by a zero, there can be some savings in hardware. As can be seen from Fig. 4.20, an 8×8 multiplier comprises three multiplexers, three encoders and one adder stage before the final addition stage. It must be noted that the multiplexer in Fig. 4.20 has an additional operation to generate three times the multiplicand, thus taking one adder-stage delay. Consequently, we could say that the OMB multiplier takes at most three adder-stage delays to produce the final output. It can be recalled that the QMB multipliers require four adder-stage delays to produce the final result.

The number of adder stages in the OMB multiplier can further be reduced if three times the multiplicand is precalculated instead of generated in every multiplexer. Thus an external carry look-ahead adder can add the multiplicand to the multiplicand shifted left by one bit to generate three times the multiplicand. The structure of such an OMB multiplier is shown in Fig. 4.22. In this way every multiplexer can accept the bits corresponding to three times the multiplicand and output them or their complement depending upon the control signal issued by the encoder. The operation of generating three times the multiplicand takes approximately the same time as the encoder takes to issue a control signal. Thus it can be said that this type of OMB multiplier requires only two adder stages to produce the final result.

From the operation of the QMB and OMB multipliers we could get a rough estimate about the speed and hardware requirements. However, at this stage it may be difficult to obtain an analytical comparison between the two multipliers since several factors like the size of the multiplexers, encoders, number of adders, and the routing strategies are involved. In the next chapter we will compare the

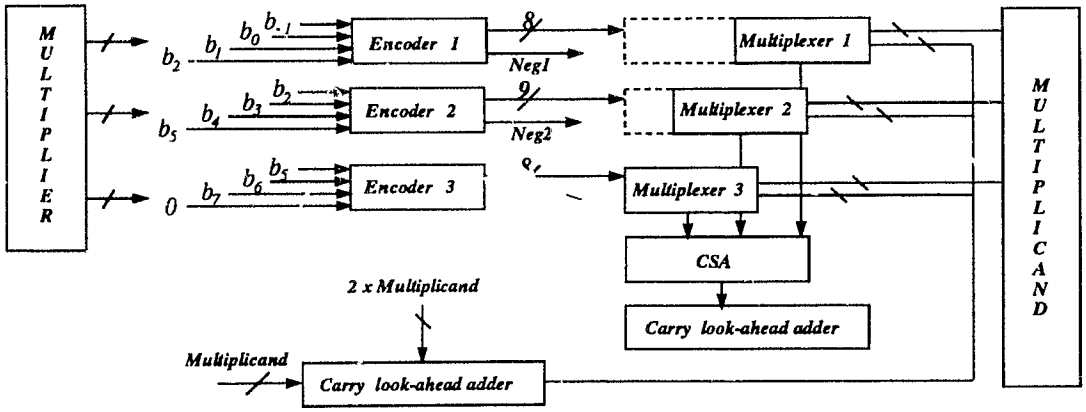


Figure 4.22: A fast 8 × 8 OMB multiplier.

two types of multipliers on the basis of speed, area, and area × time complexity using their actual hardware implementations.

4.3.3 Extension to two's complement multiplication

The multipliers of the preceding section can be easily extended to multiplication of numbers expressed in two's complement. For a two's complement OMB multiplier, only four multiplexers and three adder stages including the final one are required. The size of the multiplexer is 9 bits instead of 10 bits as in the unsigned case, since there is no need for a leading zero in the multiplicand. Strictly speaking, the MSB of the multiplier has to be sign extended up to the full length of the product and, consequently, the number of encoders has to be doubled. Since the inputs to these encoders are either 000 or 111, it is clear that these extra encoders would all be enabling control signal Z. The OMB multiplier, however, incurs only two adder-stage delays to produce the final result.

4.4 Diminished-1 multiplier for Fermat number transform

The most basic operation in digital filters is the convolution. Among the difficulties arising in the calculations of convolutions are the rounding or truncation errors that occur due to finite word length registers employed. These errors can be eliminated by using number-theoretic transforms (NTTs). Various NTTs having structures similar to that of the DFT, but defined in finite fields or rings, have been proposed for the efficient computation of convolutions. Among them, the Mersenne [59] and Fermat number transforms [60] seem particularly attractive. The advantage of these transforms is that the computations can be carried out by using only additions and shifts. A major drawback of this group of transforms is that the maximum sequence length allowed is limited by the choice of word length and is often not long enough for high-resolution convolution computation

Even though the NTTs mentioned above are carried out by just additions and data shifts, general multiplications have to carried out in order to perform convolutions or correlations of two sequences since multiplication of the transforms of the two sequences in question is involved. In the following sections we deal with the multiplication involved in the Fermat NTT. A binary arithmetic that permits the exact computation of the FNT involves arithmetic in a binary code corresponding to the simplest one of a set of code translations from the normal binary representation of each integer in the ring of integers modulo a Fermat number. The translated representation is called diminished-1 representation since the integers in this transform are equal to one less than their binary representation [61]. The normal practice to perform diminished-1 multiplication is to translate the diminished-1 number into a regular binary number and then perform the binary multiplication followed by residue reduction. It will be shown that it is not necessary to translate the diminished-1 number to regular binary number and that the multiplication can be performed in the diminished-1 representation,

thereby reducing the area of the multiplier. This, eventually, has resulted in a novel structure based on the conventional parallel multiplier structure.

4.4.1 Diminished-1 multiplier using the parallel multiplier

Multiplication of two diminished-1 numbers is performed as follows

$$(X \cdot Y - 1) = (X \cdot Y) - 1$$

A diminished-1 multiplication can be accomplished by carrying out the binary multiplication of the two numbers followed by a residue reduction. Residue reduction is accomplished by subtracting the $b+1$ MSBs from the b LSBs of the product of the binary multiplication, where $b+1 = 2^t + 1$ and $F_t = 2^{2^t} + 1$ is the tth Fermat number. In the multiplier advanced in [20] the two numbers, which are represented in their diminished-1 forms, have first to be translated to their binary representations and then multiplied. The product is then residue reduced by performing a diminished 1 subtraction.

We now propose a diminished-1 multiplier in which the use of the translation blocks is circumvented and the multiplication of the two numbers is performed with the numbers in their diminished-1 forms. As a consequence, the multiplier occupies less area and has the same speed as the multiplier proposed in [20].

Let us consider the case where $b = 4$ and express the two diminished-1 numbers as

$$\begin{aligned} X - 1 &= x_3 2^3 + x_2 2^2 + x_1 2^1 + x_0 2^0 \\ Y - 1 &= y_3 2^3 + y_2 2^2 + y_1 2^1 + y_0 2^0 \end{aligned}$$

The product of X and Y can be written as

$$\begin{aligned} P &= X \cdot Y \\ &= (x_3 2^3 + x_2 2^2 + x_1 2^1 + x_0 2^0 + 1) \cdot (y_3 2^3 + y_2 2^2 + y_1 2^1 + y_0 2^0 + 1) \end{aligned}$$

$$= (x_3 2^3 + x_2 2^2 + x_1 2^1 + x_0 2^0) \cdot (y_3 2^3 + y_2 2^2 + y_1 2^1 + y_0 2^0)$$
$$+(x_3 2^3 + x_2 2^2 + x_1 2^1 + x_0 2^0) + (y_3 2^3 + y_2 2^2 + y_1 2^1 + y_0 2^0)$$
$$+1 \qquad (4.30)$$

It is evident from the above equation that a multiplication of two numbers in their diminished-1 forms can be performed by multiplying the two numbers directly in their diminished-1 form, without converting them to the regular binary form, and adding to the result the multiplicand and the multiplier along with a 1. At the same time as the multiplication is performed by the array, a separate carry-look-ahead adder could perform the addition of the multiplier and the multiplicand with the carry input equal to 1.

It is, however, possible to circumvent the use of a carry look-ahead adder that adds the multiplier and the multiplicand. This can be achieved by modifying the conventional multiplier array such that addition of both the multiplier and the multiplicand is handled by the multiplier array. In a parallel multiplier, the first row of components are the AND gates and the second row are the AND gates in conjunction with half adders. If the first and second rows are replaced by AND gates in conjunction with full adders, the multiplicand and the multiplier can be added along with the partial products generated by the multiplier array at those cells. A 1 can be added separately using a column of half adders as shown in Fig. 4.23 for a 4×4 multiplication resulting in a 9-bit product. It can be seen that the array shown in Fig. 4.23 makes best utilization of all the cells that could be used in a multiplier array.

It must be mentioned that the above scheme of multiplication produces correct results when the OR operation of the MSBs of the multiplicand and the multiplier is equal to 0. If, however, the above OR operation produces a 1, this means that either or both the operands to the multiplier are zero. Consequently, the product of the multiplication of the two diminished-1 numbers is zero. Hence, the MSB of the product is 1 and all the other bits of the product are zeros. In order to

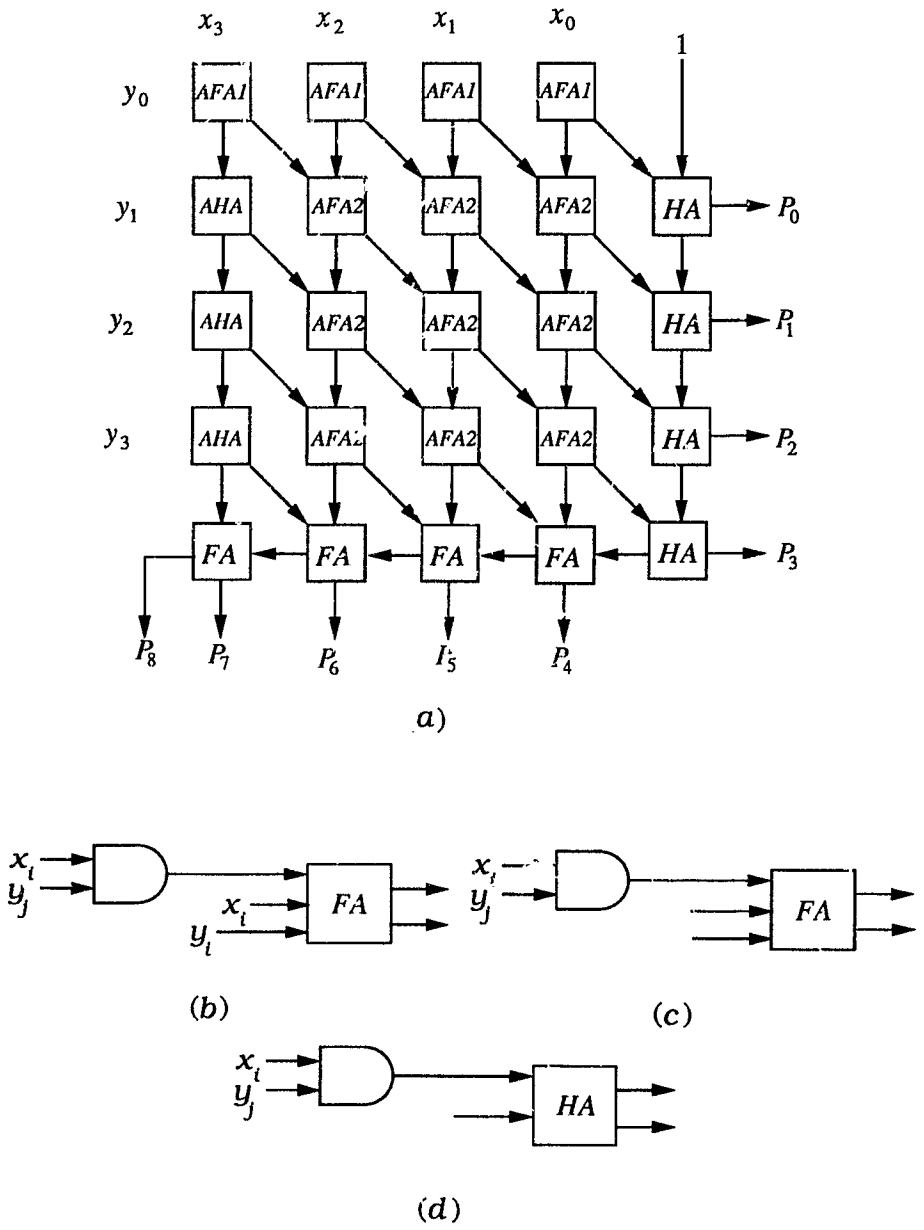


Figure 4.23: A modified 4 × 4 parallel multiplier that yields a 9-bit product. (a) Multiplier block diagram. (b) Details of AFA1 cell. (c) Details of AFA2 cell. (d) Details of AHA cell.

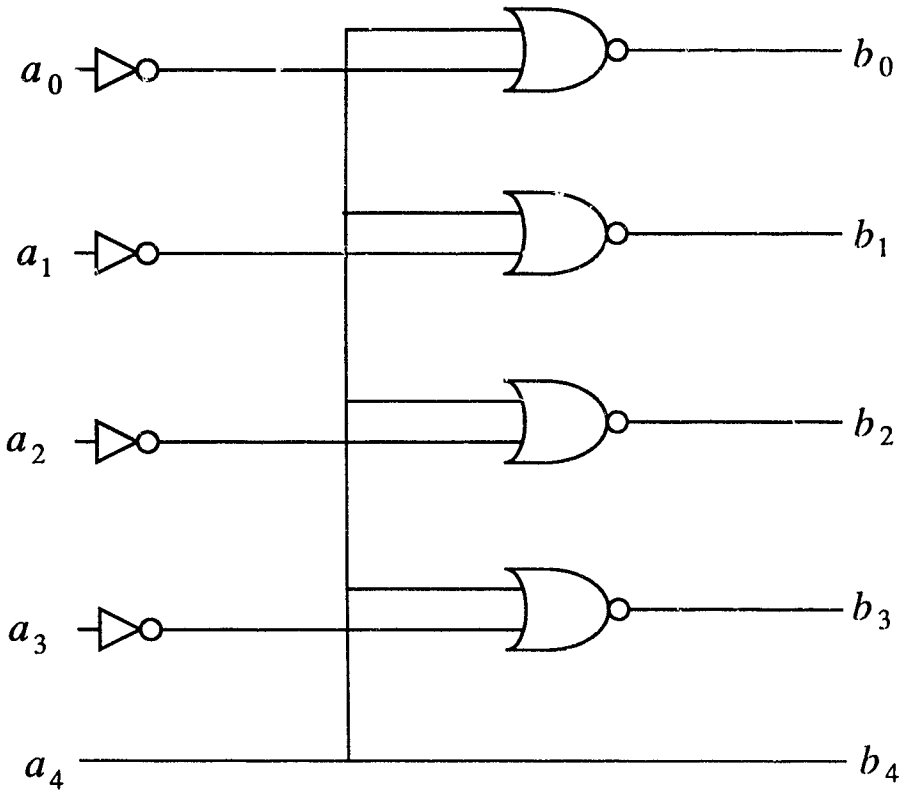


Figure 4.24: Output controller.

take into consideration this operation, we need a unit that outputs the correct diminished-1 product when the OR of the two MSBs of the operand is zero and outputs a zero when the OR operation yields a 1. We shall call such a unit as the output controller and this is shown in Fig. 4.24. The bit a_4 in the figure corresponds to the OR of the MSBs of the two operands.

4.4.2 Residue reduction

The final stage in the multiplication of two diminished-1 numbers is the residue reduction. This is performed by negating the 5 MSBs of the product and performing a diminished-1 addition with the 4 LSBs of the same product. As mentioned earlier, the negation is performed only if the MSB is 0, in which case all the bits except the MSB are complemented. If the MSB is equal to 1, the negation is

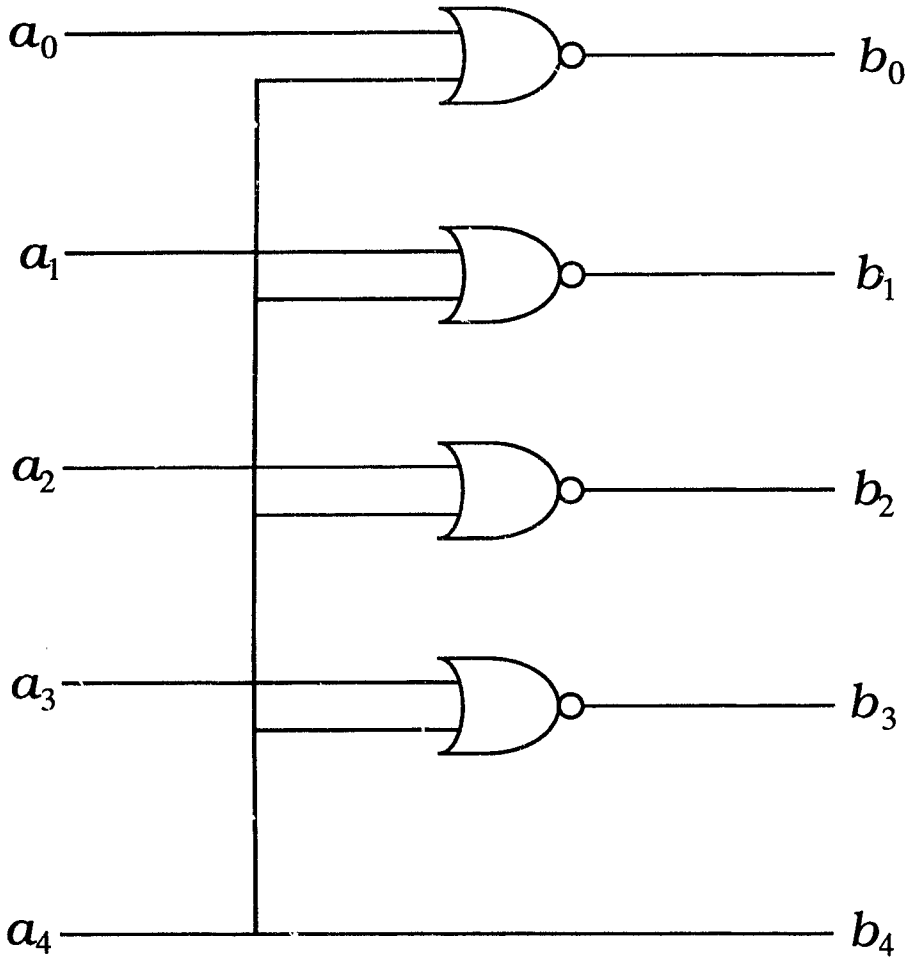


Figure 4.25: Negator.

inhibited. The schematic diagram of the negator is shown in Fig. 4.25.

