## NOTICE

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

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

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

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

## AVIS

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

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

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

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

# VLSI Architectures for Real-Time Recursive State-Space Filtering

By

Jin Yun Zhang

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

May 1991

ISBN 0-315-70477-2

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

# Abstract

The increasing demands of speed and performance become evident with the expanding utilization of signal/image processing to industrial, medical, and military environments. As the numerical properties are considered, state-space structures have been successfully exploited in realizing high performance fixed-point digital filters. However, they require more computations than other realizations. In this thesis, a general design methodology for mapping recursive algorithms onto optimal VLSI array processors is first given. Then, the parallel computational models for recursive state-space filtering are developed, and various high speed VLSI array architectures are obtained based on the design methodology. For 1-D IIR filters, the state variable representation is used to obtain a simple systolic array. Based on a block processing technique, an advanced state update algorithm is developed and a corresponding high speed array processor results for state-space filtering. By using the decomposition method, a cascade form of second-order state-space structures is suggested, which gives a high throughput rate and high efficiency with hardware less than for the $N$th order form. In the 2-D case, besides the advanced state update array (called local speedup), the inherent spatial concurrency in 2-D systems is explored and a global speedup architecture is presented. This architecture consists of a number of column array processors and it works on multiple columns in parallel. It is very flexible and modular. The throughput rate is adjustable and therefore can be very high. In order to utilize the advantages of state-space filtering, the adaptive state-space filtering is developed based on the LMS algorithm. Gradients

are derived directly from the state update equation and the observation equation. The stability monitoring, convergence rate and roundoff noise performance of the adaptive filter are studied. A VLSI architecture is proposed for speeding up the filtering and the adaptation. Finally, a very fast Kalman filter for image restoration is investigated. By using Roesser's LSS structure to represent the image generation model and the degradation model, a simple composite dynamic system representation is obtained. The fast Kalman filter is established by defining a proper state vector and using diagonal scanning method. A dedicated VLSI array architecture is designed for real-time image restoration. Proposed algorithms and architectures are verified by computer simulations.

# Acknowledgement

I wish to express my most sincere gratitude to my supervisor Dr. Willem Steenaart for his academic guidance and spiritual support during the course of the program.

The help and advice of all members of my supervisory committee, Dr. E. Petriu, Dr. D. Gibbons and Dr. C. H. Chan, are greatly appreciated.

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

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

Many thanks to all professors, colleagues and friends at the Department of Electrical Engineering, University of Ottawa, for their help and encouragement during the period of my studies.

Finally, I would like to express my deep appreciation to my family. Without their understanding and support, it is impossible for me to finish my study successfully.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Real-Time Digital Signal/Image Processing

Digital signal and image processing has become an important and very useful tool in a wide variety of fields, such as biomedical engineering, acoustics, sonar, radar, seismology, speech communication, data communication, nuclear science, weather broadcasting, astronomy, computer vision, medical image processing and many others. Simply stated, a signal is any medium for conveying information, and digital signal processing is concerned with the representation of information by sequences of numbers or symbols and the processing of these sequences. In many applications, we want to extract some characteristic parameters. Alternatively, we may wish to remove interference, such as noise, from the signal or to modify the signal to present it in a form which is more easily interpreted [1]. In general, the mathematical and algorithmic techniques encompassed in digital signal and image processing are diverse. However, most signal and image processing algorithms are dominated by convolution/correlation filtering, transform techniques and some linear algebraic methods, such as, matrix algebra method [2].

The dominating aspects in signal and image processing requirements are mathematically intensive algorithms and often real-time operation. To illustrate these

characteristics, we will use the digital filter as an example. Specifically, we will use the Finite Impulse Response (FIR) filter which in the time domain takes the general form of

$$y(n) = \Sigma_{i=1}^{N} a(i)x(n-i) \tag{1.1}$$

where $y(n)$ is the output sample at time $n$, $a(i)$ is the $i$th coefficient or weighting factor, and $x(n-i)$ is the $(n-i)$th input sample. From (1.1), to generate every $y(n)$, we have to compute $N$ multiplications and additions or sums of products. This computation is mathematically intensive, especially when $N$ is large. In addition to being mathematically intensive, digital signal processing (DSP) algorithms must be performed in real-time. Real-time can be defined as a process that is accomplished by the DSP at the sampling rate. In the FIR filter example in (1.1), the sum of the products must be computed usually within hundreds of microseconds before the next sample comes into the system. In a speech recognition system, a noticeable delay between a word being spoken and being recognized would be unacceptable and not considered real-time. In image processing, it is considered real-time if the processor finishes the processing within the frame update period [3]. In the area of image processing, there has been a tremendous growth in both numbers of images and bit rates. For example, in remote sensing, Landsats 1, 2 and 3 create $1.5 \times 10^{13}$ bits per year. Landsat D, with higher resolution images, create approximately $3.7 \times 10^{15}$ bits per year [4]. The data rate of Landsat 6 is 85 megabits per second. And for Landsat 7 will create even more data per second [5]. In biomedicine, 15 million images were processed in the United States alone in 1982. one million x-ray images (excluding those in the dental area) were processed in the United States in 1980. Those figures constantly increase as the emphasis shifts from diagnostic medicine to preventive health care. In broadcast television, the color video signal is sampled at 10.7 Mhz and each sample is quantized to 8 bits (256 levels), resulting in a bit rate of $85.6 Mb/s$ [5]. The bit rate of the future wall screen television will be up to 1 billion bits per second [6]. These examples indicate the need for fast processing

of these images.

In order to be able to process images and signals at an appropriate rate for obtaining desired information in real-time, a very large number of operations have to be performed at a very high speed. A computation rate in excess of 1000 million operations per second may frequently become necessary for real-time performance. Some typical algorithm examples encountered in the image processing area are illustrated in Table 1.1 [2].

| *Processing function* | *Necessary throughput* |
|---|---|
| *Linear operations, O(N)*<br>*-spatial filtering*<br>*-convolution*<br>*-edge detection* | $10^2 - 10^5$ *MOPs* |
| *Second-order operations, O($N^2$)*<br>*-sorting operations*<br>*-median filtering*<br>*-nearest-neighbor classification* | $10^3 - 10^7$ MOPs |
| *High order operations*<br>*-matrix based*<br>*-spectral processing*<br>*-adaptive operations* | $10^4 - 10^8$ *MOPs* |

Table 1.1: Throughput requirements for DSP algorithms.

Here it is assumed that an image quality equivalent to that of television, with a spatial resolution equivalent to $512 \times 512$ pixels and a frame rate of 30 frames a second, resulting in a data rate of $10^7$ samples per second. Hence, even for linear operations such as spatial filtering, convolution and edge detection, the throughput rate requirement will range from $10^2$ to $10^5$ MOPs (Million operations/second). For this reason conventional computer architectures which operate in a sequential manner are input-output bound when performing such tasks, and cannot usually reach such speeds.

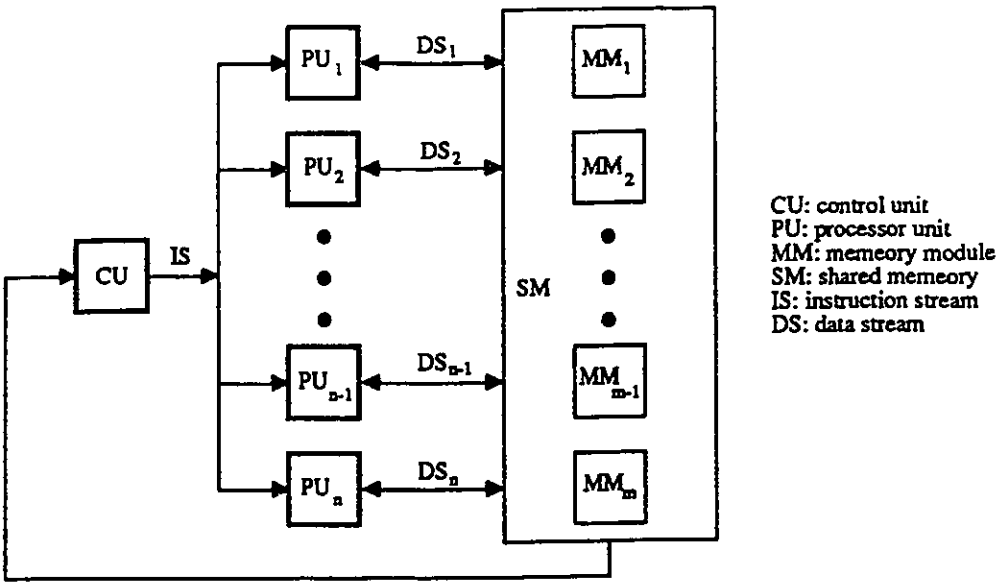
## 1.2 Parallel Computing Architectures

Over the past decades, digital signal processing machines have passed through several evolutions in order to realize real-time processing. Current parallel computers can be divided into three structural classes: pipeline computers, multiprocessor systems, and array processors [7]. A pipeline computer performs overlapped computations to exploit temporal parallelism. A multiprocessor system achieves asynchronous parallelism through a set of interactive processors with shared resources (memories, database, etc.). An array processor uses multiple synchronized logical units to achieve spatial parallelism. The first two classes belong to the general-purpose computer domain. The third class belongs to the domain of special-purpose computers [2]. Due to the complicated system organization and severe system overheads, the general-purpose computers are not suitable for real-time signal processing where a very high throughput rate is absolutely essential . On the other hand, the special-purpose array processors offer a promising solution to meet real-time processing requirements and have structural properties that are suitable for very-large-scale integration (VLSI) implementation. Therefore, we shall focus on VLSI array processors in this thesis.

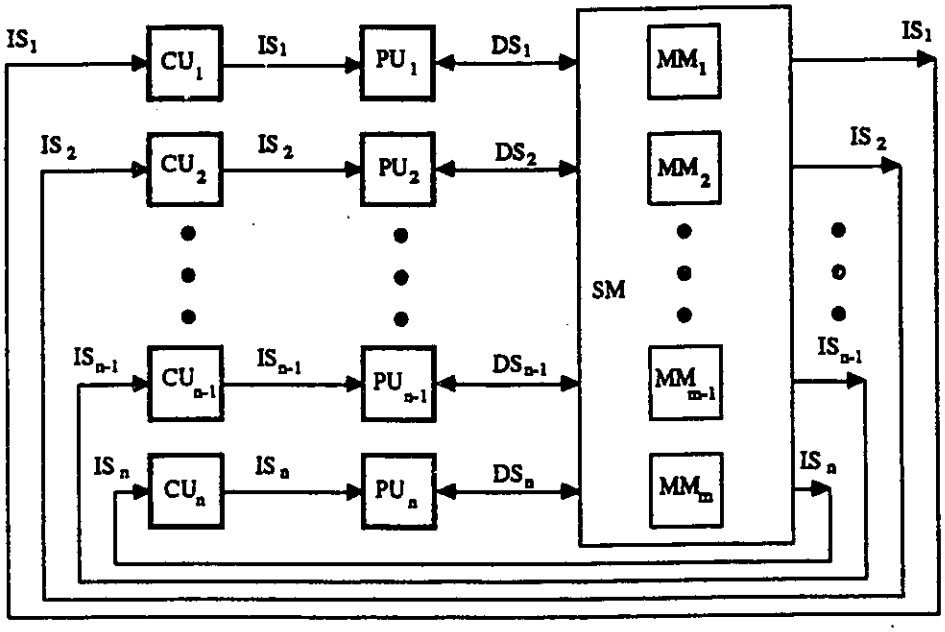
VLSI technology has made great progress in the development of fast operation and low power consumption switching elements and has vastly increased the circuit density. In addition, high density also leads to high performance and high reliability [8,9]. As indicated by Mead [8], VLSI electronics presents a challenge not only to those involved in the development of fabrication technology, but also to computer scientists and computer architects. It created a new horizon in architecture by implementing parallel algorithms directly on hardware. The current major types of VLSI parallel architectures are SIMD (single instruction and multiple data streams), MIMD (multiple instruction and multiple data streams), and systolic and wavefront arrays [10]. A SIMD array shown in Figure 1.1(a) [7] is a synchronous array of processing elements (PEs) under the supervision of one control unit. All

PEs receive the same instruction broadcast from the control unit but operate on different data sets from distinct data streams. Broadcasting of data is allowed and often needed. A MIMD machine consists of a number of PEs, each with its own control unit, program, and data (Figure 1.1(b)) [7]. The overall processing task may be distributed among the PEs for the purpose of increasing processing parallelism. A systolic array is a network of processors which rhythmically compute and pass data through the system. The systolic array features the important properties of modularity, regularity, local interconnection, a high degree of pipelining, and a highly synchronized multiprocessing. Figure 1.2 shows various systolic array configurations which are suitable for VLSI implementation [11]. The wavefront array incorporates the systolic principle with the dataflow computing concept. It is a self-timed, data-driven array. There is no global timing reference in the wavefront array (Figure 1.3) [2].

The major factors to be considered when adopting systolic arrays for special-purpose processing architectures are: simplicity and regularity of design, concurrency and local communication, and balancing computation with I/O [12]. Since 1978, when H.T. Kung and C.E. Leiserson [13] introduced the term "systolic array" and the concept behind the term, in the area of signal and image processing, much research has been done and much has been written about the design of algorithms and architectures suitable for such structures. In [14], H.T. Kung gives a partial list of problems for which systolic solutions exist. The VLSI implementation of signal and image processing algorithms have been discussed. [15] describes a linear systolic array capable of evaluating a large class of inner-product functions used in signal and image processing. These include matrix multiplication, multidimensional convolutions, as well as various nonlinear functions of vectors. [16,17,18,19,20] give the designs of systolic arrays for filtering, 1-D convolution, 2-D convolution, multidimensional convolution and resampling. VLSI structures for the computation of

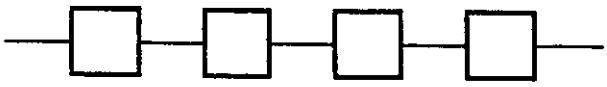



(a) SIMD computer architecture

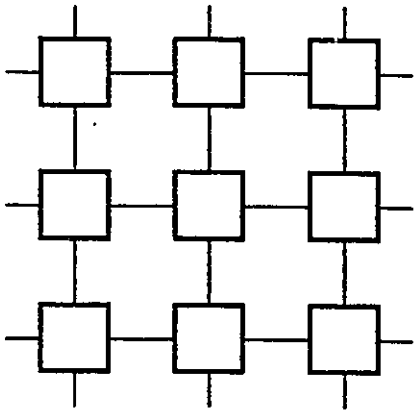



(b) MIMD computer architecture

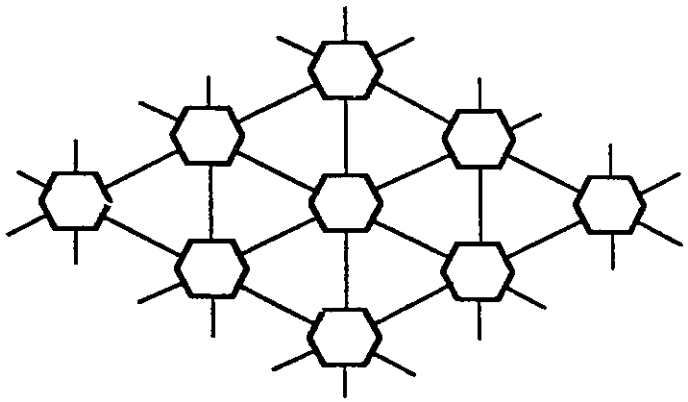
Figure 1.1: SIMD and MIMD array architectures.



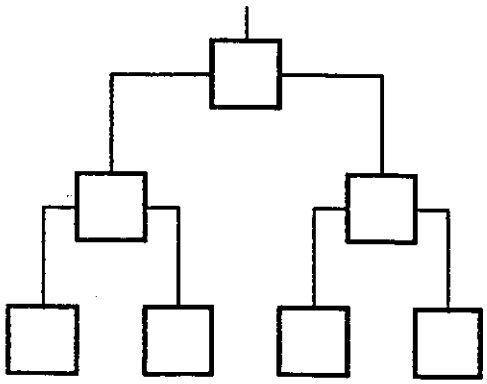
(a) One-dimensional linear array



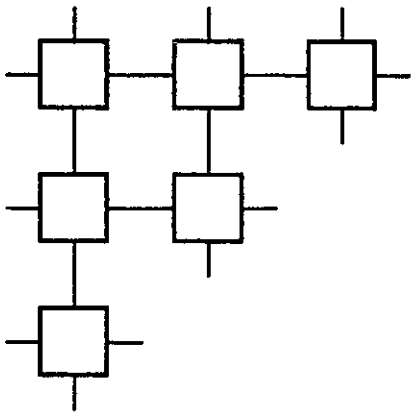
(b) Two-dimensional square array



(c) Two-dimensional hexagonal array



(d) Binary tree



(e) Triangular array

Figure 1.2: Various systolic array configurations.

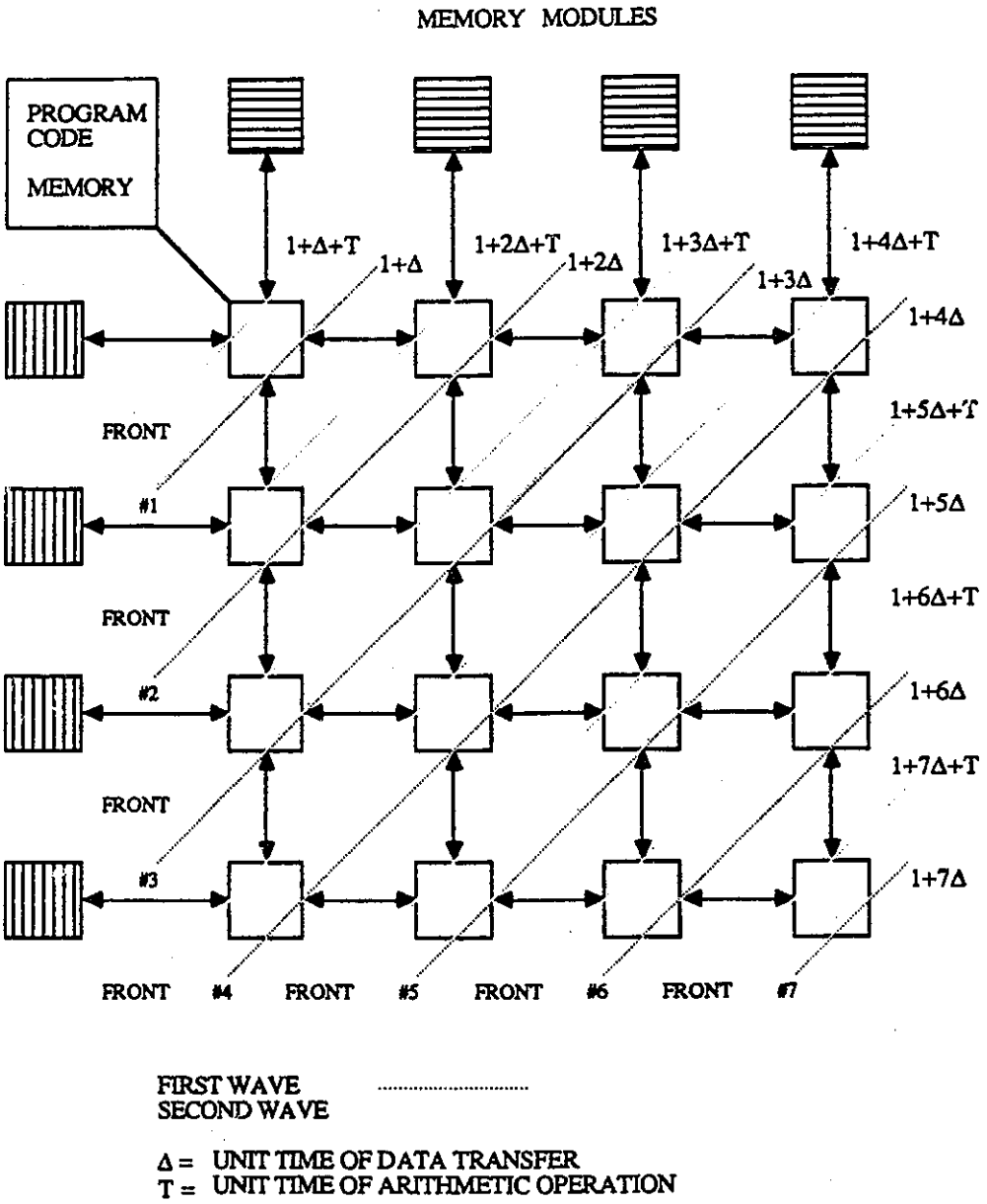



Figure 1.3: Wavefront array architecture.

the discrete Fourier transform (DFT) and fast Fourier transform (FFT) are presented in [21]-[22]. In [23], several methods for computing the FFT in hardware are reviewed. [24] reviews issues in the design of special-purpose VLSI chips in general, and suggests VLSI designs for polynomial multiplication and division. A new class of partitioned matrix algorithms is developed in [25]. Four matrix operations, L-U decomposition of a dense matrix, inversion of a triangular matrix, matrix-matrix multiplication, and solving a triangular systems of equations, are shown systematically partitionable into submatrix operations, which are feasible for direct VLSI implementation. [26] proposes a solution for mapping an arbitrary large QR algorithm into small VLSI array processors. [27] describes a matrix multiplication algorithm on a linear array of processing elements. [28] presents two systolic architectures for performing the product-sum computation $AB + C$ in the finite field $GF(2^m)$ of $2^m$ elements, where $A, B$, and $C$ are arbitrary elements of $GF(2^m)$. Recently, VLSI systolic arrays for more specific applications are proposed. In [29], a high throughput systolic implementation of the second-order recursive filter is given. In [2], S.Y. Kung presented various systolic arrays for spectral estimation, beamforming, Kalman filtering, speech processing, image processing and image analysis. Systolic arrays for adaptive null steering beamforming, linear algebraic and cellular operations, and mapping data graphs are shown in [30]. It gives discussion of the realization of systolic arrays from concept to implementation with the signal/image processing applications.

## 1.3 VLSI Design

### 1.3.1 Design Principles

VLSI architectures should maximize the processing concurrency by pipelining and parallel processing and also take into account the cost and the number of I/O pins. Consequently, VLSI architecture design principles should include modularity,

regularity, local communication, massive parallelism, and minimized I/O [12,2].

1. Regularity and Modularity

   In designing special-purpose systems, cost-effectiveness has always been a chief concern and main cost is the design cost. Therefore, the overall architecture should be as regular and modular as possible, so that design error, time and cost can be reduced. If a structure can be decomposed into a few types of simple substructures or building blocks, which are used repetitively with simple interfaces, great savings can be achieved. In addition, special-purpose systems based on simple, regular designs are likely to be modular and therefore adjustable to various performance requirements.

2. Pipelining and Parallelism

   Throughput rate is the overriding factor dictating the system performance. And, the technological trend indicates any major improvement in computation speed must come from the concurrent use of many processing elements. In order to optimize throughput rate, real-time signal processing requires extensive concurrency by using pipeline and parallel processing.

   For signal processing arrays, pipelining and parallelism at all levels should be pursued. Pipelining may bring about an extra order of magnitude in performance with very little additional hardware. Although most of the current array processors stress only word-level pipelining, the trend is to exploit the potential of multiple-level pipelining (i.e., combined pipelining in the bit-level, word-level, and array level).

3. Local versus Global Communication

   When a large number of processing elements work simultaneously, coordination and communication become significant–especially with VLSI technology. Since communication is very expensive in terms of area, power, and time consumption, it has to be restricted to localized interconnections. To avoid global

interconnections, local and regular data movements are strongly preferred.

On the other hand, the broadcasting technique is probably one of the most obvious ways to make multiple use of each input elements. There are many cases where arrays with global data communication are suggested [12]. For example, in real-time recursive filtering, the current output value depends upon the previous output. The broadcasting technique can be used to exploit the concurrency in algorithms so that the throughput rate can meet the real-time requirement. Obviously, whenever broadcasting is used, the skew has to be considered and the circuit must be designed carefully. There is a tradeoff between the throughput rate and the cost.

4. I/O Constraints

There are two I/O constraints in designing VLSI special-purpose architectures. The first one is I/O bandwidth. Since a special-purpose system typically receives data and outputs results through an attached host, I/O considerations influence overall performance. The ultimate performance goal of a special-purpose system is a computation rate that balances available I/O bandwidth with the host. Since an accurate priori estimate of available I/O bandwidth in a complex system is usually impossible, the design of a special-purpose system should be modular so that its structure can be easily adjusted to match a variety of I/O bandwidth. The second one is the limited number of I/O pins. A good design should take into account the constraints on I/O pins. Most recursive signal processing algorithms permit locality features, i.e., local data communication and distributed control, which can be fully exploited in array architectures.

### 1.3.2 Design Methodology

In general, the design of VLSI array processor for certain application, can be divided into five steps as shown in Figure 1.4. In this thesis, we will focus on mapping of DSP algorithms onto array architecture, i.e., translation from algorithmic level to architecture level.