The diminished-1 adder has four subblocks, viz., propagate/generate block, modified carry block, the sum block, and the full carry-look-ahead block. Such an adder scheme reduces the addition time and regularizes the design and increases the speed of operation [62]. The block diagram of a pipelined diminished-1 multiplier is shown in Fig. 4.26. In this figure all Rs are 4-bit registers except the ones before and after the negator which are 5-bit registers. D in the figure is a 1-bit flip-flop.

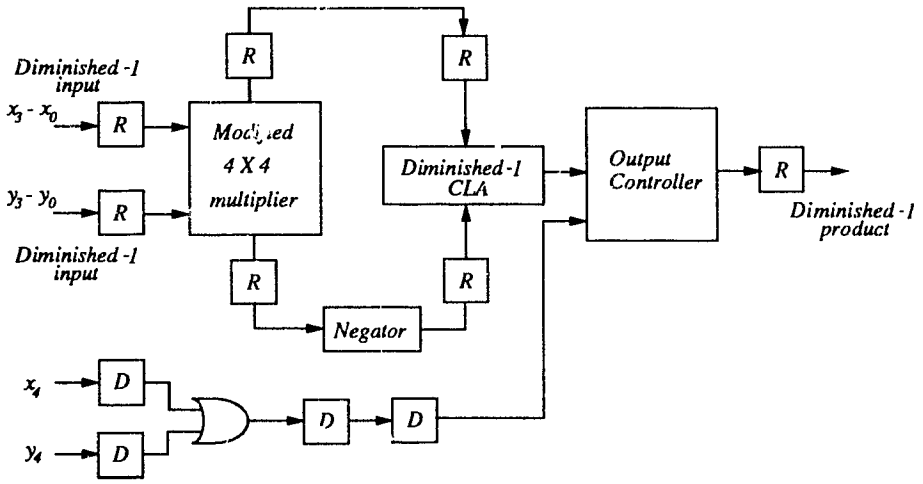


Figure 4.26: Block diagram of a modified 4×4 diminished-1 pipelined multiplier. All the R are 4-bit registers except the ones before and after the negator which are 5-bit registers. D is a 1-bit flip-flop.

4.4.3 Comparison

We shall now compare the diminished-1 multiplier presented in [20] with that proposed here in terms of area and speed where the modulus involved is F_4. The choice of F_4 is favourable since it offers long FNT and the 17-bit word length resulting from the modulo-F_4 arithmetic constitutes a reasonable compromise between dynamic range and chip area.

4.4.3.1 Area

Let A_{fa}, A_{ha}, and A_a denote the areas of a full adder, a half adder, and an AND gate, respectively. In [20] the numbers represented in diminished-1 form are translated to binary form before being multiplied. The areas that are of interest are those occupied by the two translation units and the 17×17 multiplier in [20] and the modified 16×16 multiplier used here. If A_b is the area of the multiplier plus the areas of the two translation units described in [20], then

$$A_b = 255A_{fa} + 51A_{ha} + 321A_a \tag{4.31}$$

The area of the diminished-1 multiplier proposed here is

$$A_o = 256A_{fa} + 32A_{ha} + 273A_a \tag{4.32}$$

where the area occupied by the output controller is approximately $17A_a$. The pipelined diminished-1 multiplier in [20] has two registers that store the output of the translation units. Since these are not present in the proposed multiplier, there is an additional saving in area in the proposed multiplier. Assuming that the area of a full adder is three times that of a half adder and the area of a 34-bit register is A_r, the savings in area in the proposed multiplier, A_s, is

$$A_s = A_b - A_o = 16A_{ha} + 48A_a + A_r \tag{4.33}$$

4.4.3.2 Speed

If T_{fa}, T_{ha}, and T_a are the propagation delays of a full adder, a half adder and an AND gate, respectively, the propagation delay in the diminished-1 multiplier of [20] is given by,

$$T_b = 30T_{fa} + 2T_{ha} + 17T_a \tag{4.34}$$

The propagation delay in the proposed multiplier is given by

$$T_o = 31T_{fa} + 2T_{ha} + 16T_a \tag{4.35}$$

The difference in the propagation delays of the two multipliers is

$$T_s = T_o - T_b = T_{fa} - T_a \tag{4.36}$$

The value of T_s is negligible as compared to the speed of operation of the multiplier. Moreover, since the area of the multiplier of [20] is much more than that of the proposed multiplier, more buffers may be required to drive the clock and internal signals thereby reducing the speed of operation, and as a consequence the small difference in the propagations delays of the two multipliers is more than compensated for.

4.5 Accumulator-multiplier

The conventional way to obtain an inner product is to use a multiplier-accumulator (MAC) where each multiplication is followed by an addition of the number stored in the accumulator. In other words, an operation of the kind $y_i = ax + y_{i-1}$, where a and x are any two numbers, is performed. The output is obtained when the final pair of numbers are multiplied and added to the result stored in the accumulator.

Here we propose a structure which performs an inner product that does not use an accumulator. In a MAC, the number at the output of the multiplier is added to the number in the accumulator and stored back in the accumulator. If

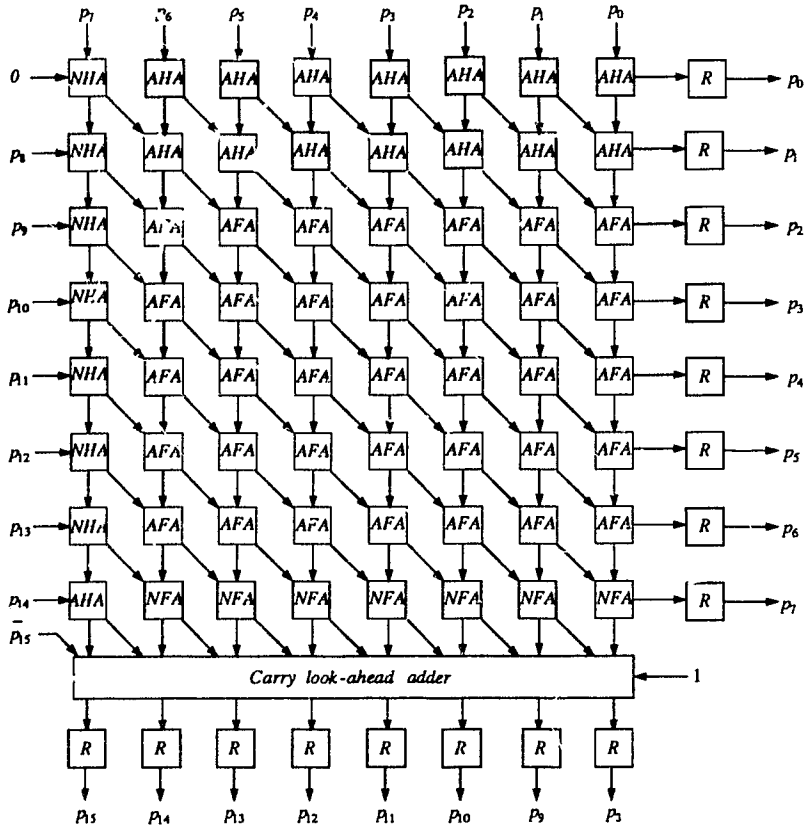


Figure 4.27: An 8×8 accumulate-multiplier.

however, the number to be added is present at the input of the multiplier at the time of multiplication, we could circumvent the use of the accumulator. We call such a multiplier as the accumulator-multiplier (ACM) for obvious reasons. Such an 8×8 two's complement multiplier is shown in Fig. 4.27. In this figure the cell NHA is the same as that shown in Fig. 4.10(b) except that the full adder FA is replaced by a half adder HA.

It can be seen from the figure that the top row of cells are composed of AND gates in conjunction with half adders and the left column of cells are composed of NAND gates in conjunction with half adders. These gates generate the partial products that are added to the bits stored in the 1-bit registers present at the output of the multiplier. To initiate the computation of an inner product, the

output registers are reset and the first multiplication is performed and the product stored in the output registers. At the next clock cycle the bits of the two operands are input to the multiplier along with the bits from the output registers. Such a process is repeated till the result of the inner product is obtained.

It must be mentioned here that in the ACM, the number stored in the registers will keep increasing as the process of addition-multiplication continues. Even if the numbers are represented in fractional sign-magnitude form, the number stored in the registers may exceed 1. However, in digital filter applications where the signals and coefficients are represented in fractional two's complement representation, the overflow carry in the result can be neglected, except in the last multiplication-addition process. In Chapter 6, we shall make use of this multiplier in the design of second-order digital filter by incorporating a overflow detection unit.

4.5.1 Comparisons

Let us now compare the area and speed of the ACM with those of the conventional MAC.

4.5.1.1 Area

The area of the MAC is given by

$$A_{ma} = N^2 A_a + (N-1)A_h + (N-1)(N-2)A_f + A_{cla1} + 2NA_l + A_{cla2} \quad (4.37)$$

where A_{cla1} is the area of an $(N-1)$-bit carry look-ahead adder, A_{cla2} is the area of a $(2N-1)$-bit carry look-ahead adder, and A_l is the area of a 1-bit latch. The area of the ACM is

$$A_{am} = N^2 A_a + (3N-2)A_h + (N-1)(N-2)A_f + A_{cla2} + 2NA_l \quad (4.38)$$

where A_{cla3} is the area of an N-bit carry look-ahead adder. The savings in area can be written as

$$A_s = A_{cla2} - (2N-1)A_h + A_{cla1} - A_{cla3} \quad (4.39)$$

If we assume that the area of a N-bit carry look-ahead adder is equal to that of an $(N-1)$-bit carry look-ahead adder, we obtain $A_s \gg 0$. This means that the area occupied by the ACM is much smaller than that by the MAC structure.

4.5.1.2 Speed

Because of the presence of an extra row of half adders in the ACM, the delay is more than that in the MAC by an amount equal to the delay through a half adder. As can be noted from above, the area of the MAC is larger, and consequently, extra buffers may be required to drive certain signals. Therefore, this small difference in speed is more than compensated for.

4.6 Conclusions

In this chapter several multiplier schemes used in digital signal processing have been discussed. The truncated multiplier finds applications in digital filters where the input and the output are of equal lengths. It has been shown by way of a design example that the performance of the truncated multiplier in better than the standard multiplier is terms of area and gives the same results as the standard multiplier. This shows that the truncated multiplier can be a preferred candidate in high-speed area-efficient digital filters. Pipelined truncated multipliers for unsigned and two's complement multiplications have also been discussed.

The second multiplication scheme considered is based on the QMB and OMB algorithms. In the case of conventional and parallel QMB multipliers, the total delay to produce the final product for the case of an 8×8 multiplication is equal to four adder-stage delays. It has been shown that for the OMB multiplier the number of stages required is two. In addition, it has been shown that the area $\times$ complexity of the OMB multiplier is less than that of the QMB multiplier. Thus such a multiplier could be used in high-speed signal processing applications.

The third multiplication scheme deals with the multiplication of two num-

bers represented in their diminished-1 forms. Multiplication of two diminished-1 numbers involves the translations of each of these numbers to their regular binary form. It has been shown that the translation stage can be circumvented and an accumulate-multiply type multiplier can perform the multiplication of two diminished-1 numbers The proposed diminished-1 multiplier occupies less area and is as fast as that presented in [20].

Finally, an ACM has been designed that performs an inner product operation without the use of an accumulator. The ACM structure occupies less area and is as fast as the conventional MAC.

Chapter 5

VLSI implementation of multipliers

5.1 Introduction

A standard-cell library offers the designer a number of logic gates that are sufficient to implement any digital system. The cells are handcrafted, designed, simulated, laid out, tested and characterized. All the cells have equal heights and the cell boundaries can be abutted without conflict. The cells can be manually or automatically routed. Timing verification of a standard-cell design is usually carried out using a gate level simulator such as SILOS. The standard-cell based approach strikes a compromise between full custom and gate array approaches. The design time is intermediate between those of a gate array and a full custom design and the performance is very close to that of a full custom approach. The standard-cell approach also results in better silicon utilization as compared to gate arrays.

The CAD tool used in this thesis is the Cadence package which is a standard one adopted by the Canadian Microelectronics Corporation (CMC). This package has a minimal cell library that comprises all the basic gates and certain macros. CMC makes available to its member Universities two CMOS processes namely,

1. The 3μ CMOS3DLM double-metal process

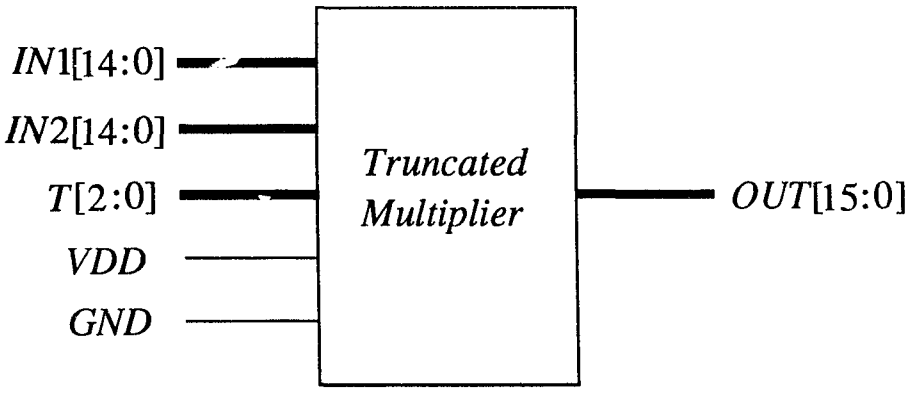


Figure 5.1: Icon for the truncated multiplier.

2. The 1.2μ CMOS4S double-metal process.

As a part of this thesis we have used the CMOS4S process to implement our chips.

In this chapter we shall discuss the VLSI implementation of three multipliers viz., the truncated multiplier, the OMB multiplier, and the diminished-1 multiplier that are described in Chapter 4.

5.2 Implementation of the truncated multiplier

5.2.1 Features

This is a 16 $\times$ 16 truncated multiplier that accepts parallel input and produces parallel output. Inputs and outputs are assumed to be latched by external flip-flops. In other words, the truncated multiplier is a purely combinational circuit.

5.2.2 Icon and block diagram of the truncated multiplier

An icon for the truncated multiplier is shown in Fig. 5.1. Figure 5.2 shows the block diagram. The various signals are as follows.

IN1[14:0] Multiplier input bus of 15 lines.

IN2[14:0] Multiplicand input bus of 15 lines.

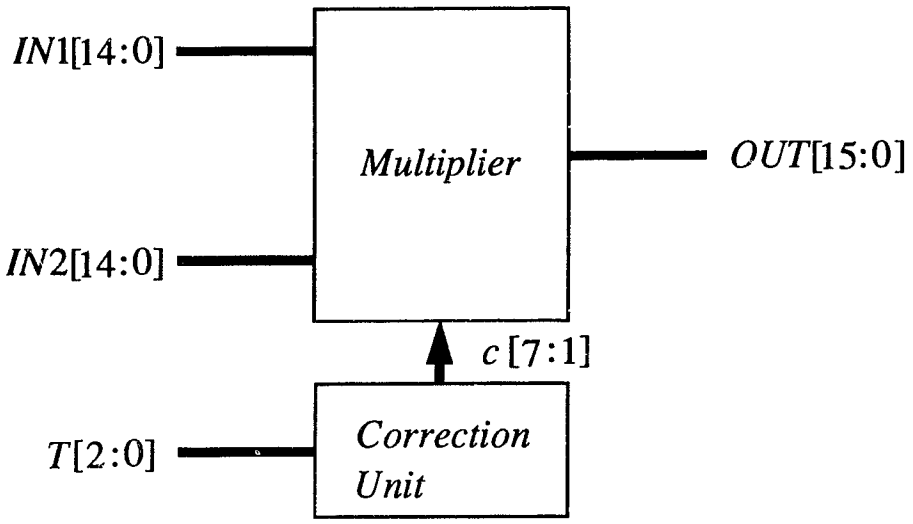


Figure 5.2: Block diagram for the truncated multiplier.

VDD Power bus line.

GND Ground bus line.

OUT[15:0] Multiplier output bus of 16 lines.

5.2.3 Functional description of the block diagram

The two units in the block diagram of the truncated multiplier are the multiplier and the correction unit.

Multiplier An 8×8 truncated multiplier is shown in Fig. 4.5. Bits $c_1 - c_7$ are the correction bits that are added to the adder cells placed at the diagonal of the multiplier array. It must be mentioned here that the multiplier is partitioned using the NMM technique and the partial products are reduced to the final two rows of bits by the Dadda tree-reduction scheme [58]. The final two rows of bits are added using a carry look-ahead adder.

Correction unit As has been mentioned in Chapter 4, the maximum error that can occur in the truncated multiplier is equal to $(N-1)2^{-N}$ where N is the size of the multiplier. However, from the graph of the variation of the expected

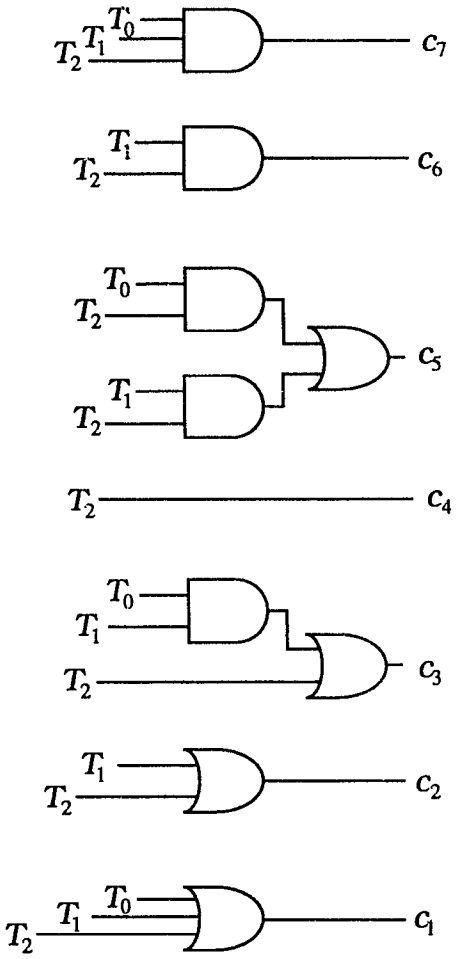


Figure 5.3: Schematic diagram of the correction unit.

value of the error it can be seen that the error in a 16×16 truncated multiplier is only in the two LSBs. We have incorporated a correction unit that outputs seven bits $c_1 - c_7$ that can be added to the diagonal cells. Figure 5.3 shows the schematic diagram of the correction unit.

5.2.4 Timing diagram

Figure 5.4 shows the timing of the various signals in the truncated multiplier. The inputs are supplied from external registers.

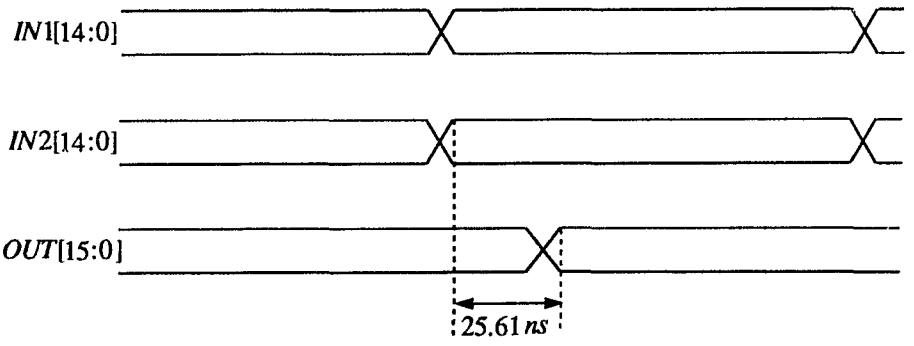


Figure 5.4: Timing diagram for the truncated multiplier.

Characteristics	Truncated multiplier	Parallel multiplier
Technology	1.2μ NTE CMOS4S	1.2μ NTE CMOS4S
Area of the chip	4818.95 $\times$ 4818.95 μm^2	7444.45 $\times$ 7444.45μm^2
Area of the core	2251.83 $\times$ 2251.83μm^2	3132.03$\times$ 3132.03μm^2
No. of pins	57	74
Transistor count	3164	5966
Propagation delay	25.61 ns	27.1 ns
Area$\times$time	1.2986e+08 μm^2ns	2.6584e+08 μm^2ns

Table 5.1: Chip statistics of the truncated and parallel multipliers.

5.2.5 Physical characteristics

The truncated multiplier is compared with the conventional parallel multiplier in Tables 5.1 and 5.2. As can be seen the speed of the truncated multiplier is approximately the same as that of the parallel multiplier. The truncated multiplier occupies only 51% of the area of the parallel multiplier. This figure is in very close agreement with the theoretical calculation carried out in Chapter 4.

5.2.6 Layout of the truncated multiplier

Figure 5.5 shows the complete photomicrograph of the 16$\times$16 truncated multiplier.

Gate	Gate count for the Truncated multiplier	Gate count for the Standard multiplier
Inverter	182	354
Buffer	117	212
Nand2	488	979
Nand3	21	27
Nand4	8	10
Nor2	26	35
Nor3	9	12
Nor4	4	5
Gate count	855	1634

Table 5.2: Gate-level statistics of the truncated and parallel multipliers.

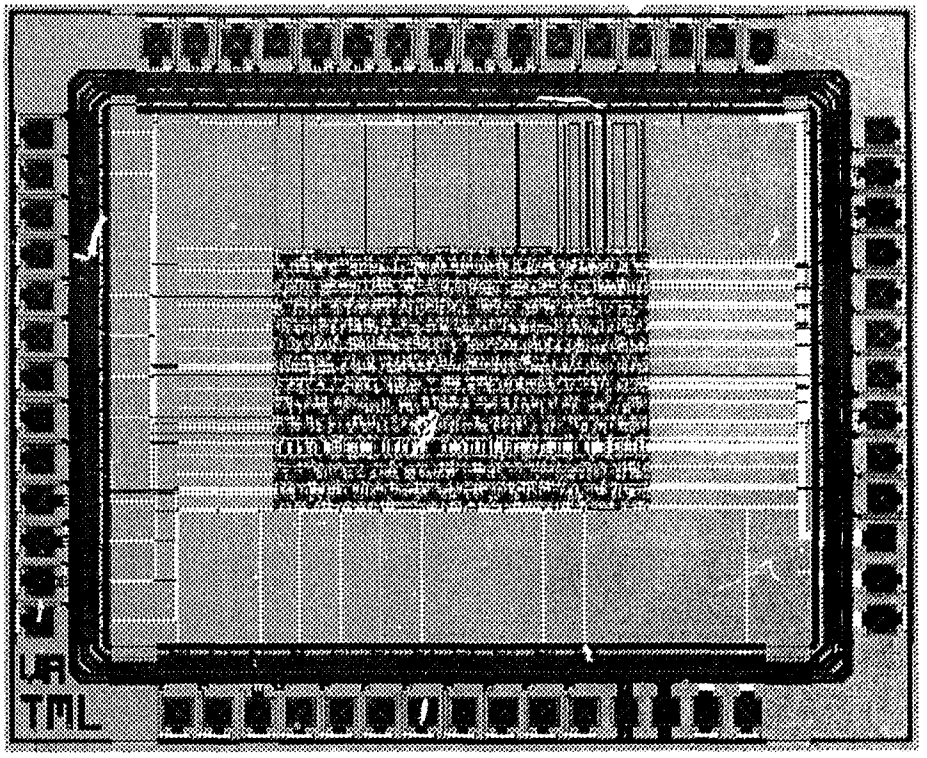


Figure 5.5: Photomicrograph of the 16 × 16 truncated multiplier.

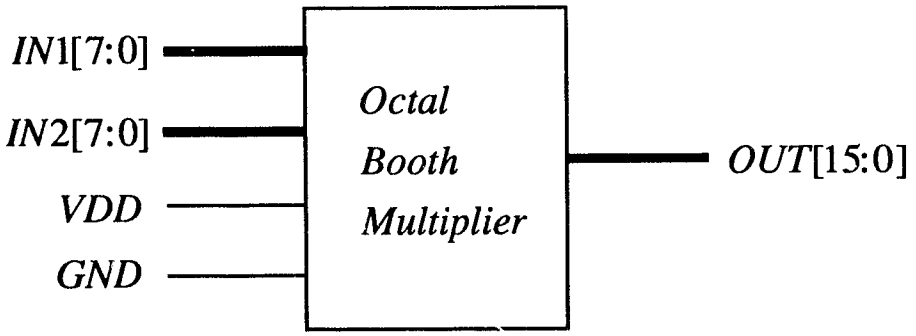


Figure 5.6: Icon for the OMB multiplier.

5.3 Implementation of the OMB multiplier

5.3.1 Features

This is an 8×8 multiplier based on the modified octal Booth algorithm. It accepts parallel input and produces parallel output. Inputs and outputs are assumed to be latched by external registers. In other words, the OMB multiplier is a purely combinational circuit.

5.3.2 Icon and block diagram of the OMB multiplier

An icon for the OMB multiplier is shown in Fig. 5.6. See Fig. 4.22 for the detailed block diagram. The various signals are as follows.

IN1[7:0] Multiplier input bus of 8 lines.

IN2[7:0] Multiplicand input bus of 8 lines.

VDD Power bus line.

GND Ground bus line.

OUT[15:0] Multiplier output bus of 16 lines.

5.3.3 Functional description of the block diagram

The two major units in the block diagram of the OMB multiplier are the encoder and the multiplexer.

Encoder The encoder accepts four inputs as shown in Fig. 4.22 and outputs ten control signals based on the truth table shown in Table 4.2. The schematic diagram of a typical encoder is shown in Fig. 5.7.

Multiplexer The multiplexer accepts the multiplicand as the input and outputs the modified multiplicand according to the control signal received from the encoder. Basically, the multiplexer comprises nine sets of tri-state gates, each set consisting of eleven tri-states gates. The multiplexer generates one, two, three and four time(s) the multiplicand. Three times the multiplicand is generated by the carry look-ahead-adder and fed to the multiplexer as shown in Fig. 4.22. In addition, the multiplexer generates the complement of all the bits of one, two, three, and four times the multiplicand. Thus if the control signals $3SUB$ and NEG from the encoder are active, the complement of all the bits of three times the multiplicand are output by the multiplexer. The NEG signal is added to the LSB of the complement of the bits of three times the multiplicand.

5.3.4 Timing diagram

Figure 5.8 shows the timing of the various signals in the OMB multiplier. Both the input and output signals are assumed to be stored by external registers.

5.3.5 Physical characteristics

The OMB multiplier is compared with the conventional QMB multiplier in Tables 5.3 and 5.4. It can be seen that the OMB multiplier has a lower area $\times$ complexity compared to that of the QMB multiplier.

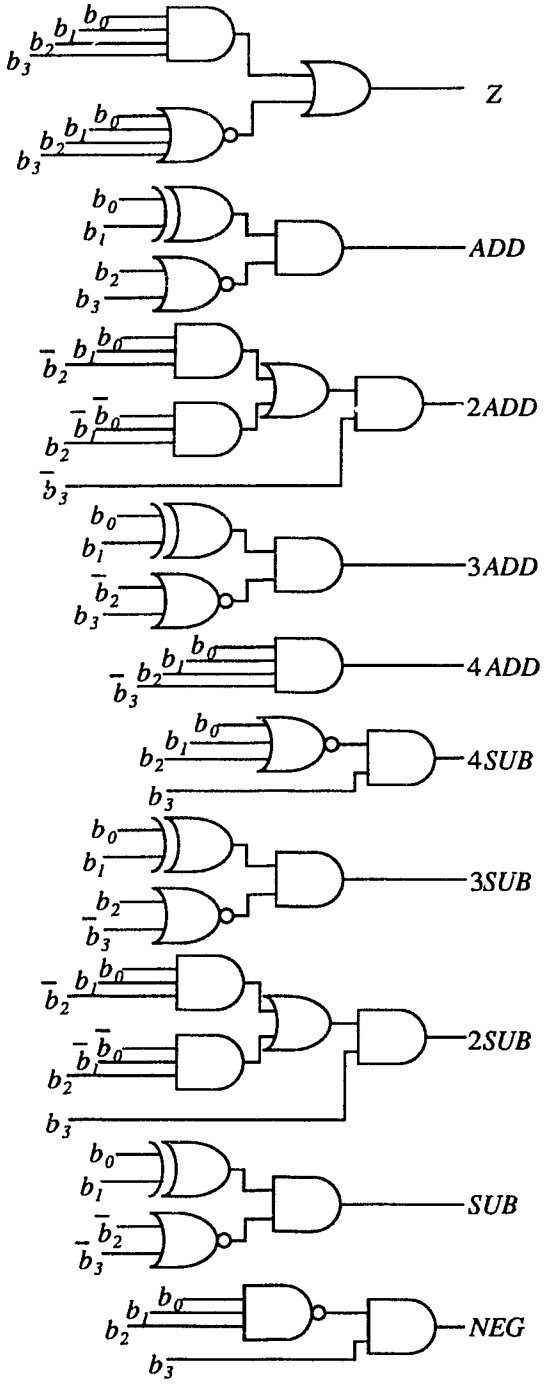


Figure 5.7: Schematic diagram for the octal encoder.

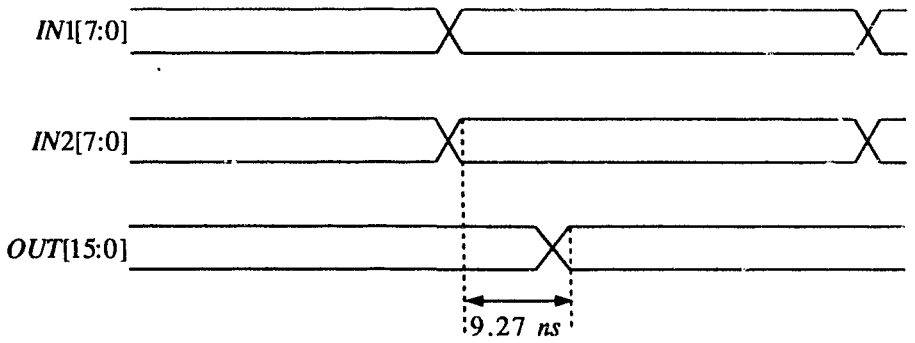


Figure 5.8: Timing diagram for the OMB multiplier.

Characteristics	OMB multiplier	QMB multiplier
Technology	1.2μ NTE CMOS4S	1.2μ NTE CMOS4S
Area of the chip	3626.7 $\times$ 3626.7 μm^2	3626.7 $\times$ 3626.7 μm^2
Area of the core	2026.43 $\times$ 2026.43 μm^2	1920.91 $\times$ 1920.91 μm^2
No. of pins	38	38
Transistor count	4202	3492
Propagation delay	9.27 ns	13.17 ns
Area$\times$time	3.8066e+07 μm^2ns	4.8595e+07 μm^2ns

Table 5.3: Chip statistics of the OMB and QMB multipliers.

Gate	Gate count for the OMB multiplier	Gate count for the QMB multiplier
Inverter	191	157
Buffer	196	139
Tri-state	237	206
Nand2	121	152
Nand3	30	18
Nand4	15	6
Nor2	40	24
Nor3	22	15
Nor4	8	3
Gate count	860	720

Table 5.4: Gate-level statistics of the OMB and QMB multipliers.