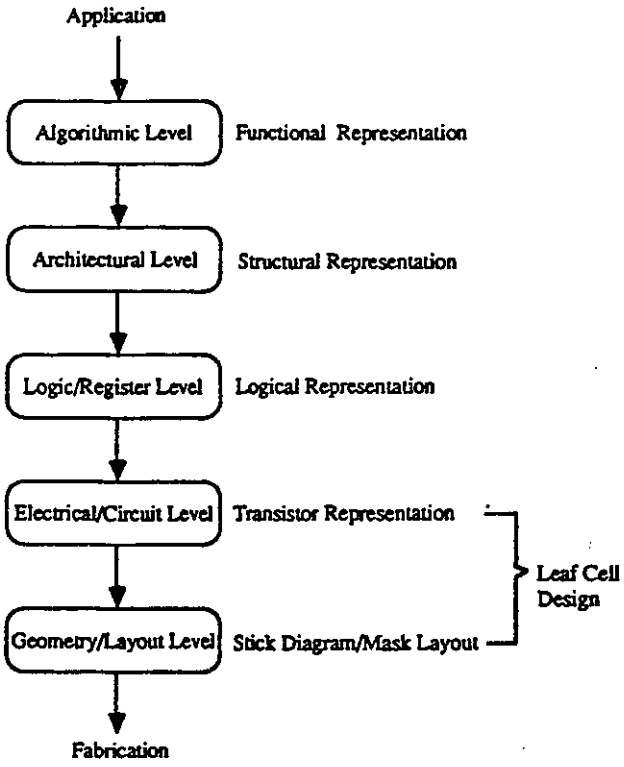



Figure 1.4: Hierarchical design levels of VLSI architectures.

Mapping of DSP algorithms onto array architectures has attracted much attention of many researchers. In the last decade, there has been a dramatic growth in mapping various signal/image processing algorithms onto array architectures. There are many articles about the design of special VLSI architectures for specific applications. There are also some attempts to develop general methods for designing VLSI architectures systematically. Attempts to synthesize systolic arrays can be classified into three categories [31]. The first approach, called functional transformation, applies formal transformations to the mathematical expression of the

algorithm [32,33,34,35]. The second approach [36], called retiming, applies a graph transformation to obtain an equivalent systolic design from a non-systolic design. The third approach is called dependence mapping. It aims at extracting the dependencies between the variables of an algorithm and mapping the algorithm onto a systolic array in such a way that the dependencies are preserved. In this direction, Moldovan in [37] presented a method to map algorithms from dependence graph (DG) to systolic structures, which formalizes the mapping methodology by developing an algebraic transformation. A computer program is required for this method to examine all possibilities to obtain the best transformation. Miranker [38] uses the space-time representation method to mapping computational structures onto arrays. In [39,40], Rao, Jagadish and Kailath developed a synthesis procedure for designing array processors for regular iterative algorithms. It basically contains obtaining a schedule, mapping onto the processor space and determining the interconnections. In [31], Quinton described another method for the systematic design of systolic arrays. The method consists of deriving uniform recurrence equations for a given problem, finding a timing function and an allocation function, and finally projecting the computation domain to the processor domain. Kung in [2] summarized the mapping methodology for general algorithms onto systolic arrays and other arrays. Especially, a general mapping methodology which consists of a series of intermediate mapping stages is described in detail. The design criteria and design guidelines for achieving an optimal design are also given.

## 1.4 Objectives and Thesis Plan

In this section, the objective of the thesis and a brief outline of the thesis are given. Background material is presented in the corresponding chapters whenever the need arises.

As mentioned in the previous section, the increasing demands of speed and

performance have become evident with the expanding utilization of signal/image processing to industrial, medical, and military environments. As the numerical properties concerned, a class of structures that has been successfully exploited in realizing high performance fixed-point digital filters is the class of state-space structures. The mathematical tractability of these structures has permitted their synthesis in an optimal form that achieves minimum roundoff noise and low coefficient sensitivity. Furthermore, in the VLSI era, structures for filters suitable for the VLSI environment are likely to consist of a number of identical processing elements. Each such processing element could be implemented by a simple circuit. The state-space representation results in the matrix operations which are easy to be implemented by systolic arrays. Therefore, there is a growing interest in realizing state-space filtering. Unfortunately, state-space structures require more multiplications and additions than other realizations.

The recent advances in VLSI lead to a reconsideration of the design and implementation criteria in digital signal processing. The past approaches based on the minimization of the number of multiplications, delay elements and quantization noise are being replaced by a new set of criteria based on considerations, such as parallelism, pipelining, concurrency, modularity, etc. As a result, high-speed, high efficiency and low-cost special purpose array processors for high numerical performance digital filters will become desirable.

Motivated by these considerations, we have formulated a general methodology which can directly map the recursive algorithms onto VLSI architectures systematically, established the parallel computational models for recursive state-space filtering and applied the design methodology to obtain various high speed VLSI array architectures.

In chapter 2, based on the existing mapping methods, a modified approach for directly mapping recursive algorithms onto systolic arrays is given. Several examples are presented to illustrate the mapping rules. One question is that assuming we

do obtain an array structure for a certain algorithm, such as matrix-vector multiplication, how can we design an optimal array for it according to certain criterion? For the recursive filtering, the output is an indefinite-length sequence and the current output is dependent upon the previous output. This problem is more serious in the case of state-space filtering since more multiplication and addition operations are required for each sample processing. The optimization of design is more important for that kind of applications. Therefore, we give several design guidelines for designing the optimal arrays in the following sections.

In chapter 3, VLSI realizable high performance structures for real time one dimensional state-space recursive filtering are shown. The state-space representation is used to obtain a direct-form systolic array. For transformed state-matrices, an advanced state calculation array and a cascade-form of second-order structures are proposed. The advanced state calculation array increases throughput rate and efficiency $N+1$ times at the expense of increased hardware (2 times) as compared to the direct state-space implementation. In the cascade-form of second-order structures, each section is implemented by parallel processing elements. This structure results in a higher throughput rate and a higher efficiency with less hardware as compared to the direct state-space implementation.

Chapter 4 is devoted to new high-speed architectures for two-dimensional state-space recursive filtering. Based on the block processing technique, the advanced state update architecture is obtained which trades increased throughput rate and a simple input/output scheme for increased hardware. By exploring the inherent spatial concurrency in 2-D systems, a global speedup architecture is presented. This architecture consists of a number of column array processors and it works on multiple columns of images concurrently. The architecture is very modular and flexible. The throughput rate is adjustable and therefore can be very high.

Adaptive recursive filters are often implemented using direct-form realizations.

In chapter 5, to find better performance adaptive structures, gradient-based adaptive algorithms for state-space systems is studied. The stability control, convergence rate and noise performance of the filter are investigated. Since the adaptive state-space filtering requires intensive computations and all computations involved are matrix operations, a high speed VLSI architecture is given for real-time applications.

In chapter 6, based on the given design method, we present a very fast Kalman filter architecture for image restoration. Optimal restoration schemes for two dimensional images by Kalman filters lead to intensive computations. By using Roesser's local state-space structure to represent the image generation model and the degradation model, a simple composite dynamic system structure is obtained. From this model, the Kalman filtering equations are established by defining a proper state vector. To achieve very fast restoration, the processing recursion is arranged along the diagonal direction, and a dedicated VLSI array processor is given, which results in the throughput rate of one estimation per multiplication/addition time period. Finally, the summary of this research and the discussions of the future research work are given in chapter 7.

# Chapter 2

# Mapping Recursive Algorithms onto Array Structures

## 2.1 Introduction

Well designed VLSI architectures have the capability of performing pipelining (temporal parallelism) and multiprocessing (spatial parallelism), hence they can greatly increase the throughput rate of computation. Due to the use of massive parallelism and pipelining, the geometric and timing regularity, and identical (or few) simple processing elements, the systolic arrays are especially useful for the design of special-purpose VLSI architectures. Therefore, it is important to provide the designers with methods that help them implement DSP algorithms by systolic or systolic-type array architectures.

As indicated in chapter 1, a number of researchers have studied the problem of systematically deriving systolic implementations for a given algorithm. The results due to Moldovan [37], Fortes [41,42], Quinton [31], Kung [2] and others are well known and a systematic methodology for systolic array synthesis has emerged. However, there still are questions regarding the design of array structures for recursive algorithms. First, for a given recursive algorithm, is there a general method

to map the algorithm directly onto a systolic array? Secondly, for a subclass of recursive algorithms such as recursive filtering, the inputs are dependent upon the previous outputs. How can we design VLSI architectures with high throughput rate and high efficiency for real-time applications? Thirdly, are these architectures feasible to be manufactured and used? Most of existing methods assume no limitation on the size of VLSI architecture, or the number of processing elements on the VLSI chip. Such an assumption appears to be unrealistic, since there are several factors (physical, technology and complexity limitations) that actually constrain the integration level of the silicon IC technology [43]. The assumption of no size limitation of VLSI architectures is also not very economical and convenient because we have to design VLSI architectures with different sizes for different computation problems even though they may use the same algorithms. We have to find methods to partition computation problems so they can be solved on the basic fixed-size VLSI architectures.

In this chapter, we present a direct mapping methodology which can map recursive algorithms directly onto systolic arrays and is a modification of a mapping method using SFG's [2]. We also give several rules to optimize the systolic array design for recursive filtering algorithms [44,45]. In section 2.2, the basic design procedure for direct mapping onto systolic arrays is described, and the constraints for choosing the schedule vector and the projection vector are given. To illustrate the method, examples are discussed in detail. Then in section 2.3, the minimization of the computation time and the pipeline period, the optimization of I/O lines, the reduction of the array size, and the improvement of the utilization efficiency are also addressed.

## 2.2 Direct Mapping to Systolic Array

In general, the design of a special-purpose array architecture for a given problem can be hierarchically decomposed into four steps: a) mapping algorithm to dependence graph (DG); b) mapping DG to obtain the signal flow graph (SFG); c) mapping SFG onto a array architecture, such as systolic array (SA) or other type of array; d) providing implementation details for the processors and the input/output lines. It is shown here that the intermediate mapping stage, the signal flow graph, can be avoided. The overall procedure can be made simpler and more direct to implement.

### 2.2.1 Dependence Graph for Recursive Algorithms

An algorithm is said to be a recursive algorithm if the algorithm can be described by recursive equations. For example, the matrix-vector multiplication $\mathbf{C} = \mathbf{AB}$ is a recursive algorithm since it can be expressed by the following single assignment recursive equation:

$$c_i^{(j+1)} = c_i^{(j)} + a_i^{(j)} b_i^{(j)} \tag{2.1}$$

where $j = 1, 2, ..., N$ is the recursion index or time index, $i$ is the space index, and

$$c_i^{(1)} = 0 \tag{2.2}$$

$$a_i^{(j)} = A(i, j) \tag{2.3}$$

$$b_i^{(j)} = B(j) \tag{2.4}$$

Many signal processing operations can be described by recursive equations, therefore, they are recursive algorithms. According to the type of interconnections, the DSP algorithms are divided into two categories: local interconnections and global interconnections. A majority of the algorithms used in signal and image processing have local interconnections, such as matrix multiplication, convolution, IIR filtering, and so on. Some other algorithms involve globally separated space

indices, they are called globally recursive algorithms. For example, FFT is an algorithm which has global interconnections between the computing nodes. We will focus on the locally recursive algorithms in this thesis, because these algorithms can be mapped onto an array structure with local communications only.

Notice that recursive filtering, such as IIR filtering, is a special kind of locally recursive algorithm. A filter is a recursive filter because the current output depends on the input and the previous outputs. There is a feedback in the system. When we say a recursive algorithm, it is not necessary to be a recursive filter algorithm. In this thesis, the emphasis is how to map recursive filtering algorithms onto array architectures.

A DG is a graphical representation of a single assignment recursive algorithm and a single assignment code is a form where every variable is assigned one value only during the execution of the algorithm. It shows the dependence of the computations that occur in an algorithm. Given a single assignment algorithm, the DG can be constructed based on the space-time indices in the algorithm. A node represents each basic computation or combination of computations, such as multiplication, division, addition, comparison etc., in the algorithm. A directed arc between nodes is given if and only if the computation represented by the second node is dependent upon the computation represented by the first node. For the example of matrix-vector multiplication, if the matrix has dimension of $N \times N$, the vector is $N \times 1$, there will be $N^2$ computation nodes in the algorithm. By viewing each dependence relation as an arc between the corresponding variables located in the $i - j$ index space, a DG for $N = 4$ is obtained in Figure 2.1(a).

Notice that in Figure 2.1(a), the value $B(j)$ of the vector **B** is broadcast to all nodes having the same $j$-index. This means that global communication is involved in this dependence graph.