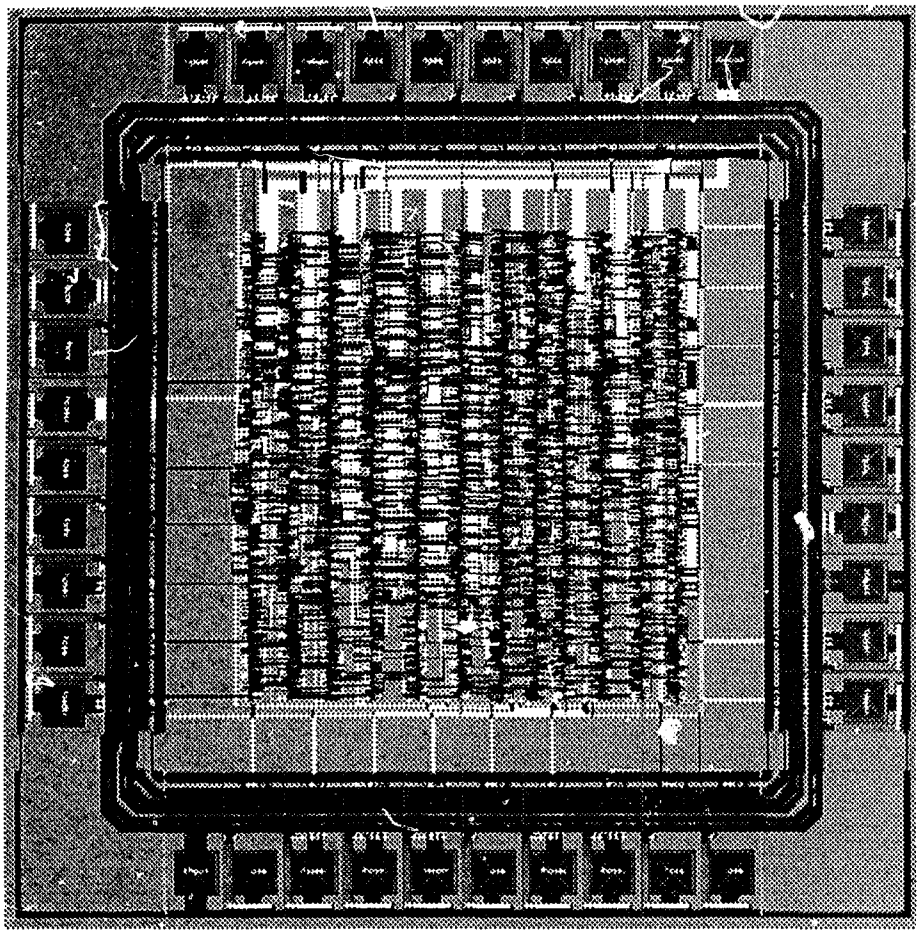


Figure 5.9: Photomicrograph of the OMB multiplier.

5.3.6 Layout of the OMB multiplier

Figure 5.9 shows the complete photomicrograph of the OMB multiplier.

5.4 Implementation of a diminished-1 multiplier

5.4.1 Features

This is a modified 16×16 diminished-1 multiplier that performs the multiplication of two numbers represented in their diminished-1 forms. It accepts parallel inputs and produces parallel outputs. The multiplier consists of four pipeline stages.

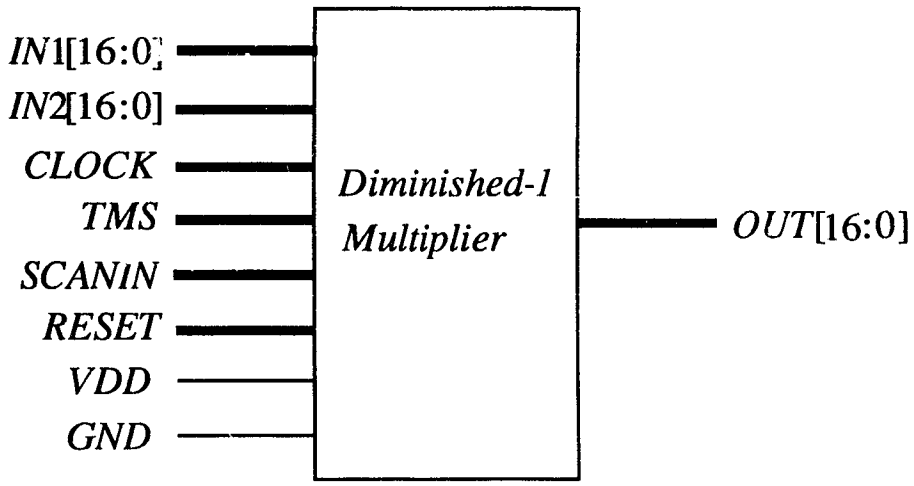


Figure 5.10: Icon for the diminished-1 multiplier.

5.4.2 Icon and block diagram of the diminished-1 multiplier

An icon for the diminished-1 multiplier is shown in Fig. 5.10. See Fig. 4.26 for a detailed block diagram. The different signals are as follows.

IN1[16:0] Multiplier input bus of 17 lines.

IN2[16:0] Multiplicand input bus of 17 lines.

OUT[16:0] Diminished-1 multiplier output bus of 17 lines.

CLOCK Clock signal line.

TMS Test mode select pin. When TMS is 1 all the registers are cascaded end-to-end in a scan-path fashion. When TMS is 0 all the registers are configured in the normal way, i.e., the registers accept data in parallel and output data in parallel.

SCANIN In the test mode when TMS is 1, the scan-path signals are input through this pin.

RESET A 1 on this pin resets all the registers in the multiplier.

VDD Power bus line.

GND Ground bus line.

5.4.3 Functional description of the block diagram

As can be seen from the diminished-1 multiplier of Fig. 4.26, the main units are the modified multiplier, the negator, and the diminished-1 adder. The details of the diminished-1 adder can be found in [62]. Figure 4.23 shows the diminished-1 multiplier that performs the multiplication of two diminished-1 numbers modulo F_2. The multiplier constructed here performs the multiplication modulo F_4.

The output controller outputs the product of the multiplication of two diminished-1 numbers if the OR operation of the MSBs of the two operands yields a 0 and outputs zero in diminished-1 representation when the OR operation yields a 1. Figure 4.25 shows the negator block diagram. As can be recalled the negator complements the 17 MSBs of the product if the MSB of the product is 0 and inhibits the complement action when the MSB is 1.

5.4.4 Timing diagram

Figure 5.11 shows the timing of the various signals in the diminished-1 multiplier. The multiplier is pipelined and has a latency of four, which is the depth of the pipeline.

5.4.5 Physical characteristics

The physical characteristics of the diminished-1 multiplier are given in Tables 5.5 and 5.6. As can be seen the maximum speed of operation of this multiplier is approximately 25 MHz.

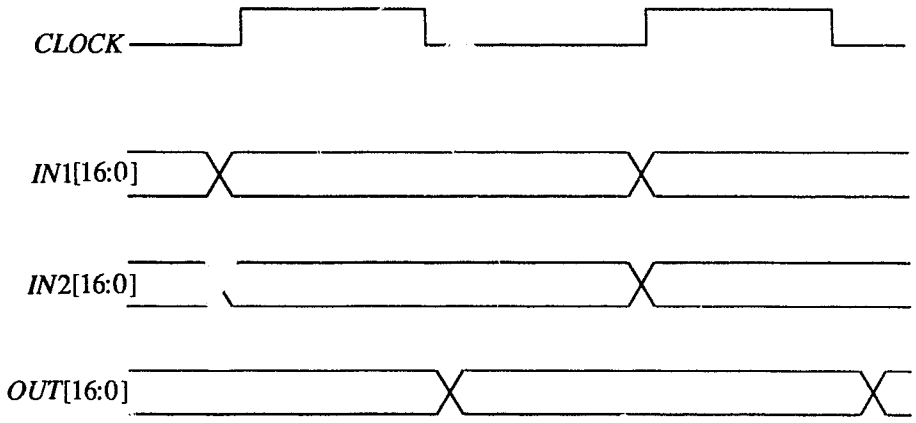


Figure 5.11: Timing diagram of the diminished-1 multiplier.

Diminished-1 multiplier	
Technology	1.2μ NTE CMOS4S
Area of the chip	6137.45x6137.45 μ^2m
Area of the core	4089.02x4089.02 μ^2m
No. of pins	63
No. of transistors	13818
Stage delay	40.2 ns
Area $\times$ time	6.7214e+08 nsμ^2m

Table 5.5: Chip statistics of the diminished-1 multiplier.

Diminished-1 multiplier	
Gate	Gate count
Inverter	498
Buffer	383
D-Flipflop	248
Nand2	1182
Nand3	57
Nand4	22
Nor2	89
Nor3	24
Nor4	11
Gate count	2514

Table 5.6: Gate-level statistics of the diminished-1 multiplier.

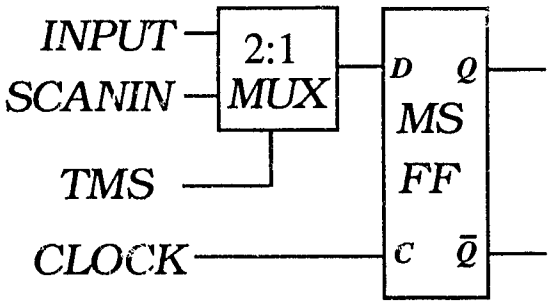


Figure 5.12: Flip-flop used in the scan-path.

5.4.6 Testing strategy

In order to check the pipeline, all the registers used are of the type which accept data depending on the signal of the TMS pin. When the TMS is 1, all the registers are configured in a scan-path mode. A typical flip-flop used in the scan-path mode is shown in Fig. 5.12. The scanin pin is used to input the scan-path signals and the output is obtain from the LSB of the final output. When TMS is 0, all the registers accept data in parallel and output data in parallel.

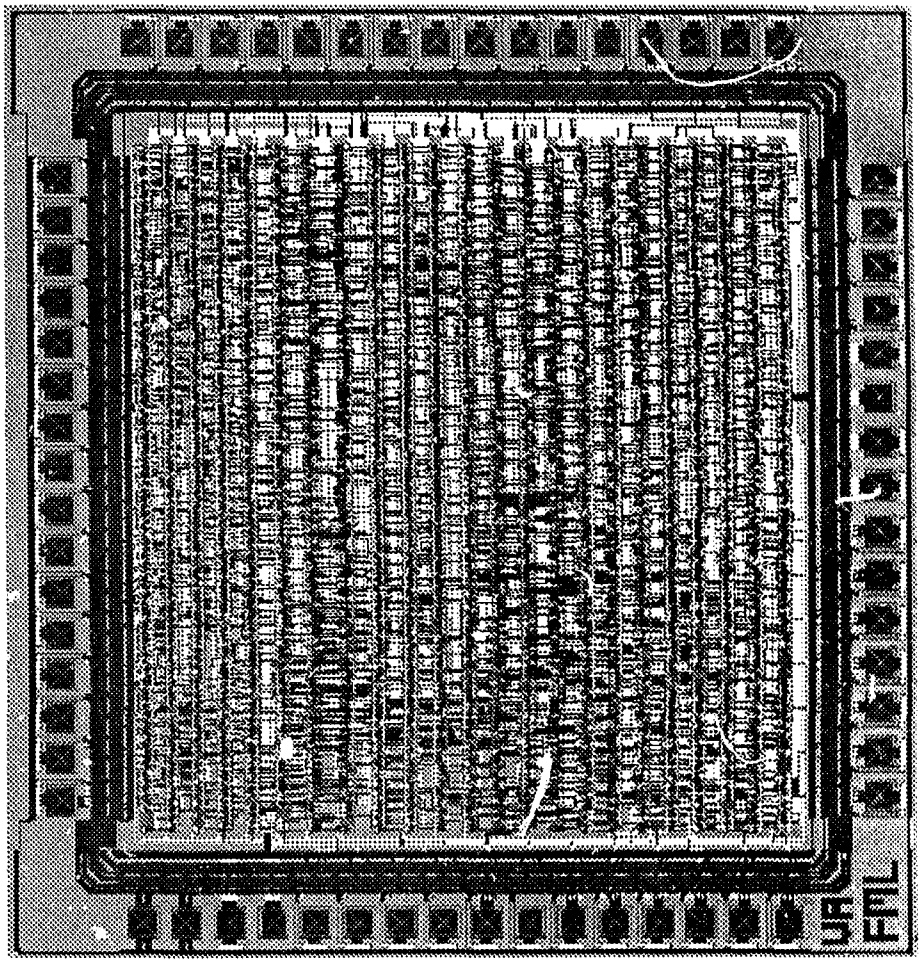


Figure 5.13: Photomicrograph of the diminished-1 multiplier.

5.4.7 Layout of the diminished-1 multiplier

Figure 5.13 shows the complete photomicrograph of the diminished-1 multiplier.

5.5 Conclusions

In this chapter the implementations of three of the multipliers described in Chapter 4, namely, the truncated multiplier, the OMB multiplier, and the diminished-1 multiplier, have been discussed. These chips have been implemented using the 1.2μ CMOS4S technology. The operation of the chips has been verified using the

SILOS simulator. The truncated multiplier has been compared with the full parallel multiplier in terms of speed and area. It has been found that the truncated multiplier occupies about 51% of the area of a full parallel multiplier while operating at the same speed. The OMB multiplier has been found to have a lower area $\times$ time complexity than the conventional QMB multiplier. This suggests that the OMB is a good candidate for use in high-speed signal processing applications. A diminished-1 multiplier that occupies less area and having lower latency than those in [20] has also been implemented.

Chapter 6

VLSI implementation of filters

6.1 Introduction

In this chapter we implement second-order filter of Fig. 2.14 by making use of the ACM, discussed in Section 4.5, as an iterative multiplier to compute the inner product involved in the filter operation. Circuits to circumvent quantization and overflow limit cycles are incorporated in the filter structure. The systolic PE of Fig. 2.11(b) that can be used as a basic unit in 1-D and M-D digital filters has also been implemented. Comparisons of the second-order filter of Fig. 2.14 and that of Fig. 2.15 built using the PE of Fig. 2.11(b) are made.

6.2 Background

Software as well as hardware digital-filter implementations require data to be stored in finite-length registers [37]. Consequently, coefficients and signal values must be quantized by rounding or truncation before they can be stored. The effect of signal quantization is to add an error or noise signal to the output of the digital filter. This noise is a composite of the errors from one or more of the following sources.

1. The quantization error of the analog-to-digital converter at the filter input. This is referred to as the *input-quantization error.*

2. The transfer function coefficients are normally evaluated to a high degree of precision during the approximation step. If they are quantized, the frequency response of the resulting filter may be different from the desired response. This type of error is referred to as *coefficient-quantization error.*

3. The accumulated errors result from the rounding or truncation of internal arithmetic operations in the filter. This type of error is referred to as the *product-quantization error.*

Source (1) is often ignored as it does not fall into the core of digital filter design. Coefficient-quantization errors introduce perturbations in the zeros and poles of the transfer function, which in turn manifest as errors in the frequency response. This type of error can be made insignificant by calculating the word length required for the coefficients for a given set of specifications [37].

If numbers are represented using fixed-point format then, as a consequence of product quantization, recursive filter structures can produce a nonzero output for zero or constant input. Such outputs are called *quantization limit-cycle oscillations.*

There is yet another type of limit-cycle oscillations called *overflow limit-cycle oscillations* which is due to the fact that the addition operation in fixed-point arithmetic can cause overflow, thereby creating a severe nonlinearity. For a stable direct-form filter, the output of the filter after an internal arithmetic overflow can (depending on the poles of the filter) become independent of the input sequence [63].

Overflow oscillations can, to a large extent, be avoided by applying strict scaling rules [64]. The problem with this approach is that signal levels throughout the filter are low and, as a result, a poor signal-to-noise ratio is obtained. The preferred solution is to allow overflow on occasions, but prevent the limit-cycle oscillations from occurring. A solution of this type involves incorporating a saturation mechanism so as to achieve a transfer characteristic of the type shown in

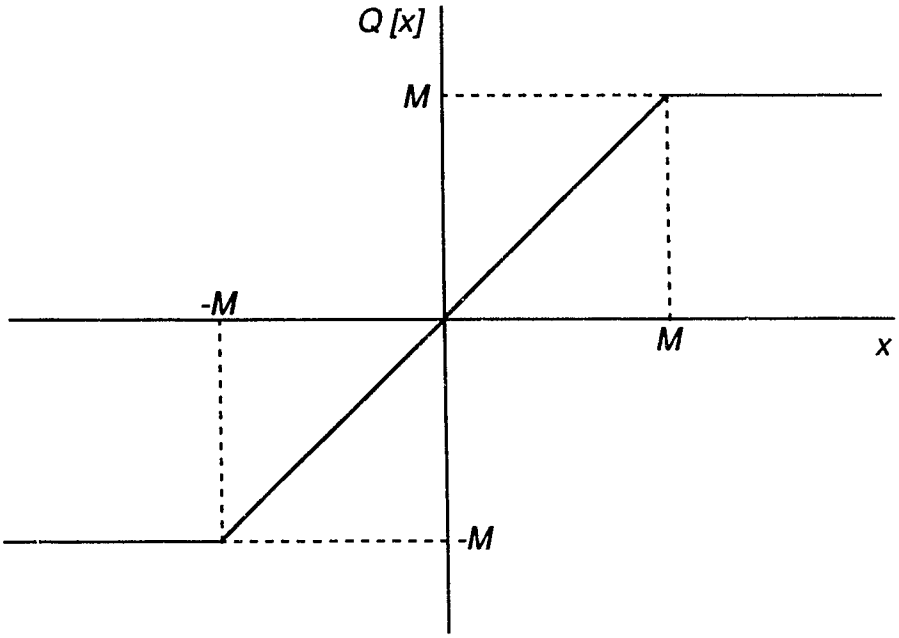


Figure 6.1: Saturation characteristics.

Fig. 6.1 [65] where

$$Q[x] = \begin{cases} x & |x| < M \\ M & |x| \geq M \end{cases} \tag{6.1}$$

$Q[x]$ is the quantized value of x and M is a positive constant.

Quantization limit-cycles can be eliminated by using appropriate signal quantization schemes. It has been shown in [66] that if a digital filter structure satisfies a condition such that the power stored in the unit delays cannot increase for zero input, then zero-input limit-cycles can be eliminated by using magnitude truncation for quantization. By power stored in the unit delays we mean squares of the values stored in the delay elements. The magnitude quantization is such that the states of the filter satisfy the inequality

$$|v_k(n)|_q \leq |v_k(n)| \tag{6.2}$$

where $|v_k(n)|_q$ is the quantized version of $|v_k(n)|$. Limit cycles can also be generated if the input assumes a constant value for a certain period of time. Limit

cycles of this type, which include zero-input limit cycles as a special case, are referred to as constant-input limit cycles. These limit-cycles can be eliminated by using the special structures mentioned in [67]. Direct-form implementation is restricted in its use in that the coefficients are located in a restricted region within the unit-circle in order to avoid limit-cycle oscillation. However, its structural regularity and simplicity makes it attractive for VLSI implementation.

6.3 Implementation of the second-order digital filter

In this section we consider the implementation of the digital-filter structure of Fig. 2.14. The second-order filter has an input-output relation given by

$$\begin{aligned} y(n) &= a_0x(n-1) + a_1x(n-2) + a_2x(n-3) \\ &\quad -b_1y(n-1) - b_2y(n-2) \end{aligned} \tag{6.3}$$

6.3.1 Features

In our design the bits of the input signal are applied in parallel and those of the output are obtained in parallel. Since our intention was to build a prototype second-order filter, we chose a word length of 12 bits, in two's complement fixed-point representation, for both the input and the filter coefficients. We have assumed the following ranges for the various signals and coefficients in the filter.

$$\begin{aligned} -1 \leq\ & x\ & \leq\ & 0.9995 \\ -1 \leq\ & y\ & \leq\ & 0.9995 \\ -2 \leq\ & a_i\ & \leq\ & 1.9995 \\ -2 \leq\ & b_i\ & \leq\ & 1.9995 \end{aligned}$$

It can be seen that the binary point for x and y is located to the right of the MSB and that for the a_i and b_i is located to the right of two MSBs.

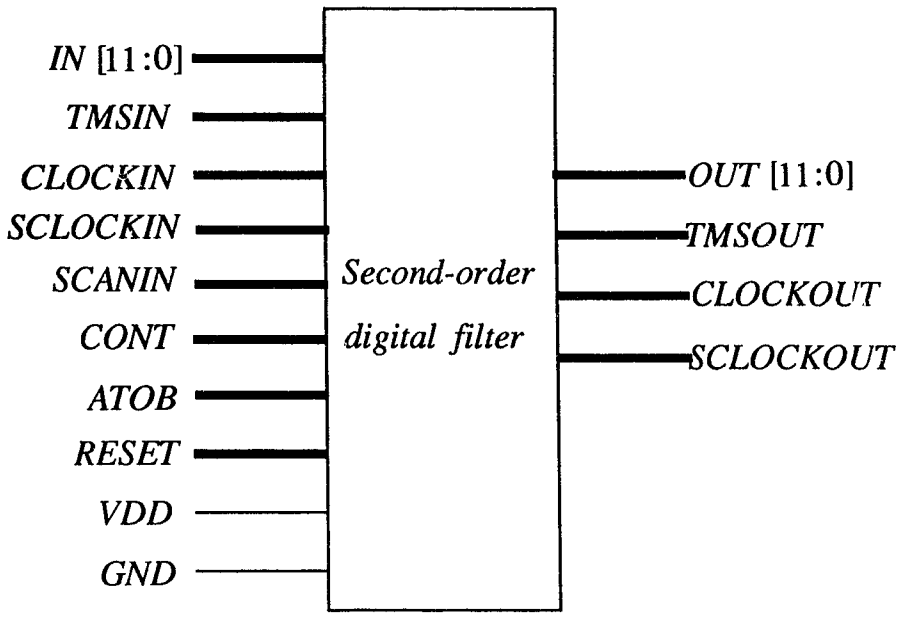


Figure 6.2: Icon for the second-order recursive digital filter.

6.3.2 Icon and block diagram of the filter

Figure 6.2 shows the icon of the digital-filter chip and Fig. 6.3 shows the block diagram. There are five registers to store the past values of input and output signals, five registers to store the five coefficients of the filter. It can also be seen that the filter has a multiplier unit along with a correction unit. The correction unit comprises a magnitude truncator, a magnitude detector and a saturation-arithmetic unit. The various signals are as follows.

IN[11:0] Input signal bus of 12 lines.

CLOCKIN Clock line for the whole digital filter.

TMSIN Test mode select pin. A 1 on this line configures all the registers in a scan-path mode and 0 will allow the normal operation of all the registers.

SCANIN This pin is used in the scan-path testing of the filter. When the TMS pin is 1, the input to the register chain are through the scanin pin. The

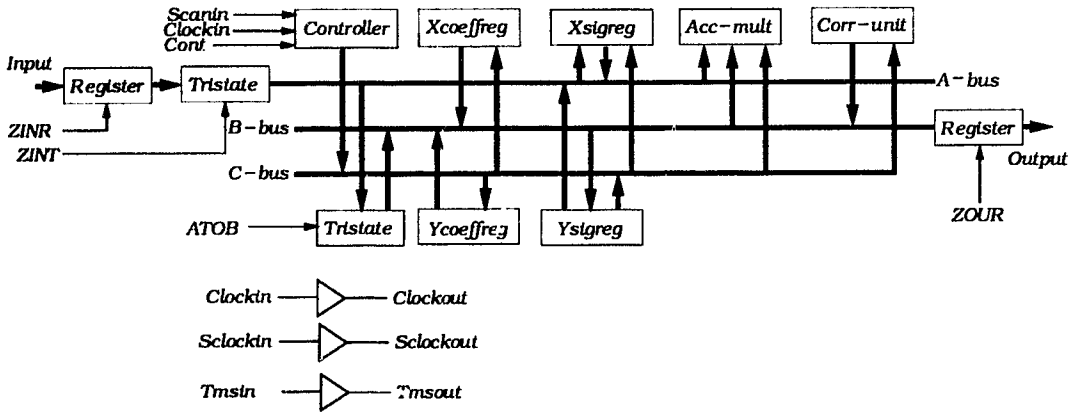


Figure 6.3: Block diagram of the second-order recursive digital filter.

coefficients of the filter are also loaded through this pin.

SCLOCKIN This is the clock line used when scan-path testing is used. The data through the register chain moves with the help of this clock. Also the coefficients of the filters is loaded into the filter with the help of this clock.

CONT This pin is used to preset a pattern in the controller such that any part of the filter can be tested.

ATOB This line is used to allow the communication between the two buses A and B. In order to test any section of the filter, the signal present in the A bus has to be output and this is done when $ATOB$ is 1.

VDD Power bus line.

GND Ground bus line.

OUT[11:0] Output signal bus of 12 lines.

CLOCKOUT This line is used to clock the next chip when a cascade of second-order filters is used.

SCLOCKOUT This clock line is used to perform the same function as the **SCLOCKIN** for the next adjacent chip.

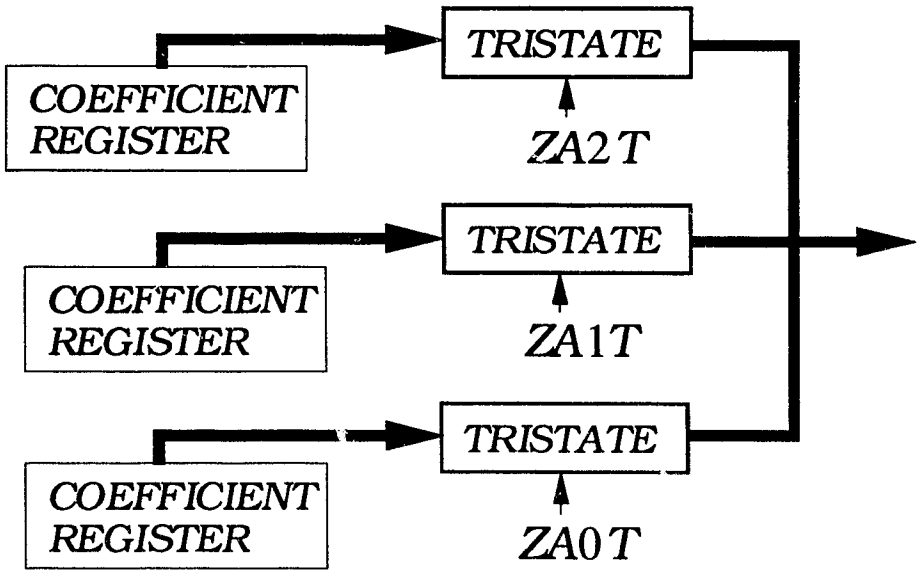


Figure 6.4: Block diagram of Xcoeffreg unit.

TMSOUT The **TMSIN** pin is used to configure the adjacent chip in the testing mode.

RESET This pins resets all the internal registers of the filter.

6.3.3 Functional description of the building blocks

Xcoeffreg Figure 6.4 shows the block diagram for the Xcoeffreg unit. There are three registers to store the a_i coefficients and the control signals $ZA0T$, $ZA1T$, and $ZA2T$ allow the coefficients a_0, a_1, and a_2, respectively, to be sent to the multiplier through the B bus.

Ycoeffreg Figure 6.5 shows the block diagram for the Ycoeffreg unit. There are two registers to store the b_i coefficients and the control signals $ZB1T$ and $ZB2T$ allow the coefficients $-b_1$, and $-b_2$, respectively, to be sent to the multiplier through the B bus.

Xsigreg Figure 6.6 shows the block diagram for the Xsigreg unit. There are three registers to store the past input signals and the control signals $ZA0T$, $ZA1T$,

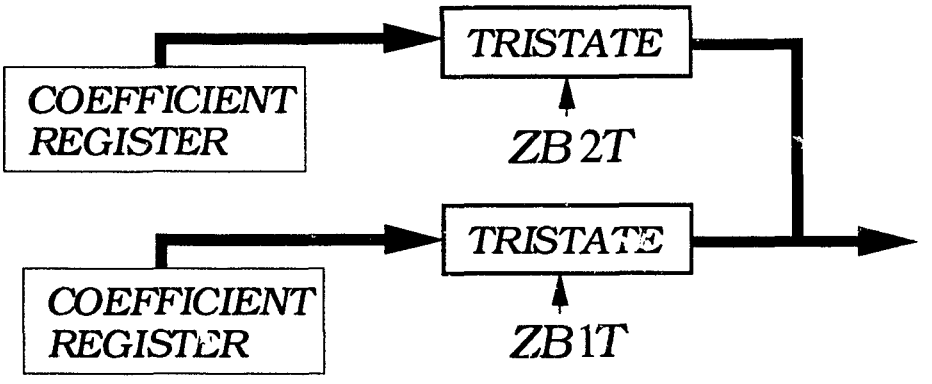


Figure 6.5: Block diagram of Ycoeffreg unit.

and $ZA2T$ allow $x(n-1)$, $x(n-2)$, and $x(n-3)$, respectively, to be sent to the multiplier through the A bus. The signal $ZXSH$ is used to shift the values of the input signals stored in the registers from one to the next to effect time shift.

Ysigreg Figure 6.7 shows the block diagram for the Ysigreg unit. There are two registers to store the past output signals and the control signals $ZB1T$ and $ZB2T$ allow $y(n-1)$ and $y(n-2)$, respectively, to be sent to the multiplier through the A bus. The signal $ZYSH$ is used to shift the values of the output signals stored in the registers from one to the next to effect time shift. This signal, however, is the same as $ZXSH$ signal.

Accumulator-multiplier As was mentioned earlier, the second-order filter entails five additions and five multiplications for each output. If the speed rather than the area is the important criterion, then each multiplication and addition is assigned its own hardware. However, if silicon area is important then, some kind of multiplexing should be employed.

To this end, we have used the ACM discussed in Chapter 4 as an iterative multiplier except that there is an overflow detection unit that detects the overflow in the final addition of the inner product. As has been mentioned earlier the ACM occupies less area and is as fast as the conventional MAC structure. The multiplication of a coefficient and a signal yields a number that has a binary point to the right of three MSBs. Since five multiplications and additions are to

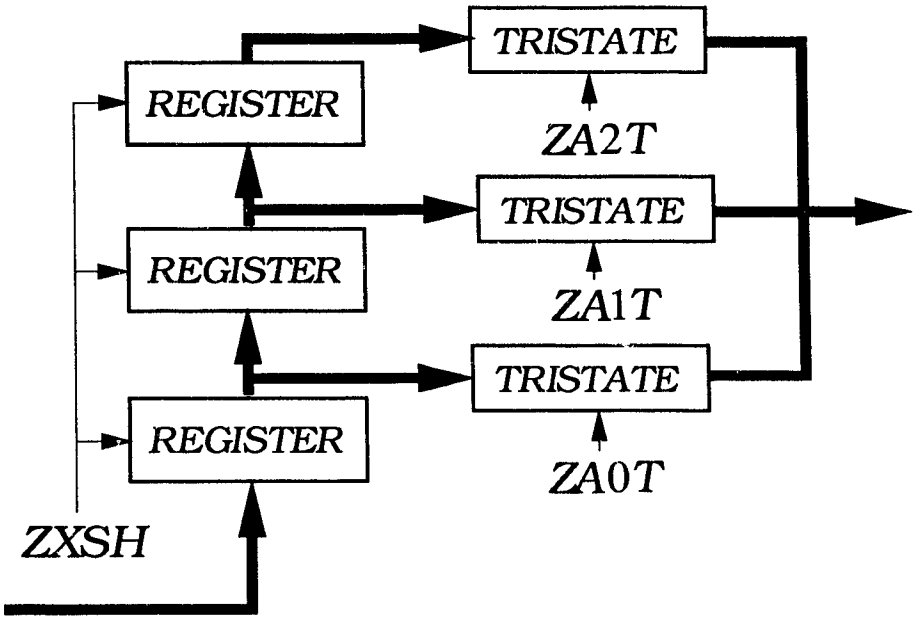


Figure 6.6: Block diagram of Xsigreg unit.

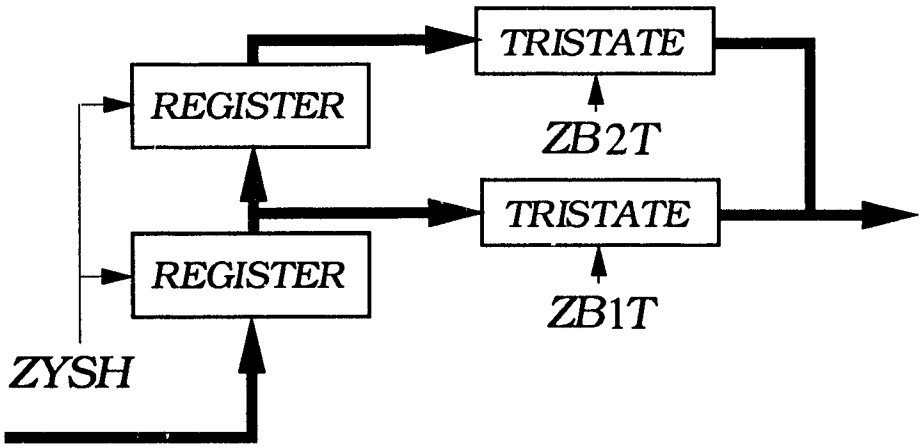
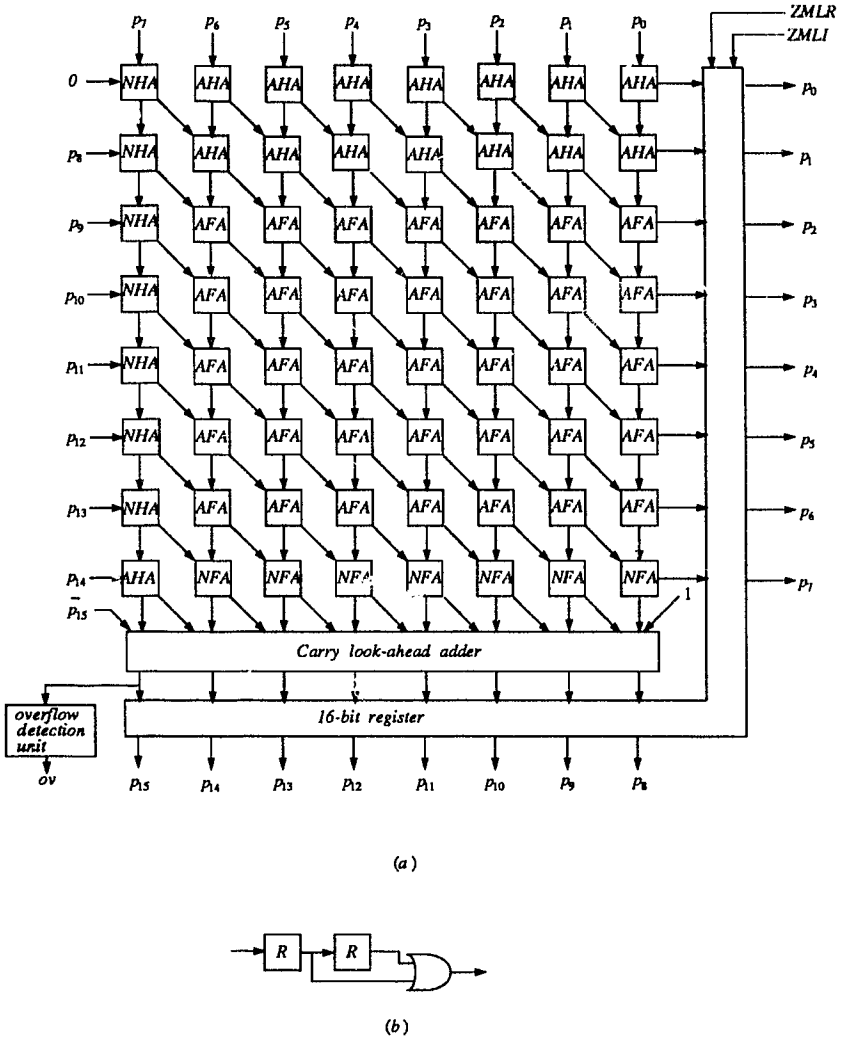


Figure 6.7: Block diagram of Ysigreg unit.

be carried out, the binary point in the final product is located to the right of four MSBs. As is well known in two's complement arithmetic, the overflow after each multiplication-addition can be neglected, except at the final multiplication-addition, provided the output is restricted to the range -1 to 0.9995. A circuit is incorporated in order to detect the overflow in the final addition. The overflow detector performs an OR operation of the MSBs of the partial products resulting from the fourth and the fifth iterations of the ACM. Thus, an overflow is indicated if the result of this OR operation is 1. Figure 6.8 shows an 8×8 two's complement iterative multiplier that incorporates this overflow detection unit. The overflow bit is used by the magnitude truncator to selectively truncate the number at its output. The control signal ZMLR resets the multiplier before any inner product calculation, while ZMLI allows the multiplier to be used iteratively. In a total of five clock cycles the multiplier effects all the operations in (6.3).