We refer it as to the DG with global communication. In order to obtain a DG with local communication, we can transfer $B(j)$ from one node to the next one

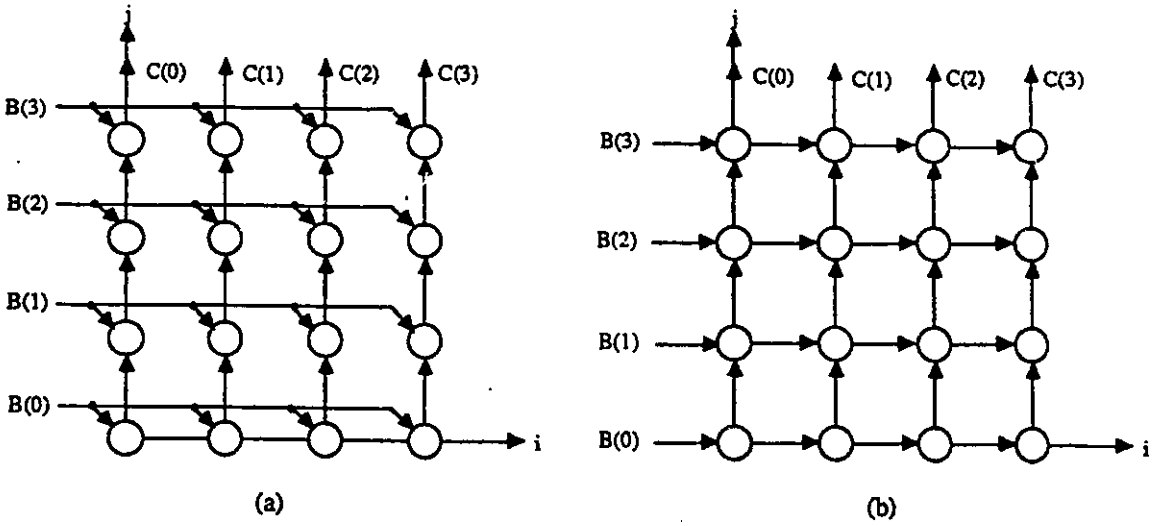



Figure 2.1: Dependence graph for matrix-vector multiplication. (a) with global communication; (b) with local communication. (The elements of matrix A are assumed stored in the nodes.)

without modification, that is $b_{i+1}^{(j)} = b_i^{(j)}$ and $b_0^{(j)} = B(j)$. As a result, a localized DG for matrix-vector multiplication is shown in Figure 2.1(b).

### 2.2.2 Processor Assignment and Scheduling

To achieve high speed processing, one simple method is to use a distinct processor for executing the computation represented by every node in DG. However, there are dependencies between those computations as shown in the DG. Therefore, each processor is active only for a fraction of the time required for completing the algorithm. The array architecture will be very inefficient. For a better utilization efficiency, it is necessary to reuse the same processor to handle a large number of computations. The principle of design is to obtain a desirable processing rate by using fewer processors as well as simple interconnections.

In determining an implementation for an algorithm, we must decide how many processors are to be used, at which processor each computation will be performed and in which order the operations will be executed in each node. In other words,

there are two things to be done for mapping a DG onto an array processor: one is the processor assignment, the other one is the scheduling.

Processor assignment can be done by the projection method, by which nodes of the DG along a straight line are assigned to a common processor element (PE). Mathematically, a linear projection is often represented by a projection vector $d$ or projection matrix $p$, where $p^t d = 0$ and $t$ indicates the transpose of a vector or a matrix hereafter. To obtain a systolic array, a scheduling scheme has to be applied to specify the sequence of the operations in all PEs. For any dependence arc $b$, the schedule vector $s$ and the projection vector $d$ have tc satisfy the following conditions:

$$s^t b > 0, \tag{2.5}$$

$$s^t d > 0. \tag{2.6}$$

These conditions are necessary because in the systolic array architecture, no global communication is allowed. There is at least one delay for each edge in the array. The nodes projected onto one processor have to be performed at different time. There is at least one delay between the two consecutive nodes. These conditions are also sufficient because If $s^t b > 0$, no edges with no delays can be resulted. $s^t d > 0$ ensures that one delay or more delays will exist between two consecutive computations for every processor element.

The objective of mapping is to project an $N$ dimensional DG into an $N-1$ dimensional SA. During the mapping, the node $n$ in DG is mapped to the processor $c$ in the array, the arc $b$ in the DG will be mapped to the edge $e$ in the systolic array. Assume that the coordinates of DG start from $(0,0)$, the mapping equations are given in [2]. However, for output mapping, a new equation should be given as shown in (2.10).

$$Node\ \ Mapping \ : \ c = p^t n \tag{2.7}$$

$$Arc\ Mapping\ :\ \begin{bmatrix} D(e) \\ e \end{bmatrix} = \begin{bmatrix} s^t \\ p^t \end{bmatrix} [b] \tag{2.8}$$

$$Input\ Mapping\ :\ \begin{bmatrix} T(c) \\ c \end{bmatrix} = \begin{bmatrix} s^t \\ p^t \end{bmatrix} [n] \tag{2.9}$$

$$Output\ Mapping\ :\ \begin{bmatrix} T(c) \\ c \end{bmatrix} = \begin{bmatrix} s^t \\ p^t \end{bmatrix} [n] + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.10}$$

where $D(e)$ is the number of delays on each edge $e$ required, and $T(c)$ is the number of delays for each input and output relative to the starting time. Notice that for the same node the number of delays for the output data is different by one delay from the input data mapping, this can be understood by considering the execution time. For the input and the output on the same node, the coordinates are the same. However, the output is generated one clock time later than the input data. Thus, we have to add one delay for reflecting the execution time.

### 2.2.3 Mapping Procedure

In practice, the mapping procedures is as follows:

1. Select hyperplanes. Given a DG, choose the hyperplanes so that all nodes on the same hyperplane are to be processed at the same time. The selection of hyperplanes depends upon the recursive algorithm. It is suggested to start from the node without any intermediate variables generated from other nodes, usually, the node at the origin of the time-space coordinates. Then following the outputs of this node, we can find the next hyperplane, on which all nodes can start to work at the same time.

2. Obtain the schedule vector. The schedule vector is then normal to the hyperplanes. The hyperplanes, therefore the schedule vector, may be chosen different as long as the condition $s^t b > 0$ is satisfied. However, different schedule vectors will result in different computation time.

3. Select projection vector $d$. This is determined by the condition: $s^t d > 0$. $d$ then determines the projection matrix $p$ as $p^t d = 0$.

4. Combination of $s$ and $p^t$ to $\begin{bmatrix} s^t \\ p^t \end{bmatrix}$. That has been found to give direct mapping from DG to SA, resulting in a SA that has minimal delay for every edge (but no edges without delay).

5. Projection DG to SA and calculation of delays of each edge.

### 2.2.4 Examples

In order to illustrate this direct mapping method, several examples are given as follows.

- Example 1: Convolution-FIR filtering.

  Given two sequences $u(i)$ and $w(i), i = 0, 1, ...N-1$, the convolution of the two sequences is

  $$y(i) = \Sigma_{j=0}^{i} u(j) w(i-j) \tag{2.11}$$

  where $i = 0, 1, ..., 2N-2$. The localized DG for convolution with $N = 4$ is given in Figure 2.2.

  Notice that $b^t$ in the DG are: $\begin{bmatrix} 1 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 1 \end{bmatrix}$, and $\begin{bmatrix} 1 & 1 \end{bmatrix}$, the hyperplanes for convolution are determined as shown in Figure 2.2. The schedule vector $s$ is orthogonal to the hyperplanes, i.e., $s^t = \begin{bmatrix} 1 & 1 \end{bmatrix}$.

  Because the projection $d$ has to satisfy the condition $s^t d > 0$, $d^t$ may be chosen as $\begin{bmatrix} 1 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 1 \end{bmatrix}$, and $\begin{bmatrix} 1 & 1 \end{bmatrix}$.

  For $d^t = \begin{bmatrix} 1 & 1 \end{bmatrix}$, according to (2.7)-(2.10):

  The node mapping is given by

  $$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i - j. \tag{2.12}$$

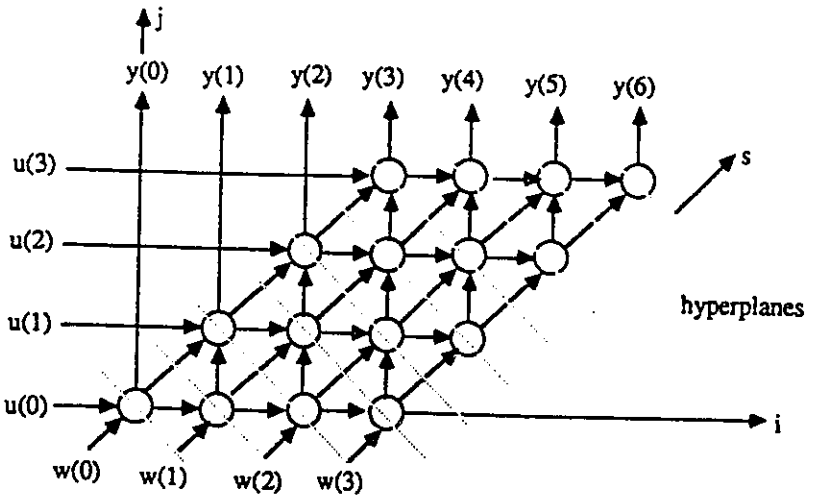



Figure 2.2: Localized dependence graph for convolution

The arc mapping is :

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & -1 & 0 \end{bmatrix}. \tag{2.13}$$

The input $u(j)$ is mapped to

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ i \end{bmatrix} = \begin{bmatrix} 2i \\ 0 \end{bmatrix}. \tag{2.14}$$

The input $w(i)$ is mapped to

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ 0 \end{bmatrix} = \begin{bmatrix} i \\ i \end{bmatrix}. \tag{2.15}$$

The output $y(i)$ for $0 \leq i \leq 3$ is mapped to

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ i \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2i \\ 0 \end{bmatrix} \tag{2.16}$$

The output $y(i)$ for $3 \leq i \leq 6$ is mapped to

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} i+4 \\ i-3 \end{bmatrix}. \tag{2.17}$$

The resulting systolic array is shown in Figure 2.3(a). If we preload the coefficients $w(i)$ into the processing elements, and notice that the output $y(4), y(5)$ and $y(6)$ can be obtained from the leftmost cell too since there is an arc from the right to left for transferring the partial results, a modified array is obtained in Figure 2.3(b).

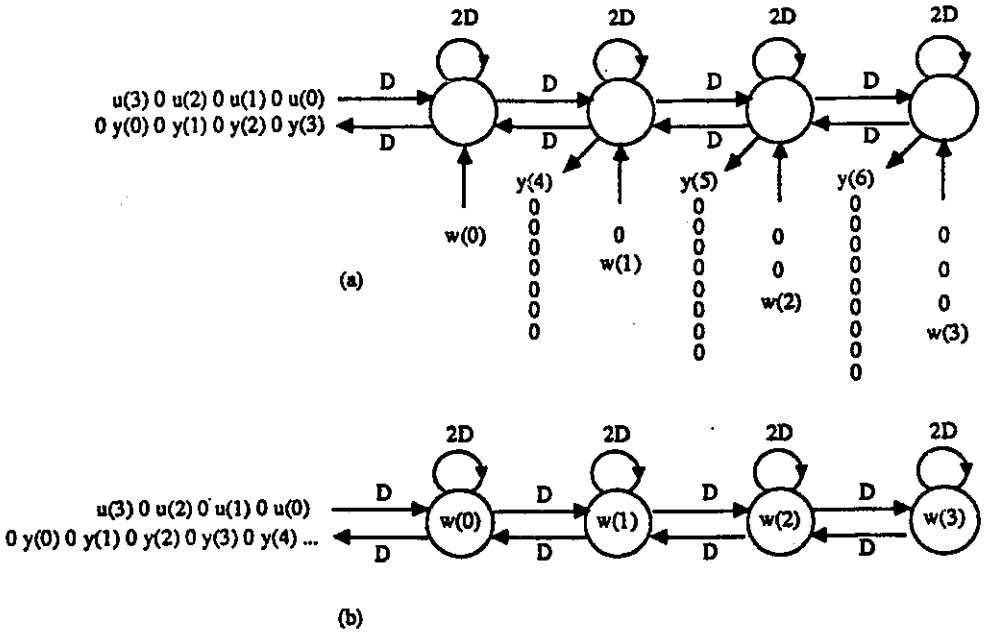



Figure 2.3: Systolic array for convolution: (a) direct mapping result; (b) modified array.

- Example 2: AR Filtering.

  An AR filtering algorithm is described by the following difference equation: for $i = 0, 1, 2..., \infty$

$$y(i) = \Sigma_{j=0}^{N-1} a_j y(i-j-1) + u(i) \tag{2.18}$$

  The localized DG for this AR filtering algorithm with $N = 4$ is given in Figure 2.4. Since it is a recursive filter, the output is an indefinite-length sequence even if the input sequence $u(i)$ has a finite duration. In other

words, the index space $i$ is not finite. Therefore, the projection vector is unique, i.e., $d^t = \begin{bmatrix} 1 & 0 \end{bmatrix}$. Other projection vectors will result in an array with infinite processing elements. Therefore, $p^t = \begin{bmatrix} 0 & 1 \end{bmatrix}$. Since $b^t = \begin{bmatrix} 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 \end{bmatrix}$, thus, the schedule vector $s^t$ can be chosen as: $\begin{bmatrix} 2 & 1 \end{bmatrix}, \begin{bmatrix} 3 & 1 \end{bmatrix}, \begin{bmatrix} 3 & 2 \end{bmatrix}$ and so on.

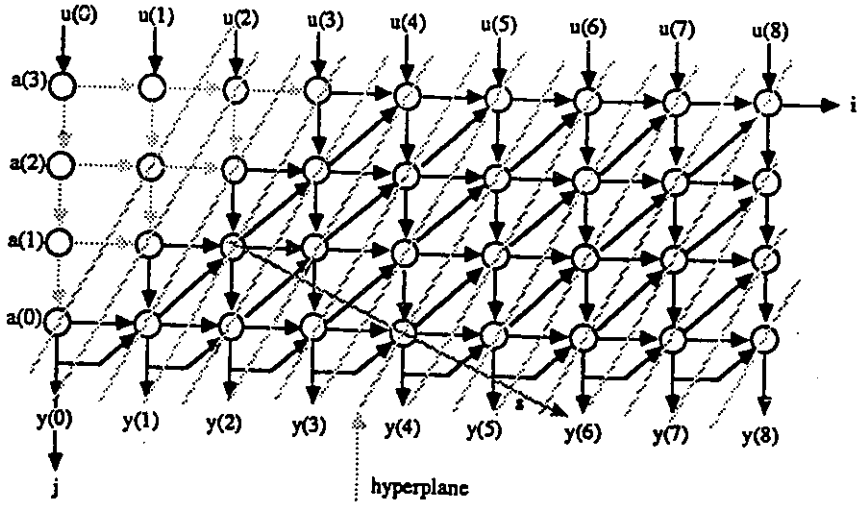



Figure 2.4: Dependence graph for AR filtering.

For $s^t = \begin{bmatrix} 2 & 1 \end{bmatrix}$:

The arc mapping is given by

$$\begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix}. \tag{2.19}$$

The input $u(i)$ is mapped to

$$\begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ 0 \end{bmatrix} = \begin{bmatrix} 2i \\ 0 \end{bmatrix}. \tag{2.20}$$

The input $a(j)$ is mapped to

$$\begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ j \end{bmatrix} = \begin{bmatrix} j \\ j \end{bmatrix}. \tag{2.21}$$

The output $y(i)$ is mapped to

$$\begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2i+4 \\ 3 \end{bmatrix}. \tag{2.22}$$

Similarly, we can preload the coefficients $a(j)$ into the array. The resulted systolic array is shown in Figure 2.5.

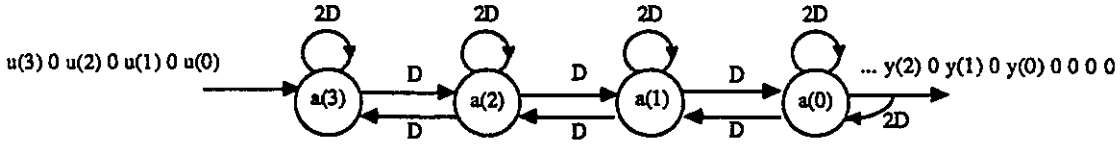



Figure 2.5: Systolic array for AR filtering with minimum delays.

- Example 3: Matrix-Matrix Multiplication.

  Many signal processing operations are based on matrix-matrix multiplication and the multiplication of a data vector by a matrix or vector of coefficients. The matrix-matrix multiplication can be viewed as a multiplication of a matrix by a series of vectors. The dependence graph for matrix-matrix multiplication is presented in Figure 2.6.

  Since $s^t b > 0$ is required, $s$ can not lie in the planes $(i,j)$, $(i,k)$ or $(j,k)$. The best choice is $s^t = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$. While $d^t$ can be $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$ and others. For $d^t = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, p^t = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$,

  the node mapping is :

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} j \\ k \end{bmatrix}, \tag{2.23}$$

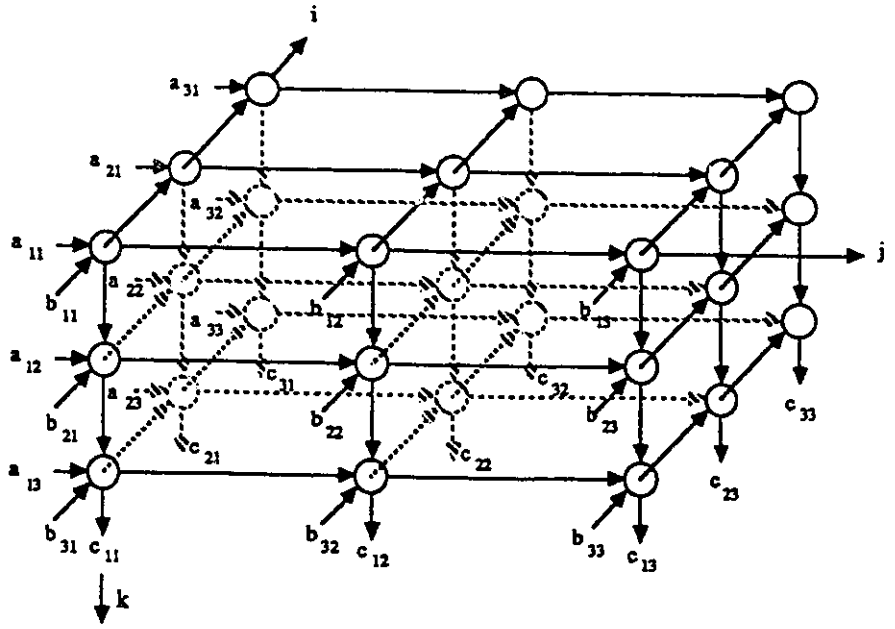



Figure 2.6: Dependence graph for matrix-matrix multiplication.

the arc mapping is:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.24}$$

the input mapping :

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i & 0 \\ 0 & j \\ k & k \end{bmatrix} = \begin{bmatrix} i+k & j+k \\ 0 & j \\ k & k \end{bmatrix}, \tag{2.25}$$

the output mapping:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} i+j+3 \\ j \\ 2 \end{bmatrix}. \tag{2.26}$$

The systolic array for this mapping is a two-dimensional array shown in Figure 2.7. One matrix, say matrix B resides in the array, the other matrix, A

is entered from left side of the array, and the output matrix, C is obtained from the bottom of the array.

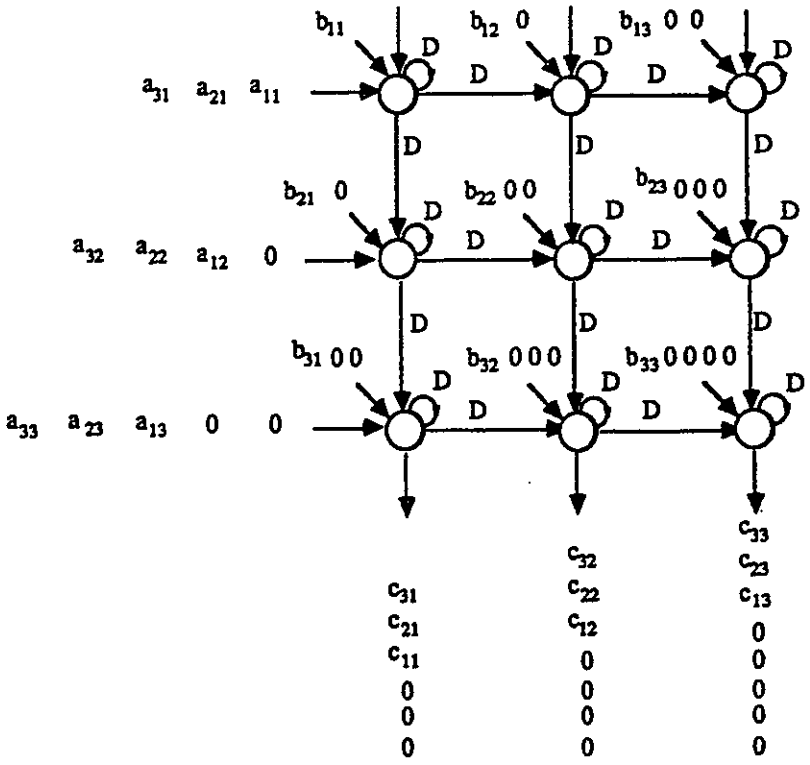



Figure 2.7: Systolic array for matrix-matrix multiplication.

## 2.3 Proposed Design Optimization

It is well known that design criteria for VLSI array processors are: maximum parallelism, maximum pipelinability, balance among computations, communications and memory, trade-off between computation and communication, and numerical performance and quantization effects. In practice, the final choice of optimality criteria will have to be application dependent. Some practical considerations are computation time, pipelining period, utilization efficiency, array size, I/O lines and so on. In this section, we will discuss how the design can be optimized for recursive algorithms.

### 2.3.1 Minimization of computing time and pipelining period

Suppose $T_c$ is the time interval of the clock period. Computation time, denoted by $T$, is defined as the time interval between starting the first computation and finishing the last computation of a problem by the array processor. Pipeline period is defined as the time interval between two successive computations in a given processor. If the pipeline period is denoted by $\alpha$, then the processor does one clock period useful work during every $\alpha$ clock times. Usually $\alpha = 1, 2, 3$ or other integers.

There are two points worthy of remarks regarding the relation between computation time and pipeline period. 1). In the case that the DG is finite, such as fixed length convolution or a fixed size matrix-matrix multiplication as shown in last section, the computation time is perhaps more important criterion than the pipeline period. However, in many DSP applications, such as the IIR filtering we will discuss in the following chapters, input data lengths are usually very long or infinite. In this case, minimizing the computation time is equivalent to minimizing the pipeline period. If the pipeline period is short, so is the computation time. 2). In order to achieve high speed processing, parallel processing techniques can be used at different processing levels. Given an application problem, we can map the dependence graph of the algorithm onto an array and process the data by the array processor sequentially. We refer to this method as a sample level speed-up. In this case, the computation time is directly proportional to the pipeline period. On the other hand, we can exploit the inherent parallelism in the algorithm and process input data concurrently. This method can be referred to as a sequence level speed-up. In this case, the computation time is not only dependent upon the pipeline period, but also the arrangement of and the number of the processing elements.

Here, we consider the sample level speedup case. According to the mapping methodology given above, a different choice of projection vectors and schedule vectors will result in different systolic arrays. Since the pipeline period is the time

interval between two consecutive computations for a processor, and the projection vector, when it is coprime, will determine the coordinate distance between two consecutive computations, the pipeline period $\alpha$ can be calculated as:

$$\alpha = s^t d \tag{2.27}$$

where both the systolic schedule $s$ and the projection vector $d$ are in coprime form, i.e., the components of the vectors are coprime.

From examples given in the last section, it is also clear that the projection vector determines the array architecture, the schedule vector determines the computation order and at which time each computation is to be executed by the processor. Therefore, it determines the computation time. If the schedule vector $s$ is coprime, then the computation time can be determined by:

$$T = (L-1)v\alpha + s^t q + 1 \tag{2.28}$$

where $q$ is the coordinate vector for the final node in DG, $s^t q + 1$ is the time for finishing the last computation in the dependence graph for the first sample. $L$ is the number of the input samples. When the inputs are scalar signals, $v$ is equal to 1. When the inputs are vector signals, there are two cases. If the components of each vector input to the array in parallel, $v$ equals to 1. If the components of each vector input to the array serially, $v$ is equal to the length of the vector.

From (2.27) and (2.28), it is obvious that the pipeline period $\alpha$ is dependent on both $s$ and $d$. The computation time is determined by $s$ and $\alpha$. Thus, it is also determined by $s$ and $d$. Therefore, minimizing $T$ and $\alpha$ equals minimizing $s$ and $d$, under the constraints:

$$s^t b > 0 \tag{2.29}$$

$$s^t d > 0 \tag{2.30}$$

for any dependence arc $b$ in the DG. Here, minimizing $s$ and $d$ means minimizing their components.

Now we will use the example of matrix-vector multiplication to illustrate how the schedule vector and the projection vector effect the computation time and the pipeline period. The DG of matrix-vector multiplication is given in Figure 2.1. In that DG, the elements of matrix A are assumed sitting in the nodes. In effect, they are also the input data. The DG with matrix A as inputs is given in Figures 2.8-2.11.

By using different projection vectors and schedule vectors, four different systolic arrays are obtained. For this example, $L = 1$. Hence, the computation time $T = s^t q + 1$.