Magnitude Truncator As mentioned earlier, in order to eliminate quantization limit-cycle oscillations, the internal states (values stored in delay elements located at the output of an adder or a multiplier) of the filter are truncated using magnitude truncation. The output of the multiplier is 25 bits long including the overflow bit in which the binary point is to the right of four MSBs. The magnitude truncator truncates the output of the multiplier to 15 MSBs when the signal is positive and a one is added to the signal after truncation when the signal is negative so as to satisfy (6.2). The sign of the output of the multiplier is detected by the magnitude-decoder circuit mentioned below. The block diagram of the magnitude truncator is shown in Fig. 6.9. The XOR gates are used to produce the sum bit and the AND gate is used to generate the carry bit. The value of the output of the magnitude truncator lies between -8 and 7.9995, since the binary point lies to the right of four MSBs.

Magnitude decoder The signal from the output of the magnitude truncator is 15 bits long. Before, it is passed to the saturation-arithmetic (described below) the type of overflow, viz., positive, negative, or no overflow, has to be deter-



(a)

(b)

Figure 6.8: An 8×8 iterative multiplier that includes the overflow detection unit. (a) The multiplier unit. (b) Overflow detection unit.

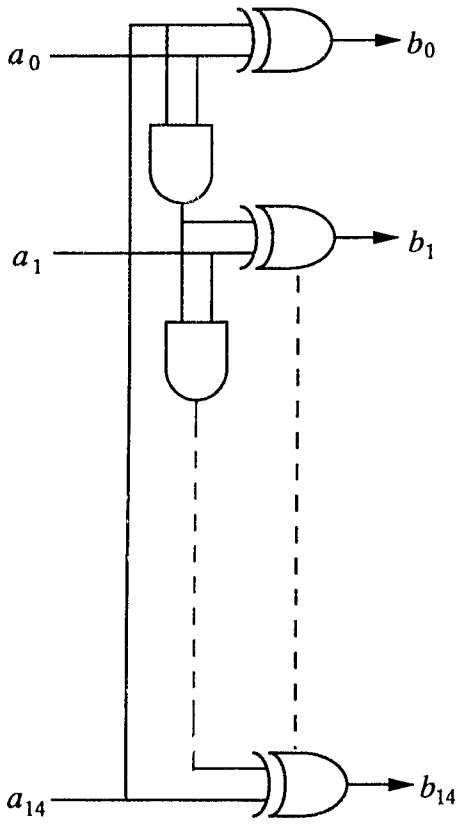


Figure 6.9: Schematic diagram of the magnitude truncator.

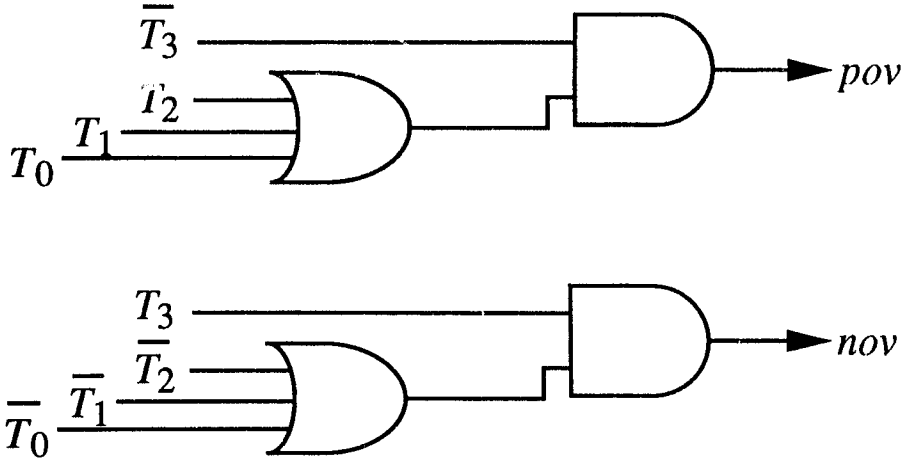


Figure 6.10: Block diagram of the magnitude decoder.

mined. Depending on the type of overflow that has occurred in the final product of the multiplier, the magnitude-decoder issues control signals to the saturation-arithmetic unit to perform a positive saturation, a negative saturation or allow the signal to be passed without any modification. Figure 6.10 shows the block diagram of the magnitude decoder. In this figure *pov* corresponds to a positive overflow and *nov* corresponds to a negative overflow. $T_3 - T_0$ correspond to the 4 MSBs of the output of the magnitude truncator.

Saturation-arithmetic unit In order to avoid overflow limit-cycle oscillations, a saturation mechanism is incorporated such that the final output before being stored in any storage unit has value between -1 and 0.9995. The magnitude decoder detects the type of overflow and outputs two signals for the two types of overflow, as mentioned above. Figure 6.11 shows the block diagram of the saturation-arithmetic unit. In this figure, if T_0 (which is connected to *pov* of the magnitude decoder) = 1, the saturation-arithmetic unit outputs 0.9995 and if T_1 (which is connected to *nov* of the magnitude decoder) = 1, the saturation-arithmetic unit outputs -1. If both T_0 and T_1 are equal to zero, then no overflow has occurred and the input to the saturation-arithmetic unit is passed to the output without any modification. When $ZOUT$ = 1, the output of the saturation-

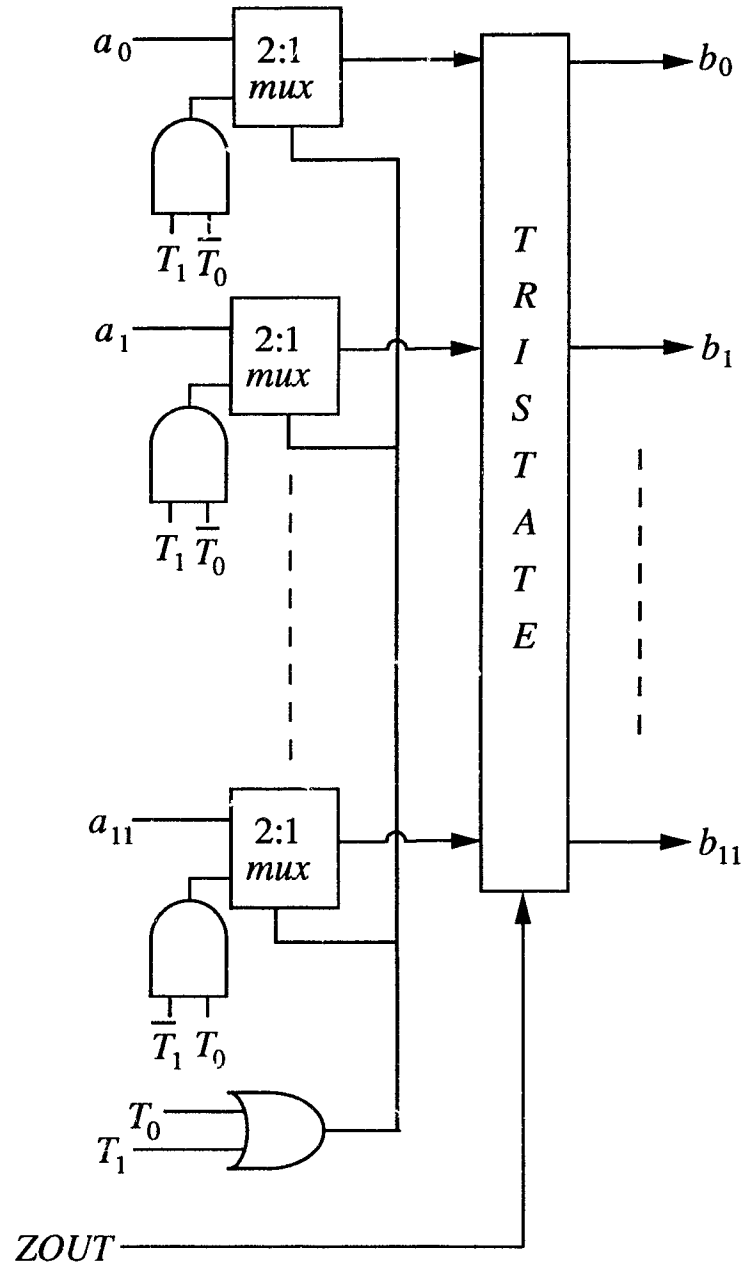


Figure 6.11: Block diagram of the saturation-arithmetic unit.

ZINR	Enable the input register
ZINT	Enable the input tristate buffer
ZXSH	Shift the past input signal values
ZMLR	Reset the multiplier
ZA0T	Load a_0 and $x(n-1)$ to the bus
ZA1T	Load a_1 and $x(n-2)$ to the bus
ZA2T	Load a_2 and $x(n-3)$ to the bus
ZB1T	Load b_1 and $y(n-1)$ to the bus
ZB2T	Load b_2 and $y(n-2)$ to the bus
ZMLI	Enable the multiplier to iterate
ZOUT	Enable the output tristate buffer
ZOUR	Enable the output register

Table 6.1: Control signals for the second-order digital filter.

arithmetic unit is sent to the B bus.

Controller The controller coordinates the operation of the filter module. In other words the controller enables each block within the system to operate at a specified time. There are several ways to implement any controller viz., using programmable logic arrays (PLAs), microprogram sequencers, one-flip-flop-per-state method etc. In the second-order filter, 12 control signals are generated. We have used the one-flip-flop-per-state method to implement a controller. Figure 6.12 shows the block diagram of the controller while Table 6.1 describes the various control signals in the digital filter.

6.3.4 Timing diagram

Figure 6.13 shows the timing of the various signals in the second-order filter. The input signal is present as long as $ZINT$ is high and the output of the filter is produced when $ZOUR$ goes low.

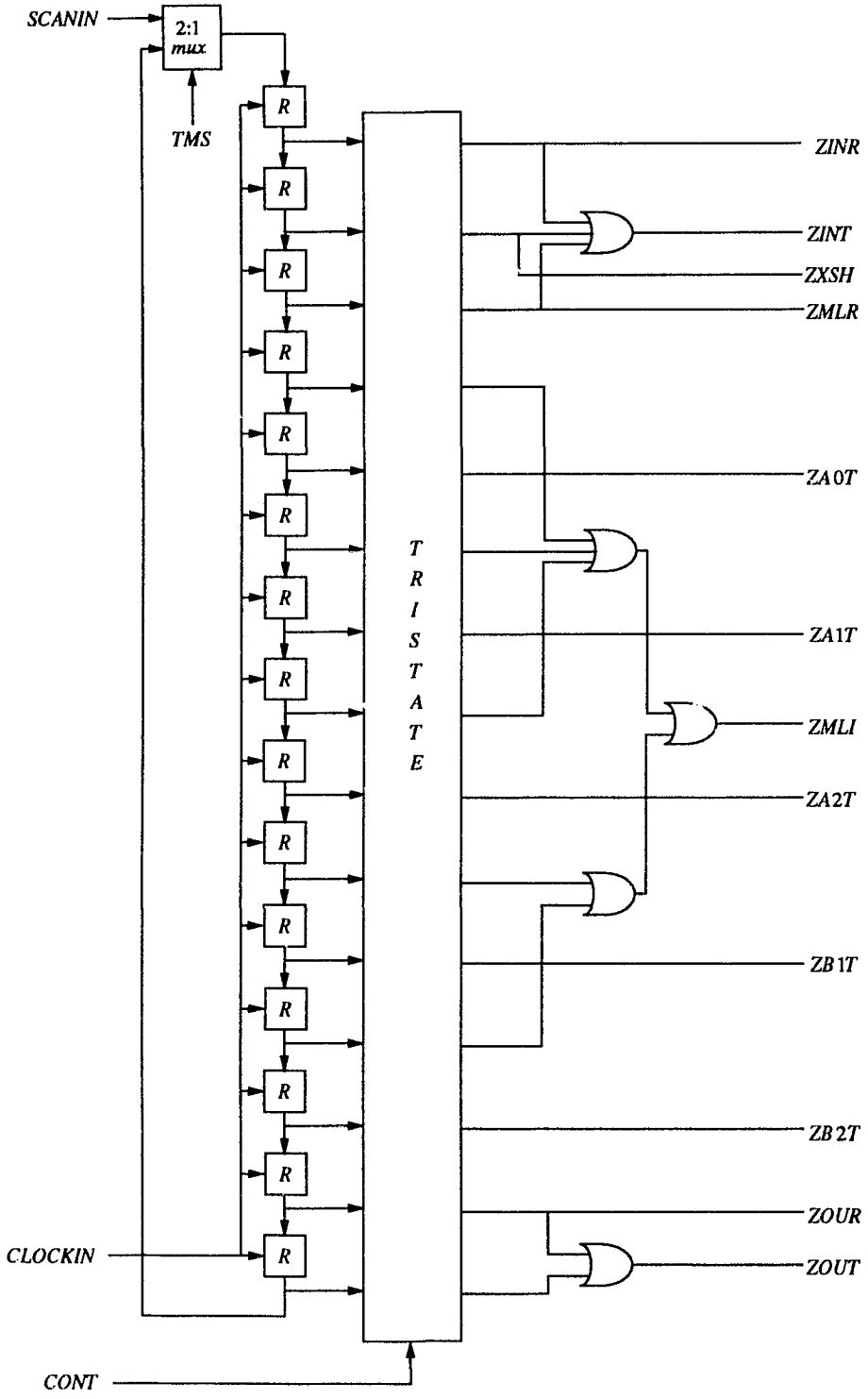


Figure 6.12: Block diagram of the controller.

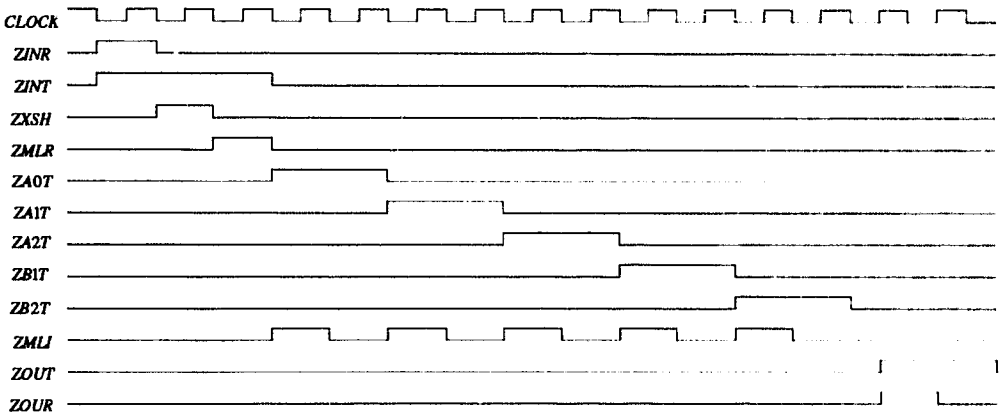


Figure 6.13: Timing diagram for the second-order filter.