Case 1. For $s = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $d = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, we have:

$$\textit{Node mapping} \ : \ \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = j$$

$$\textit{Arc mapping} \ : \ \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$\textit{Input mapping} \ : \ \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i & i \\ j & 0 \end{bmatrix} = \begin{bmatrix} i+j & i \\ j & 0 \end{bmatrix}$$

$$\textit{Output mapping} \ : \ \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ j \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 4+j \\ j \end{bmatrix}$$

$$\textit{Computation time} \ : \ T = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} + 1 = 7$$

$$\textit{Pipeline period} \ : \ \alpha = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$$

Case 2. For $s = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $d = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, we have:

$$\textit{Node mapping} \ : \ \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i - j$$

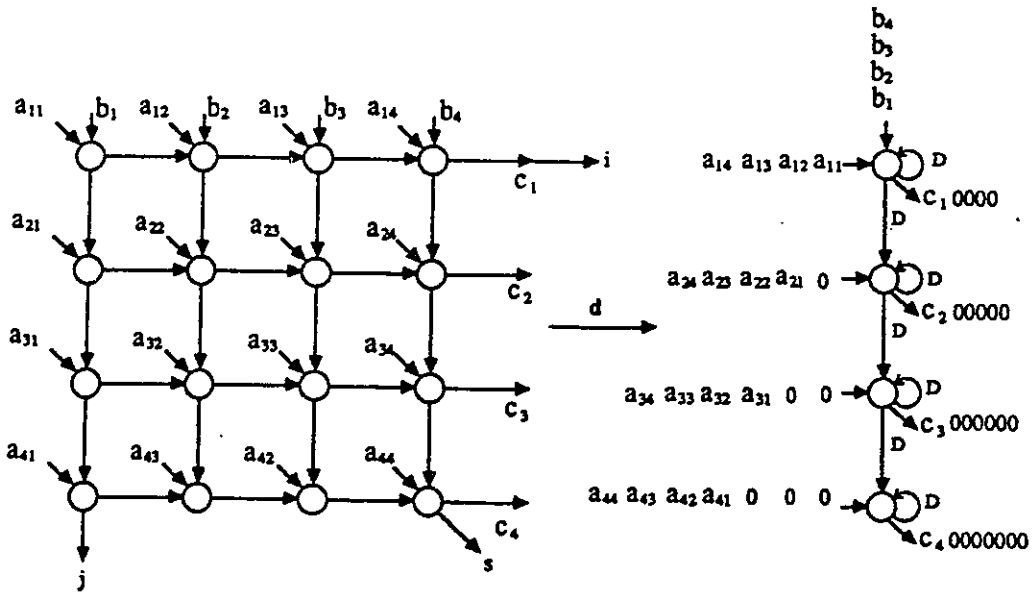



Figure 2.8: Systolic array 1 for matrix-vector multiplication.

$$\textit{Arc mapping} \;:\; \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\textit{Input mapping} \;:\; \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i & i \\ j & 0 \end{bmatrix} = \begin{bmatrix} i+j & i \\ i-j & i \end{bmatrix}$$

$$\textit{Output mapping} \;:\; \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 3 \\ j \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 4+j \\ 3-j \end{bmatrix}$$

$$\textit{Computation time} \;:\; T = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} + 1 = 7$$

$$\textit{Pipeline period} \;:\; \alpha = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2$$

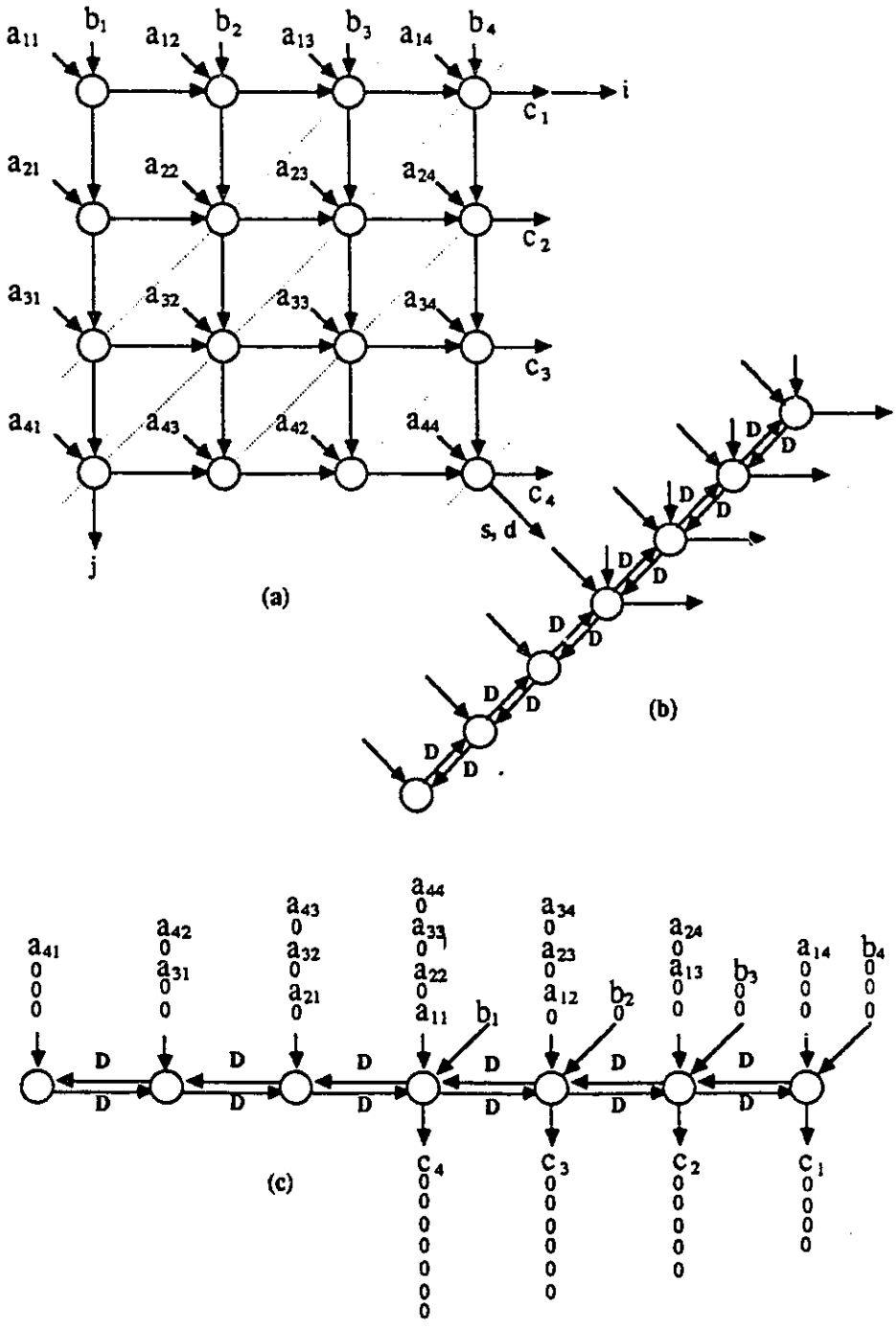



Figure 2.9: Systolic array 2 for matrix-vector multiplication.

Case 3. For $s = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ and $d = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, we have:

$$\textit{Node mapping} \ : \ \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = j$$

$$\textit{Arc mapping} \ : \ \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix}$$

$$\textit{Input mapping} \ : \ \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i & i \\ j & 0 \end{bmatrix} = \begin{bmatrix} 2i+3j & 2i \\ j & 0 \end{bmatrix}$$

$$\textit{Output mapping} \ : \ \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ j \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 7+3j \\ j \end{bmatrix}$$

$$\textit{Computation time} \ : \ T = \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} + 1 = 16$$

$$\textit{Pipeline period} \ : \ \alpha = \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 2$$

Case 4. For $s = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ and $d = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, we have:

$$\textit{Node mapping} \ : \ \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i - j$$

$$\textit{Arc mapping} \ : \ \begin{bmatrix} 2 & 3 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 1 & -1 \end{bmatrix}$$

$$\textit{Input mapping} \ : \ \begin{bmatrix} 2 & 3 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i & i \\ j & 0 \end{bmatrix} = \begin{bmatrix} 2i+3j & 2i \\ i-j & i \end{bmatrix}$$

$$\textit{Output mapping} \ : \ \begin{bmatrix} 2 & 3 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 3 \\ j \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 7+3j \\ 3-j \end{bmatrix}$$

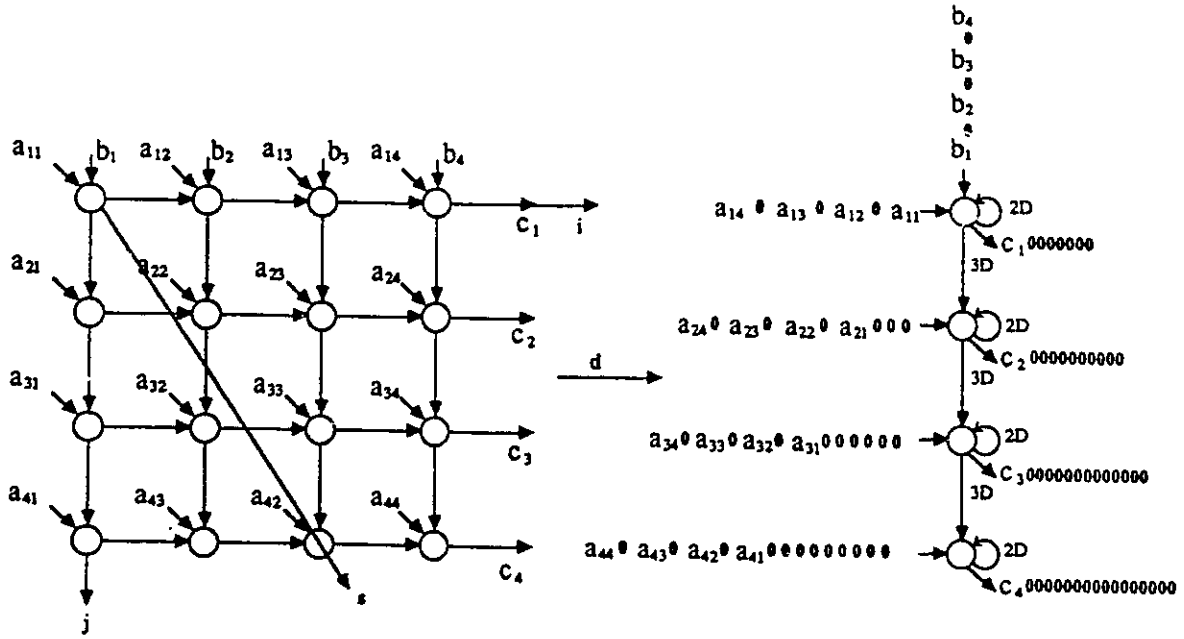



Figure 2.10: Systolic array 3 for matrix-vector multiplication.

$$\textit{Computation time} \;:\; T = \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 3 \\ \vdots \\ 3 \end{bmatrix} + 1 = 16$$

$$\textit{Pipeline period} \;:\; \alpha = \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = 5$$

In this example, when $s$ and $d$ are minimum, as for case 1, both the computation time and the pipeline period are minimized. For case 2, the computation time is still minimum since the schedule vector is the same as the case 1. However, the pipeline period is increased to 2. Notice that the minimum computation time is obtained at the expense of an increase in number of the processing elements. The larger schedule vector $s$ results in more computation time as shown in cases 3 and 4, which should be avoided.

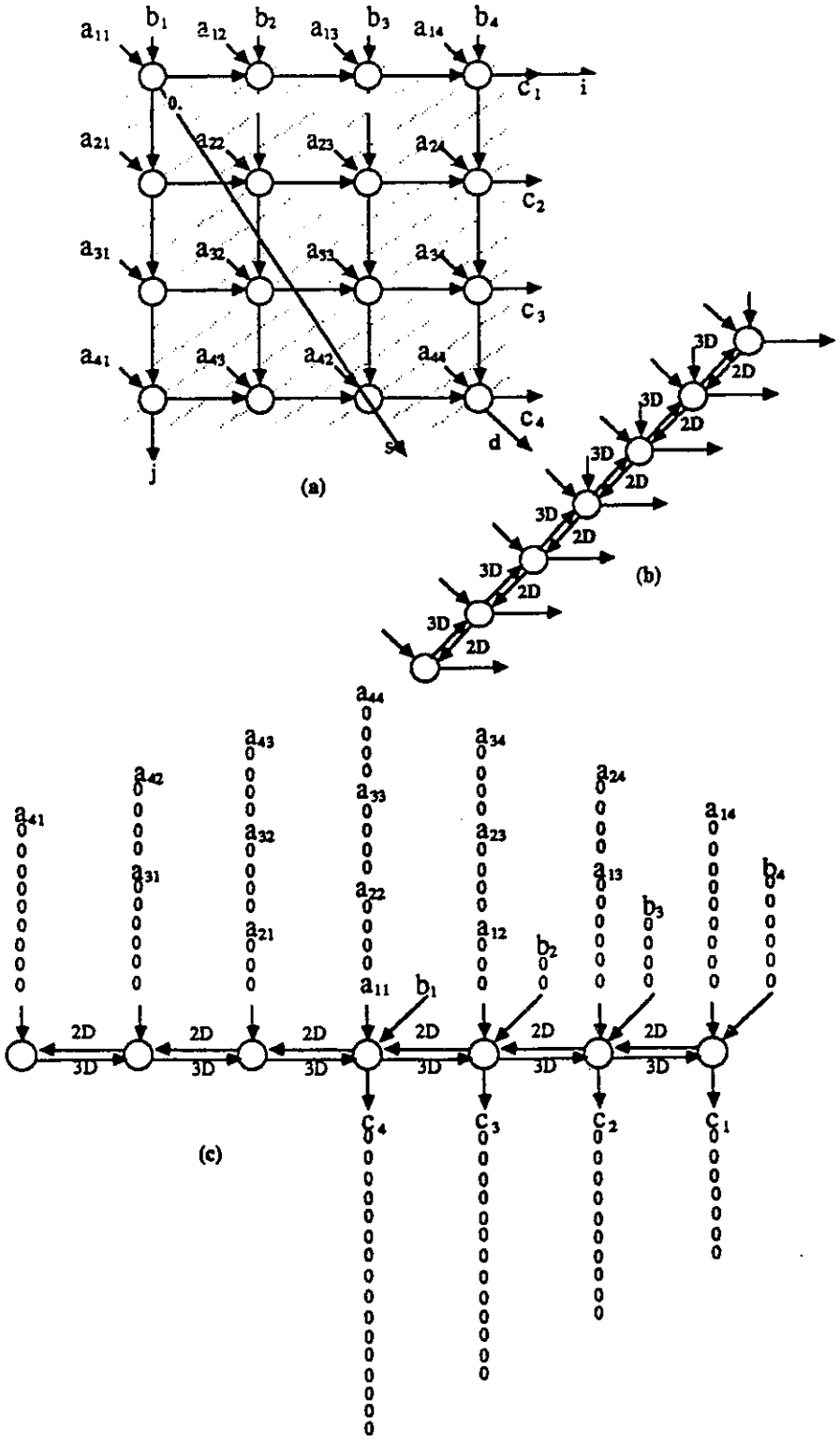



Figure 2.11: Systolic array 4 for matrix-vector multiplication.