Second-order digital filter	
Technology	1.2μ NTE CMOS4S
Area of the chip	5312.27x5312.27 μm^2
Area of the core	4460.67x4460.67 μm^2
No. of pins	40
No. of transistors	14288
Maximum frequency	3.87 MHz
Area $\times$ time	7.2745e+09 $\mu m^2 ns$

Table 6.2: Chip statistics of the second-order digital filter.

6.3.5 Physical characteristics

The detailed statistics of the chip are given in Tables 6.2 and 6.3. The maximum sampling frequency of the filter was found to be approximately 4 MHz.

6.3.6 Testing strategy

To test the second-order filter, the scan-path technique has been used. All the registers used in the design are of the type shown in Fig. 5.12. Under testing conditions, the TMS pin is raised high thus allowing the signal to pass through the scan-in pin to the circuit. In this design, scan-path is also used to load the coefficients and 75 clock cycles are required to load the five filter coefficient

Second-order digital filter	
Gate	Gate count
Inverter	238
Buffer	388
Tri-Buffer	171
D-Flipflop	171
Nand2	616
Nand3	17
Nand4	6
Nor2	33
Nor3	8
Nor4	3
Gate count	1847

Table 6.3: Gate-level statistics of the second-order digital filter.

registers and the control signals into the controller. The CONT pin is used to partition the circuit by presetting a pattern in the controller counter. When CONT is 1 the connection between the controller and the rest of the circuit is tristated and after a selected pattern is placed in the controller counter, the CONT is made 0. Sometimes it may be necessary to output the signal present on the A bus and this is accomplished by raising the ATOB signal which establishes a connection between the A and B buses. In this way every part of the circuit can be individually tested. In addition, the filter can be initialized to any known state. It must be mentioned here that due to the unavailability of an automatic test pattern generator, a program was written to generate approximately 500 random test verification vectors for the filter circuit. Simulation results for these vectors have shown the correctness of operation of the circuit.

6.3.7 Layout of the second-order digital filter

A photomicrograph of the second-order digital filter is shown in Fig. 6.14.

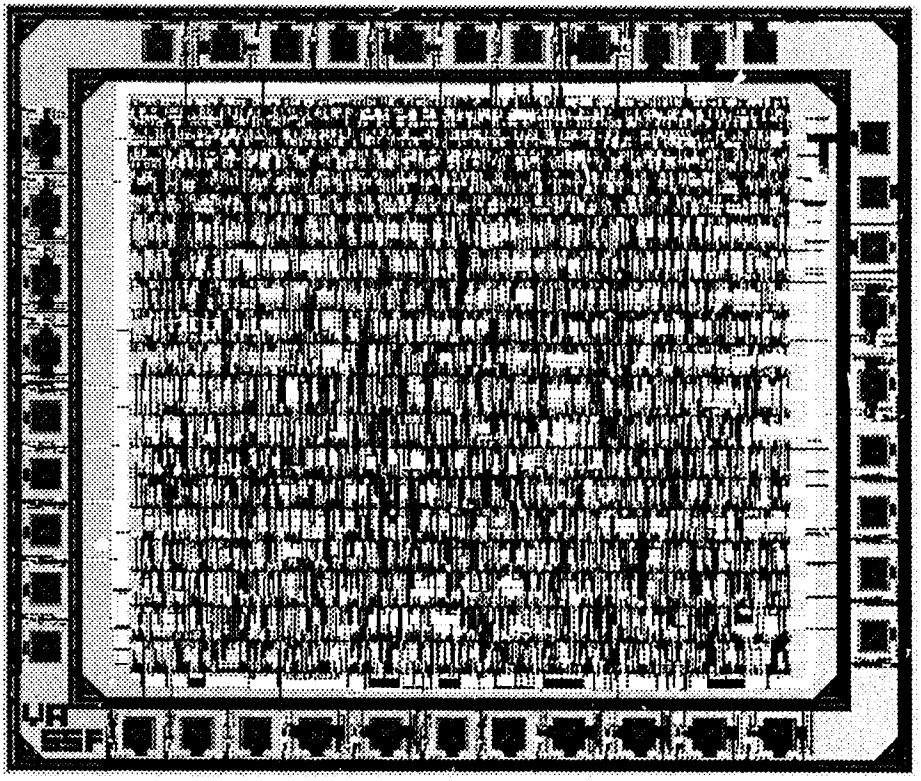


Figure 6.14: Photomicrograph of the second-order digital filter.

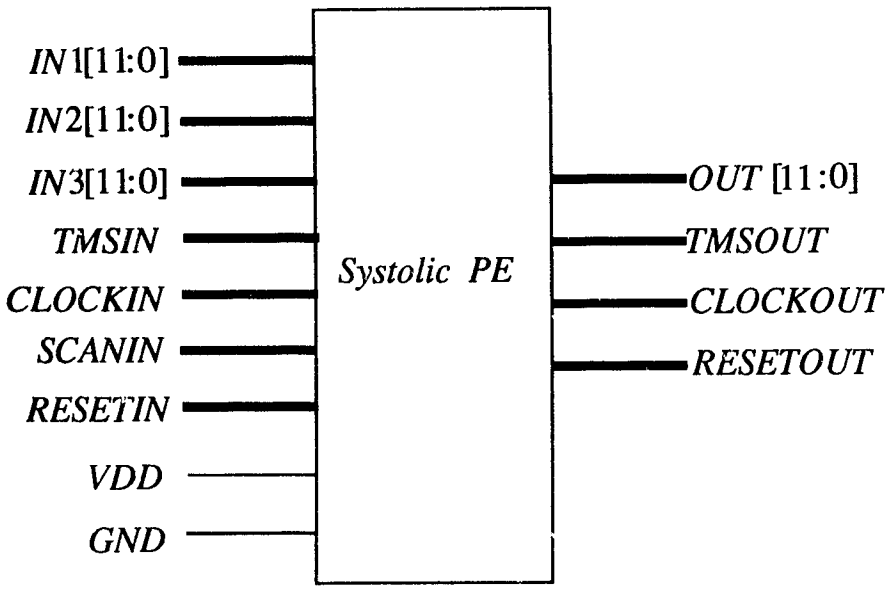


Figure 6.15: Icon for the systolic PE.

6.4 Implementation of a systolic PE

6.4.1 Features

A single PE of Fig. 2.11(b) has been implemented. This PE can be used as a basic unit for 1-D and M-D digital filters. The components of this PE are two multipliers, two adders, a correction unit and a set of registers to store the coefficients and the output. The correction unit comprises, as was stated before, the magnitude truncator, the magnitude decoder, and the saturation-arithmetic unit. By using the ACM unit, we can reduce the number of adders to one.

6.4.2 Icon and block diagram of the systolic PE

An icon for the PE is shown in Fig. 6.15 and Fig. 6.16 shows the block diagram. The various signals are as follows.

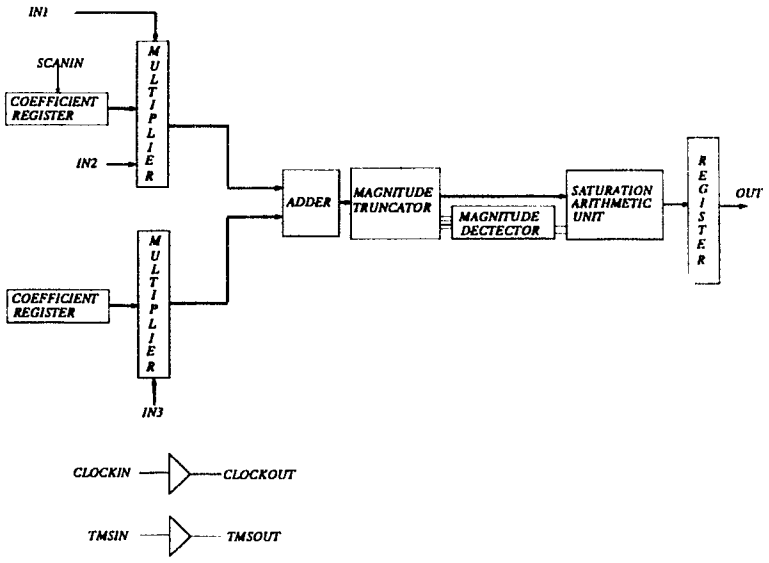


Figure 6.16: Block diagram of the systolic PE.

IN1[11:0] Input signal bus of 12 lines.

IN2[11:0] Partial product signal bus from the adjacent PE of 12 lines.

IN3[11:0] Output signal bus 12 lines from the output of the filter.

CLOCKIN Clock line for the whole PE.

TMSIN Test mode select pin. A 1 on this line configures all the registers in a scan-path mode and 0 will allow the normal operation of all the registers.

SCANIN This pin is used in the scan-path testing of the filter. When the TMS pin is 1, the input to the register chain is through the SCANIN pin.

CLOCKOUT This line is used to clock the next chip when a cascade of PE is used.

TMSOUT The **TMSIN** pin is used to configure the adjacent chip in the testing mode.

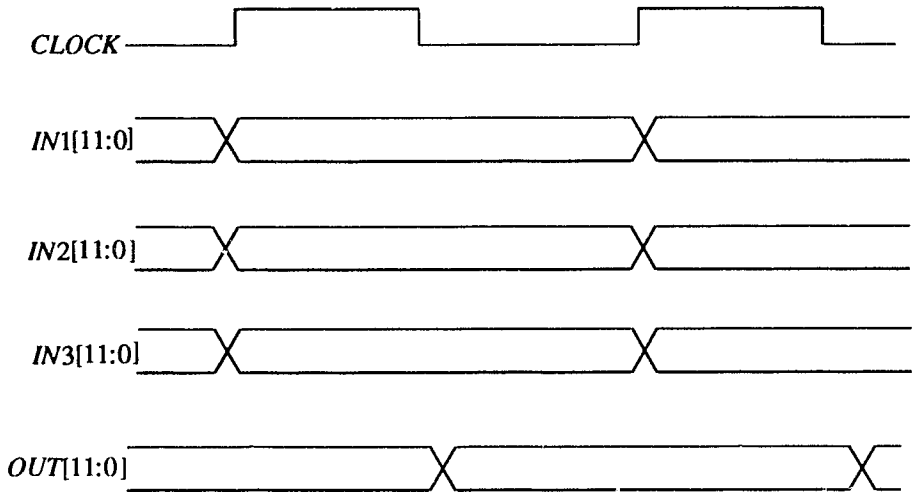


Figure 6.17: Timing diagram for the systolic PE.

RESETIN This pin resets all the internal registers of the PE.

RESETOUT This signal is the same as **RESETIN** and is sent to the adjacent PE.

VDD Power bus line.

GND Ground bus line.

6.4.3 Functional description of the block diagram

The systolic PE basically comprises the input and output registers, two coefficient registers, two ACM units, an adder, and a correction unit. The functions of each of these units have been mentioned earlier. It must be mentioned that since the output of each PE is stored in a register, we need to use one correction unit in each PE to eliminate limit-cycle oscillations.

6.4.4 Timing diagram

Figure 6.17 shows the timing of the various signals in the systolic PE. Since the PE is systolic, output is obtained at every clock cycle.

Processing element	
Technology	1.2μ NTE CMOS4S
Area of the chip	5387.07x5387.07 $\mu^2 m$
Area of the core	4144.87x4144.87 $\mu^2 m$
No. of pins	63
No. of transistors	10466
Maximum frequency	12.31 MHz
Area $\times$ time	2.3572e+09 ns$\mu^2 m$

Table 6.4: Chip statistics of the systolic PE.

Processing element	
Gate	Gate count
Inverter	487
Buffer	350
D-Flipflop	84
Nand2	1282
Nand3	62
Nand4	24
Nor2	73
Nor3	24
Nor4	12
Gate count	2398

Table 6.5: Gate-level statistics of the systolic PE.

6.4.5 Physical characteristics

This PE has 40 input pins and 15 output pins. A 68-pin PGA pad frame was used for the chip. As many as four pairs of power pads were incorporated in the design to account for metal migration and noise spikes. Of these, two pairs of power pads were used for the core area and two pairs for the I/O pads. Maximum sampling frequency of the PE was found to be approximately 12 MHz. The detailed statistics of the chip are given in Tables 6.4 and 6.5.

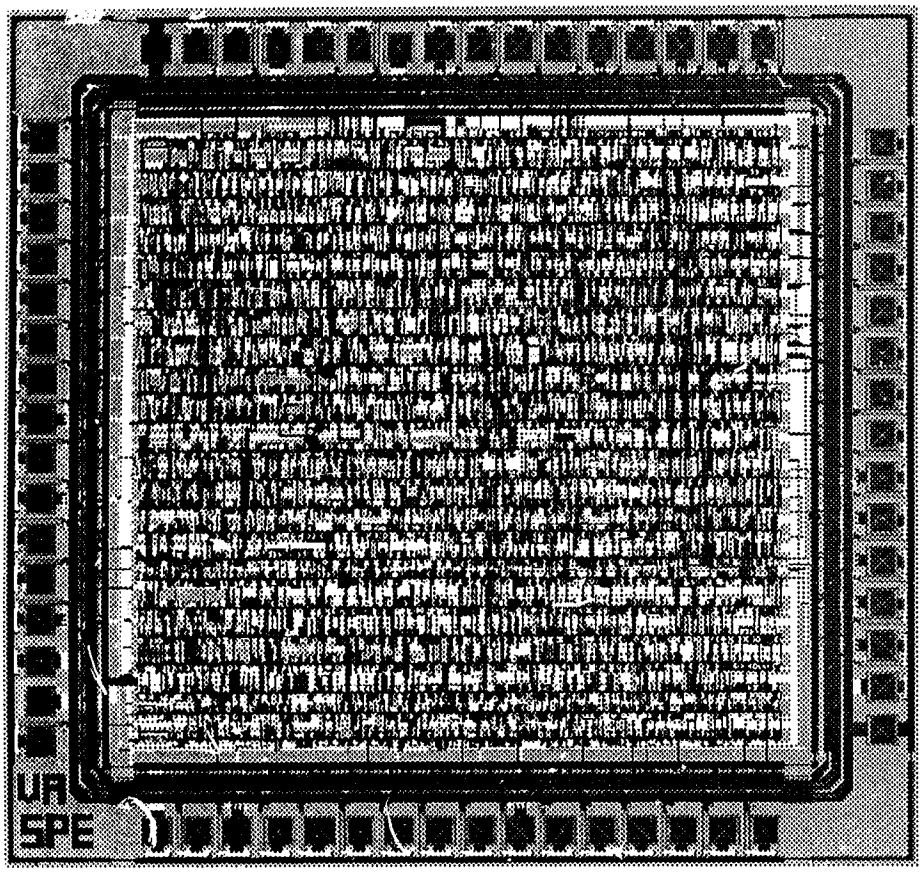


Figure 6.18: Photomicrograph of the systolic PE.

6.4.6 Testing strategy

For the purpose of testability, the scan-path technique has been used. For this design 24 clock cycles are required to load the coefficients, since there are two coefficient registers, each 12 bits in length.

6.4.7 Layout of the systolic PE

A photomicrograph of the systolic PE is shown in Fig. 6.18.

6.5 Comparisons

For brevity, we shall call the second-order filter of Fig. 2.14 as the single second-order filter (SSF) and that of Fig. 2.15 as cascaded second-order filter (CSF). It can be seen from Tables 6.2 and 6.4 that the area occupied by a single PE is about the same as that of the SSF. It may be noted that three PEs are required to build the CSF. In the CSF, there are three sites where product truncation is performed. On the other hand, for the SSF, the product truncation is performed only at one site and full precision is maintained in all the multiplication and addition operations. Consequently, the roundoff noise generated in the CSF is about three times that generated in the SSF. Moreover, even though the sampling frequency of the SSF is lower than that of the CSF, the area $\times$ time complexity of both SSF and CSF are approximately the same. In other words, for the same area $\times$ time complexity, the SSF has better performance than the CSF in terms of roundoff noise.

6.6 Conclusions

In this chapter, we have discussed the VLSI implementation of an SSF and a single systolic PE mentioned in Chapter 2. In these structures units to circumvent the quantization and overflow limit-cycle oscillations have been incorporated. The two chips have been implemented in 1.2μ CMOS4S technology. The VLSI statistics of the chips have been estimated using the tools available in Cadence. It has also been found that for the same area $\times$ time complexity of both SSF and CSF, the SSF has a better performance than the CSF in terms of roundoff errors.

Chapter 7

Conclusions

The objective of this thesis has been to develop methods for mapping 1-D and M-D digital-filter algorithms onto systolic arrays that are modular and hierarchical using the z-domain approach, to obtain new multiplier structures that can be used in the implementation of digital filters, and to implement multiplier and digital-filter structures using VLSI techniques.

In this chapter the contributions described in this thesis are summarized and recommendations for future research are presented.

7.1 Contribution

In Chapter 2, we have discussed two approaches for the mapping of the filter algorithms onto systolic arrays. One is based on the SFG and the other is based on the z-domain representation of the filter algorithm. In the SFG method, the filter algorithms are converted into a single-assignment code form and are then mapped onto an index space. Different scheduling and projection vectors have been determined and used to obtain several systolic arrays. It has been shown that the number of structures that can be obtained by employing this method is limited because of the restricted number of feasible transformations. As the dimension of the filter increases, it becomes difficult to obtain appropriate projection and scheduling vectors for the SFG. In the z-domain method, the filter equations are transformed into their z-domain representations and recursive expressions similar

to single-assignment codes are derived using Horner's rule or other polynomial evaluation techniques. By obtaining different recursive expressions, different systolic structures have been derived. Since no matrix transformation is involved in this method, the question of determinig the scheduling and projection vectors does not arise.

In Chapter 3, the z-domain approach has been extended to include the derivation of systolic structures for M-D digital filters that are hierarchical and modular at every level of the hierarchy. It has been shown that the number of possible structures increases with the dimension of the filter.

Several multipliers that can be used in digital filters have been described in Chapter 4. A truncated multiplier that occupies approximately 50% of the area of a conventional parallel multiplier has been designed. It has also been shown by way of an example that for approximately the same variation of the output noise PSD in a digital filter, the truncated multiplier occupies less area and is as fast as the parallel multiplier. A pipelined version of the truncated multiplier for both unsigned and two's complement multiplications have been advanced. Quasi-serial multipliers for both unsigned and two's complement multiplications have been discussed. These multipliers accept input in parallel and produce output in a bit-serial fashion. Multipliers based on the modified Booth algorithm have been proposed. In this algorithm four-bit segments of the multiplier are scanned and encoded to perform appropriate operations on the multiplicand. As an illustration, an 8×8 OMB multiplier has been designed and compared with the conventional QMB multiplier. It has been shown that the area $\times$ time complexity of the OMB multiplier is less than that of the QMB multiplier. A diminished-1 multiplier that can be applied in the implementation of the Fermat NTT has also been described. This multiplier circumvents the need to translate a number from its diminished-1 form to its binary form. This feature leads to reduced chip area and latency. An ACM has been designed that occupies less area and is as fast as the conventional MAC.

In Chapter 5, a 16×16 two's complement truncated multiplier, an 8×8 OMB multiplier, and a diminished-1 multiplier have been implemented in 1.2μ CMOS4S technology and simulated using SILOS. Comparisons of the truncated multiplier with the full multiplier and those of the OMB multiplier with the QMB multiplier have been made in terms of some VLSI parameters.

A second-order digital filter described in Chapter 2 has been implemented using in 1.2μ CMOS4S technology in Chapter 6. An ACM multiplier in conjunction with an overflow detection unit has been used as an iterative multiplier to perform the inner product operation involved and to reduce the silicon area. Hardware units to circumvent both quantization and overflow limit cycles have been incorporated.

A systolic PE that can be used as a basic unit in the design of 1-D and M-D digital filters has also been implemented. It has been shown that the SSF has a lower area $\times$ time complexity than the CSF. In addition, it has been shown that the signal-to-noise ratio in the SSF is about a third of that in the CSF. Comparisons of the two filter structures with respect to other VLSI characteristics have also been carried out.

7.2 Recommendations for further research

The z-domain approach has been exclusively used to derive digital-filter structures since direct-form structures are easily derived from the filter equations. It would be interesting to see if the z-domain method or some delay-operator method can be used to map matrix problems like the SVD, QR decomposition or the solution of linear equations onto systolic arrays.

In all the mapping techniques discussed in the literature so far and in the z-domain approach advanced here, very little has been said about systematic control strategies for the systolic arrays. The control strategies used in all the systolic structures derived in this thesis have been heuristic and structure dependent. It would be worthwhile devising a systematic methodology by which systolic struc-

tures as well as control strategies can be derived.

There are many digital filters that can be derived using the wave network characterization [33]. One such configuration is derived by using the wave characterization in conjunction with the GIC and is, consequently, called the GIC-based wave configuration [68]. These filters are known to exhibit a desirable feature of low sensitivity to coefficient variations and in certain circumstances low roundoff noise. Unfortunately individual second-order sections are not regular and cannot be designed as a cascade of simple PEs. It may be possible to regularize these structures by introducing redundant multipliers or delays and obtain modular structures.

In Chapter 4, we have designed a truncated multiplier for multiplying two real numbers. A full multiplier which can multiply two complex numbers can be designed based on the algorithm forwarded in [53]. Instead of carrying out four multiplications and three additions a parallel complex multiplier could be designed by treating each bit as a complex bit of the form $a + jb$, where $a, b \in \{0, 1\}$. Then, from this design a truncated multiplier could be designed that multiplies two complex numbers.

It would be interesting to compare the VLSI characteristics of larger sizes of the QMB and OMB multipliers. For instance, a thorough analysis in terms of speed, area, and area $\times$ time complexity of 16×16 QMB and OMB multipliers could be carried out. This would give insight into the use of OMB multipliers in real-time signal processing requiring high processing rates.

It would also be interesting to investigate Booth-like algorithms for the multiplication of complex numbers. Such multipliers find applications in seismic and radar signal processing.

In Chapter 6, a single systolic PE has been implemented. Using this PE as a basic unit, a VLSI chip for a 2-D digital filter using one of the structures described in Chapter 3 can be designed that can be used in high-speed video processing.

Bibliography

[1] K. Hwang and A. Briggs, *Computer Architecture and Parallel Processing*, McGraw Hill, 1987.

[2] M. J. Flynn, "Very high speed computing system," *Proc. IEEE*, vol. 54, pp. 1901-1909, 1966.

[3] H.T. Kung, "Why systolic architectures?," *IEEE Computer*, vol. 25, pp. 37-46, Jan. 1982.

[4] S.Y. Kung, *VLSI Array Processors*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[5] S.K. Rao, *Regular Iterative Algorithms and their Implementations on Processor Arrays*, Ph.D. thesis, Stanford University, Stanford, California, 1985.

[6] D. I. Moldovan, "On the analysis and synthesis of VLSI algorithms," *IEEE Trans. Comput.*, vol. C-31, pp. 1121-1126, Nov. 1982.

[7] D. I. Moldovan, "On the design of algorithms for VLSI algorithms," *Proc. IEEE*, vol. 71, Jan. 1983.

[8] W. L. Miranker and A. Winkler, "Space-time representations of computational structures," *Computing*, vol. 32, pp. 93-114, 1984.

[9] P. R. Capello and K. Steiglitz, "Unifying VLSI array designs with geometric transformations," *Proc. International Conference on Parallel Processing*, pp. 448-457, 1983.

[10] P. Quinton, "The systematic design of systolic arrays," *IRISA Internal Publication*, 193, Apr. 1983.

[11] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Co., pp. 271-292, 1980.

[12] M.A. Sid-Ahmed, "A systolic realization of 2-D filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-37, pp. 560-565, Apr. 1989.

[13] P.S. Liu and T.Y. Young, "VLSI array architecture for picture processing," in *Picture Engineering*, K.S. Fu and T. Kunii, Eds., Springer Verlag, New York, pp. 171-186, 1982.

[14] P.S. Liu and T.Y. Young, "VLSI array design under constraint of limited I/O bandwidth," *IEEE Trans. Comput.*, vol. C-32, pp. 1160-1170, Dec. 1983.

[15] M. S. Lam and J. Mostow, "A transformational model of VLSI systolic design," *IEEE Computer*, vol. 18, pp. 42-52, Feb. 1985.

[16] H. T. Kung and W. T. Lin, "An algebra for VLSI algorithm design," *Proc. Conf. Elliptic Problem Solvers*, Monterey, CA, 1983.

[17] D. Cohen and V. Tyree, "VLSI system for synthetic aperture radar (SAR) processing," *SPIE*, vol. 186, pp. 166-177, May 1979.

[18] M. Hatamian and G. L. Cash, "A 70-MHz 8×8-bit parallel pipelined multiplier in 2.5 μm CMOS," *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 505-513, Aug. 1986.

[19] A. R. Cooper, "Parallel architecture modified Booth multiplier," *IEE Proc.*, vol. 135, pt. G, pp. 125-128, June 1988.

[20] M. Benaissa, A. Bouridane, S. S. Dlay, and A. G. J. Holt, "Diminished-1 multiplier for a fast convolver and correlator using the Fermat number transform," *IEE Proc.*, vol. 135, pt. G, pp. 187-193, Oct. 1988.

[21] T. G. Noll, D. Schmitt-Landsiedel, and G. Enders, "A pipelined 330-MHz multiplier," *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 411-416, June 1986.

[22] D. A. Henlin, M. T. Fertsch, M. Mazin, and E. D. Lewis, "A 16 bit × 16 bit pipelined multiplier macrocell," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 542-547, Apr. 1985.

[23] C. P. Lerouge, P. Girard, and J. S. Colardelle, "A fast 16 bit NMOS parallel multiplier," *IEEE J. Solid-State Circuits*, vol. SC-19, pp. 338-1984, June 1984.

[24] D. G. Crawley and G. A. J. Amaratunga, "8 ×8 bit pipelined Dadda multiplier in CMOS," *IEE Proc.*, vol. 135, pt. G, pp. 231-240, Dec. 1988.

[25] S. Nakamura and K.-Y. Chu, "A single chip parallel multiplier by MOS technology," *IEEE Trans. Comput.*, vol. C-37, pp. 274-282, Mar. 1988.

[26] P. Denyer and D. Renshaw, *VLSI Signal Processing: A Bit-Serial Approach*, Addison-Wesley Publishing Co., Reading, Mass., 1985.

[27] P. R. Capello and K. Steiglitz, "A Note on free accumulation in VLSI filter architectures," *IEEE Trans. Circuits Syst.*, vol. CAS-32, pp. 291-296, Mar. 1985.

[28] P. R. Capello and C. W. Wu, "Computer aided design of VLSI FIR filters," *Proc. IEEE*, vol. 75, pp. 1260-1271, Sept. 1987.

[29] L. B. Jackson, J. F. Kaiser, and H. S. McDonald, "An approach to implementation of digital filters," *IEEE Trans. on Audio Electroacoust.*, vol. AU-16, pp. 413-421, Sept. 1968.

[30] S. L. Freeny, "Special purpose hardware for digital filtering," *Proc. IEEE*, vol. 63, pp. 633-648, Apr. 1975.

[31] J. K. J. Van Gingerdeuren, H. J. De Man, N. F. Gonçalves, and W. A. M. Van Noije, "Compact NMOS building blocks and a methodology for dedicated digital filter applications," *IEEE J. Solid-State Circuits*, vol. SC-13, pp. 565-572, Oct. 1978.

[32] K. K. Parhi and M. Hatamian, "A high sample rate recursive digital filter chip," in *VLSI Signal Processing III*, R.W. Brodersen and H. S. Moscovitz, Eds., IEEE Press, New York, 1988.

[33] A. Fettweis, "Digital filter structures related to classical filter networks," *Arch. Elektron. Übertrag.*, vol. 25, pp. 79-89, 1971.

[34] M. Renfors, B. Sikström, U. Sjöström, and L. Wanhammar, "VLSI-implementation of a digital PCM filter, "*Proc. Int. Conf. on Computers, Systems, and Signal Processing*, Bangalore, India, pp. 1468-1472, Dec. 1984.

[35] N. Petrie, J. Mavor, and S. G. Smith, "General-purpose adaptor structure for wave-digital filter realizations," *Electron. Lett.*, vol. 19, pp. 1038-1039, Nov. 1983.

[36] T. Noll and W. Ulbrich, "Digital filter with parallel arithmetic for custom designs," *Proc. 6th Eur. Conf. on Circuit Theory and Design*, Stuttgart, W. Germany, pp. 281-283, Sept. 1983.

[37] A. Antoniou, *Digital Filters: Analysis and Design*, McGraw-Hill, New York, 1979.

[38] B. C. McKinney and F. El-Guibaly, "A multiple-access pipeline architecture for digital signal processing," *IEEE Tran. Comput.*, vol. 37, pp. 283-290, Mar. 1988.

[39] F. El-Guibaly, S. Sunder, and A. Antoniou, "Systolic Implementation of FIR filters," in *Signal Processing V: Theories and Applications*, L. Torres, E.

Masgrau, and M. A. Lagunas, Eds., Elsevier Science Publishers, pp. 1423-1426, 1990.

[40] H. K. Kwan, "New systolic array for realising second-order recursive digital filters," *Electronic Letters*, vol. 23, No. 9, pp. 442-443, Apr. 1987.

[41] S. Sunder, P. S. R. Diniz, F. El-Guibaly, and A. Antoniou, "Synthesis of systolic structures for second-order recursive digital filters," *IEEE Pacific Rim Conf. Comm., Comp., Signal Processing*, Victoria, B. C., pp. 514-517, May 1991.

[42] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[43] H. K. Kwan and T. S. Okullo-Oballa, "Systolic array for implementation of a decimator and interpolator," *IEE Proc.*, vol. 135, pt. E, pp. 70-72, Jan. 1988.

[44] H. T. Kung, L. M. Ruane, and D. W. L. Yen, "A two-level pipelined systolic array for convolutions," in *VLSI Systems and Computations*, H. T. Kung, B. Sproull, and G. Steele, Eds., Computer Science Press, Maryland, pp. 255-264, 1981.

[45] R. Gnanasekaran, "2-D filter implementations for real-time signal processing," *IEEE Trans. Circuits Syst.*, vol. CAS-35, pp. 587-590, May 1988.

[46] T. Venkateswarlu, C. Eswaran, and A. Antoniou, "Realization of multidimensional digital transfer functions," *Multidimensional Systems and Signal Processing*, vol. 1, pp. 179-198, 1990.

[47] C. Eswaran, T. Venkateswarlu, and A. Antoniou, "Realization of multidimensional GIC digital filters," *IEEE Trans. Circuits Syst.*, vol. CAS-37, pp. 685-694, June 1990.

[48] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1984.

[49] S. Sunder, F. El-Guibaly, and A. Antoniou, "Systolic implementation of two-dimensional recursive digital filters," *Proc. IEEE Sym. Circuits and Syst.*, New Orleans, LA, pp. 1034-1037, May 1990.

[50] D. Raghuramireddy and R. Unbehauen, "A systolic structure for complex digital filters," *Proc. IEEE Symp. Circuits Syst.*, pp. 1252-1255, May 1990.

[51] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, CBS College Publishing, 1982.

[52] J. J. F. Cavanagh, *Digital Computer Arithmetic*, McGraw-Hill, 1984.

[53] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Comput.*, vol. C-22, pp. 1045-1047, Dec. 1973.

[54] E. E. Swartzlander, Jr., "The quasi-serial multiplier," *IEEE Trans. Comput.*, vol. C-22, pp. 317-321, Apr. 1973.

[55] T. G. McDonald and R. K. Guha, "The two's complement quasi-serial multiplier," *IEEE Trans. Comput.*, vol. C-24, pp. 1233-1235, Dec. 1975.

[56] O. L. MacSorley, "High-speed arithmetic in binary computers," *Proc. IRE*, vol. 49, pp. 67-91, Jan. 1961.

[57] C. S. Wallace, "A suggestion for parallel multipliers," *IEEE Trans. Electron. Comput.*, vol EC-13, pp. 14-17, Feb. 1964.

[58] L. Dadda, "Some schemes for parallel multipliers," *Alta Freq.*, vol. 34, pp. 349-356, 1965.

[59] C. M. Rader, "Discrete convolutions vis Mersenne transforms," *IEEE Trans. Comput.*, vol. C-21, pp. 1269-1273, Dec. 1972.

[60] R. C. Agarwal and C. S. Burrus, "Fast convolution using the Fermat number transforms with application to digital filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 87-99, Apr. 1974.

[61] L. M. Leibowitz, "A simplified binary arithmetic for the Fermat number transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 356-359, Oct. 1976.

[62] J. H. McClellan and C. M. Rader, *Number Theory in Digital Signal Processing*, Prentice Hall, Engelwood Cliffs, NJ, 1979.

[63] R. A. Roberts and C. T. Mullis, *Digital Signal Processing*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1987.

[64] L. B. Jackson, *Digital filters and signal processing*, Kluwer Academic Publishers, Norwell, MA, 1986.

[65] P. M. Ebert, J. E. Mazo, and M. G. Taylor, "Overflow oscillations in digital filters," *Bell Syst. Tech. J.*, vol. 48, pp. 2999-3020, Nov. 1969.

[66] K. Meerkötter, "Realization of limit cycle-free second-order digital filters," *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 295-298, 1976.

[67] P. S. R. Diniz and A. Antoniou, "More economical state-space digital-filter structures which are free of constant-input limit cycles," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 807-815 Aug. 1986.

[68] A. Antoniou and M. G. Rezk, "Digital-filter synthesis using concept of generalized-immittance converter," *IEE J. Electron. Circuits Syst.*, vol. 1, pp. 207-216, Nov. 1977 (see vol. 2, pp. 88, May 1978 for errata).