### 2.3.2 Optimization of I/O lines

As shown in the examples above, projecting a DG onto an systolic array may result in a large number of I/O ports. Furthermore, in many cases, resulting I/O will fall on interior nodes of the array, i.e., the input/output data are not only on the boundary of the array. A typical requirement in the design of processor arrays is that inputs must be fed in and outputs tapped at the boundaries of the processor arrays and the input/output lines are to be as few as possible. When these arrays are implemented at the board level with one chip per processor, the restriction of inputs and outputs to the boundaries is unnecessary and results in inefficiency. However, when these arrays will be completely or partially implemented on one chip, it is very important to be able to map an algorithm onto an array with minimum I/O lines and without interior I/O. In both cases, it is desirable to load the inputs and unload the outputs efficiently.

To optimize the input/output lines, several methods are possible. First of all, when coefficients, as the inputs to the array, are constants during the processing period, they can be preloaded into the processors before starting the computations. Hence, they can be ignored during the mapping. For example, the coefficients for convolution-FIR filtering and AR filtering, the elements of one matrix for matrix-matrix multiplication and matrix-vector multiplication can be treated in this way. Similarly, the output data, accumulated in the elements, after all computation is done, they can be post-dumped. The preloading makes the implementation hardware easier by adding operation time for loading the coefficients. When assuming that the elements of matrix are sitting in the processors, all I/O for the DG of matrix-vector multiplication occurs on the boundaries as shown in Figure 2.1.

Next, although all I/O for a DG is on the boundaries, projection of the DG onto a linear array may result in I/O on the array's interior nodes. For example, the DG for $\mathbf{C} = \mathbf{AB} + \mathbf{C}$ is given in Figure 2.12(a), where $\mathbf{A}$ is a matrix, $\mathbf{B}, \mathbf{C}$ are vectors. By using projection vector $d^t = [0 \quad 1]$, a systolic array is obtained

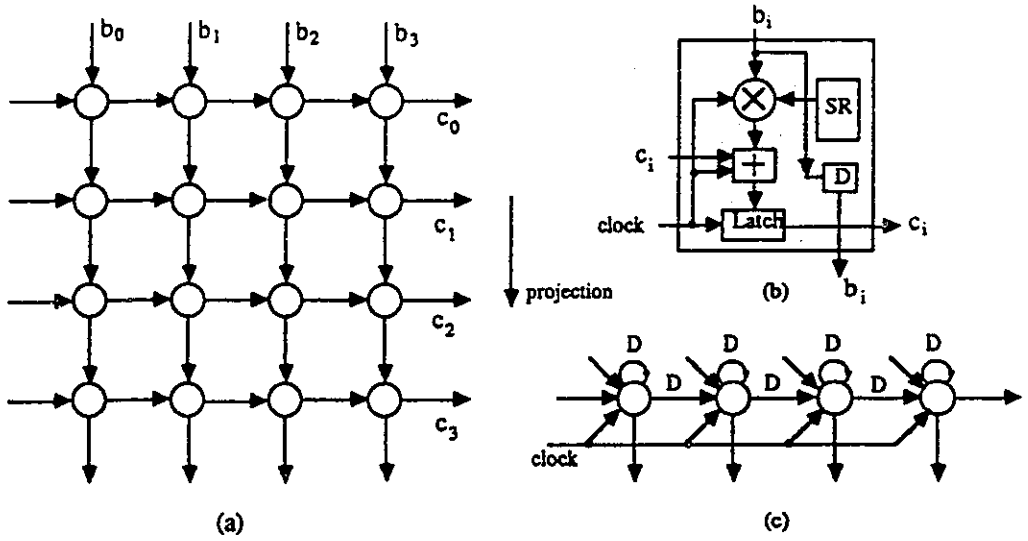



Figure 2.12: Matrix-vector multiplication and summation. (a)Dependence graph; (b) A systolic array from vertical projection; (c) Configuration of basic cell.

in Figure 2.12(c), which has interior input/output lines. The basic cells for this array is the simplest one shown in Figure 2.12(b). For each cell in this array, there are two input data lines and two output data lines. In addition, there is a control signal connected to each cell to synchronize the operations. If we wish all of the inputs and outputs to be at the boundaries, the DG has to be expanded such that its corresponding boundaries are parallel to the chosen projection direction [2,39]. Then the projection of these boundary nodes will result in points that lie on the boundary of the processor space. Given a DG, the mapping procedure without interior I/O is as follows:

1. Choose the projection vector and decide the I/O boundary of the DG. It is obvious the I/O boundary of the DG is parallel to the projection direction.

2. Expand the DG with some non operating nodes so that all inputs and outputs are along the boundaries of the DG. Note that in the expanded region of the DG, the algorithm does not perform any computation, but only propagates the relevant inputs to the appropriate locations in the true DG and transmits

the outputs from interior nodes to the boundary cells of the array.

3. Modify the functions of the processing elements (PEs) and determine the control signal. In the expanded region of the DG, the arcs and the nodes are different from those in the original DG. However, they will mapped onto the array according to the same rules. Sometimes the PEs only propagates the input or output data. At other times they perform the computations. Therefore, a control signal is needed when the function of the PE needs to be modified.

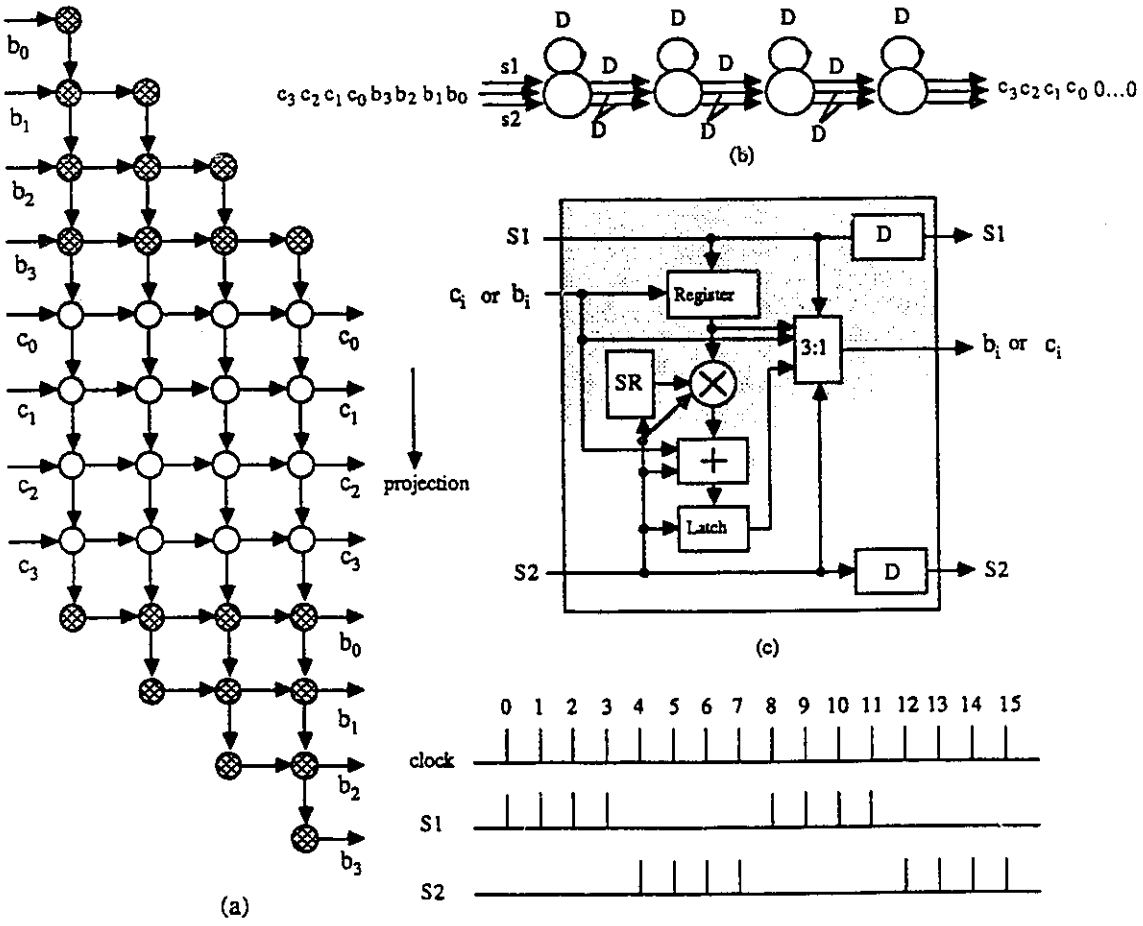



Figure 2.13: Mapping without interior I/O for matrix-vector multiplication and summation.

For the example of matrix-vector multiplication and summation, when we choose the projection vector as $d^t = \begin{bmatrix} 0 & 1 \end{bmatrix}$, the border lines for this projection direction are shown in Figure 2.13. We extend the DG with non operating nodes, depicted by filled circles. This new DG is then mapped onto an array according to the mapping equations. Since $d^t = \begin{bmatrix} 0 & 1 \end{bmatrix}$, and $s^t = \begin{bmatrix} 1 & 1 \end{bmatrix}$, we have:

$$\textit{Arc mapping} \quad : \quad \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\textit{Input mapping} \quad : \quad \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ j \end{bmatrix} = \begin{bmatrix} j \\ 0 \end{bmatrix}$$

$$\textit{Output mapping} \quad : \quad \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ j+4 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} j+8 \\ 3 \end{bmatrix}$$

The systolic array is given in Figure 2.13(b). It is a simple array, the inputs and outputs are at the boundary cells. However, the configuration of the basic cells and the control signal have to be changed so that the array can operate properly. As shown in Figure 2.13(c), we need an extra register in each cell for storage of the input data, and a selector to choose the output data from three resources: the input data, the partial results, and the data in the register. For switching the processor from transmission of data to computing, two control signals are required as shown. There is only one input and one output for each cell, but there are two control signals.

Another method for reducing the I/O lines is to use a serial/parallel buffer and a parallel/serial buffer. The input data can be stored in the buffer sequentially and then fed to the array in parallel. In this case, the speed for putting the data in the buffer must be $N$ times higher than the speed at which the array operates. In the special case, when there is a correlation between consecutive input vectors, we can use this property to simplify the I/O lines. For example, as shown in Figure 2.14(a),there are different delays for inputs according to the mapping equations. If $B_1$ is a delayed version of $B_2$ for the matrix-vector multiplication-addition, the

inputs at any moment for all entries are the same. Therefore, only one input data is needed actually. It can be applied to the boundary cell and broadcast to the others. The resulting array is shown in Figure 2.14(b).

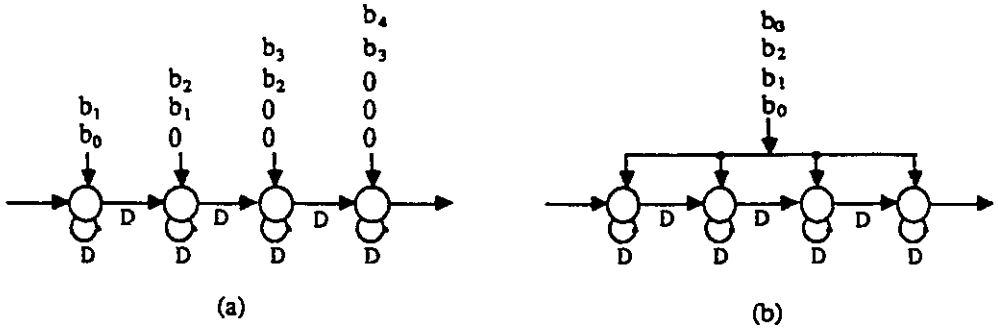



Figure 2.14: Reduction of input lines by distributing the inputs.

As a conclusion, there always is a trade-off possible between different designs. Optimization of I/O lines can be achieved by adding more operation time for the preloading method, using more complicated cells and more control signals for the expanding DG method, and including a sequential/parallel buffer or using the broadcasting technique.

### 2.3.3 Reduction of Array Size and Partitioning

Up to now, the design method described above is based on the assumption that there is no limitation on the size of VLSI architecture, or the number of processing elements on the VLSI chip. Such an assumption is not practical as we discussed at the very beginning of this chapter. It is very common that the problem size is much larger than the array size, and the problem sizes are variable. Therefore, it is important to be able to design a small fixed size array, which can fit on one VLSI chip, and mapping computations of a larger size problem to an array processor of a smaller size. There are basically two methods to partition a problem with large size. One is to partition the problem into more than one VLSI chip if an entire

function cannot be accommodated on a single chip. The other one is to partition the problem into several subproblems and implement these on a small size array sequentially.

To partition a problem into a smaller size problem, the first consideration is whether the algorithm can be decomposed mathematically. If it is possible to do so, then the implementation is very easy. For example, if we factor the numerator and denominator polynomials of an IIR digital filter's transfer function, the transfer function, $H(z)$, can be written in a cascade form of first- and second-order subsystems, or it can be expressed as a partial-fraction expansion in the parallel form of first- and second-order subsystems. Thus, we can design a fixed size array processor for the small order sections. The overall problem is then implemented on more than one chip. This method will be used in the following chapters. It will be shown that by decomposing the algorithm the hardware complexity for state-space filtering can be reduced significantly.

The second partitioning method is to partition the DG of a problem into sub-DGs. There are two methods for mapping the partitioned DG to an array: the locally sequential globally parallel (LSGP) scheme and the locally parallel globally sequential (LPGS) scheme.

1. LSGP scheme: In the LSGP method, one sub-DG is mapped to one PE. Thus, the number of blocks is equal to the number of PEs. For each block, we choose a proper projection vector and a schedule vector to map it onto a PE. Then these PEs are pipelined to form a linear array. Each PE sequentially executes the computations of the corresponding block. Different PEs are working in parallel. Local memory within each PE is needed for saving the intermediate results. The LSGP schemed array is given in Figure 2.15. Notice that the projection vector $d^t = \begin{bmatrix} 0 & 1 \end{bmatrix}$ and the schedule vector $s^t = \begin{bmatrix} 1 & 2 \end{bmatrix}$ are used. Therefore, the pipeline period $\alpha = 2$.

2. LPGS scheme: In this scheme, one block is mapped to the fixed size array. The

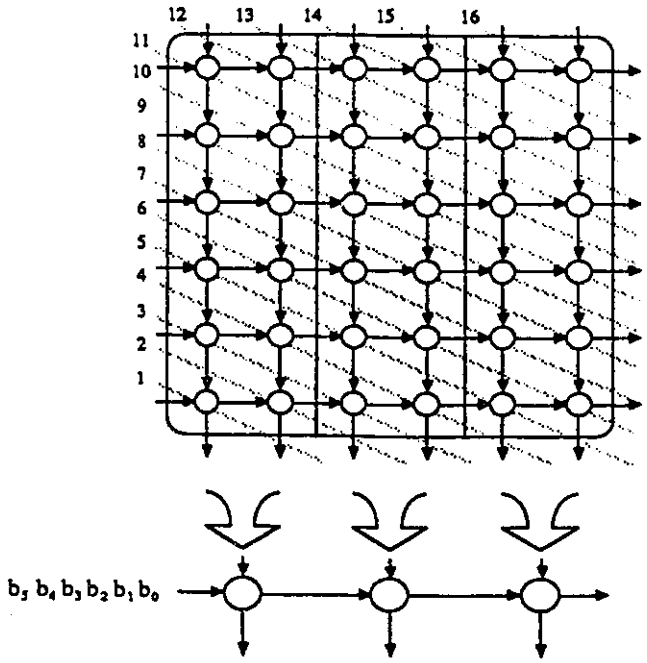



Figure 2.15: LSGP type of array for matrix-vector multiplication.

computations within one block are performed partially in parallel. However, after the computations within one block are completed, the next block can then be started, i.e., globally sequential. Therefore, the intermediate data, i.e., the elements of the vector, has to be stored in a FIFO (first in first out) buffer and be used for the next block. The LPGS type of array for matrix-vector multiplication is given in Figure 2.16. Notice that since $d^t = \begin{bmatrix} 0 & 1 \end{bmatrix}$ and $s^t = \begin{bmatrix} 1 & 1 \end{bmatrix}$, the pipeline period $\alpha$ is equal to 1.

### 2.3.4 Improvement of Utilization Efficiency

By mapping a recursive algorithm onto an array, the pipeline period of the resulting array, $\alpha$, can be greater than 1, as shown in previous examples. When $\alpha$ is greater than 1, the input data rate is interleaved with o samples and slowed down by a factor $\alpha$; therefore, the efficiency of the array is only $\frac{1}{\alpha}$. In the case of recursive

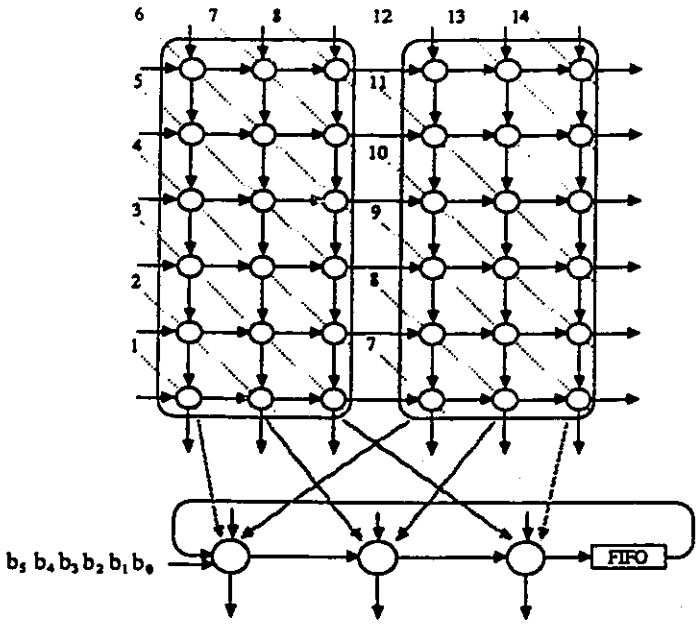



Figure 2.16: LPGS type of array for matrix-vector multiplication.

filtering, the next inputs to the array are dependent on the previous outputs of the array. When we map the algorithm onto the array, the pipeline period is equal to 1. However, due to the computation delay of the array, the inputs are not yet available. Hence, the input data will be interleaved by zeroes. This also results in lower utilization of the array.

In order to improve the utilization efficiency of the array, one simple way is to interleave several inputs. If we have $\alpha$ similar problems and they are independent from each other, we can interleave their inputs and outputs to keep the array busy all the time. The problem for this method is that it is not always available to have several independent problems that need the same processing. An example of multiplying four input sequences $a(n), b(n), c(n)$ and $d(n)$ with a matrix $M$ by pipeline interleaving method is shown in Figure 2.17.

The second way is to use the multiprojection technique, i.e., we may project the DG a number of times to obtain a smaller size and highly efficient array [46]. Since

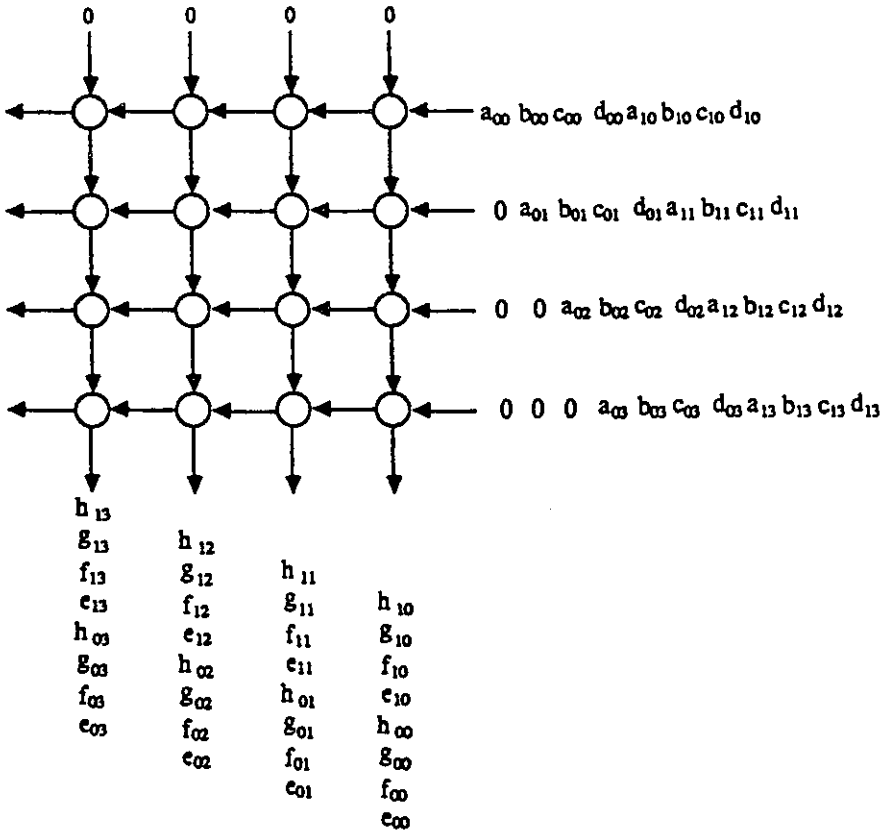



Figure 2.17: Pipeline interleaving for improving the utilization of the array.

the pipeline period of the original array is $\alpha$, PEs only work every $\alpha$ clock times. Therefore, we can map $\alpha$ PEs onto one PE. This PE will fulfil the computations for all $\alpha$ PEs sequentially. In this way the utilization efficiency is improved $\alpha$ times. The recursive state-space filtering results in successive matrix-vector multiplications. Therefore, it can be viewed as a matrix-matrix multiplication. A two-dimensional systolic array can be used to implement this filter. Because there are zeroes in the input data flow, the efficiency is $\frac{1}{\alpha}$, where $\alpha$ is equal to the size of the array. By projecting this array, a one-dimensional array is obtained with efficiency of 100% as shown in Figure 2.18.

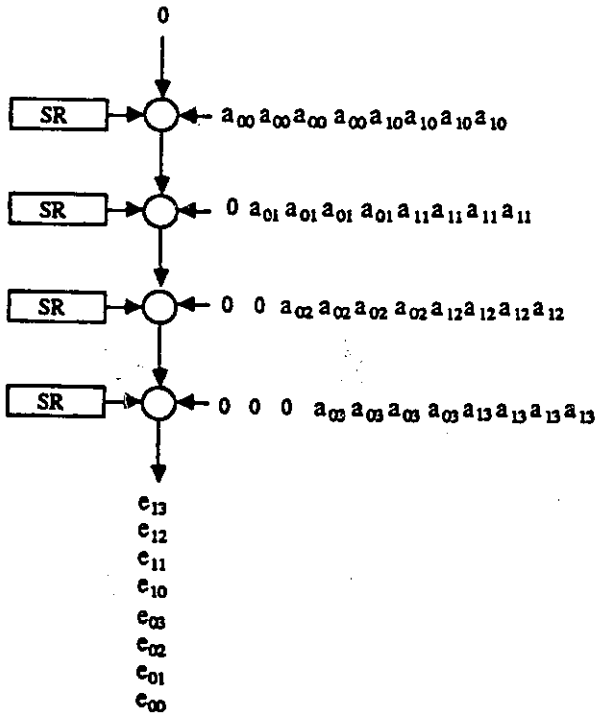



Figure 2.18: Resulted array by multiprojection of DG.

The third is to develop a new algorithm or modify the original algorithm for the problem so that the algorithm has maximum parallelism. It is clear an algorithm will be favored if it expresses a higher parallelism. For the example of recursive state-space filtering, the next input state is the current output state of the array.

No matter what kind of array is used, the next iteration can not start until the current iteration is finished. If we use transform technique to modify the algorithm, the next input state vector is not the current output vector but the one which is already generated. Therefore, the efficiency of the array can be improved. Other methods to decorrelate the input data can also be used to exploit the maximum parallelism in the algorithms.

## 2.4 Conclusion

In this chapter, the direct mapping methodology for mapping the recursive algorithms onto array architectures has been described, and several optimization methods have been developed. The systematic transformation from DG to systolic arrays directly was shown to simplify the system design. As to optimal systolic design, it depends on the performance criterion and is application oriented. In VLSI design, the overall architecture should be as simple, regular and modular as possible. For DSP applications, the throughput rate and numerical performance are two dominant factors to judge the design of an array. In practice, array size, I/O arrangement, utilization efficiency and reliability are major considerations. To design systolic array architectures for particular applications, optimization considerations are generally needed. In the following chapters, dedicated array architectures for recursive state-space filtering, adaptive state-space filtering and Kalman filtering are investigated. The mapping procedure and the optimization methods will be applied to those designs.

# Chapter 3

# VLSI Architectures for Real-time One-Dimensional Recursive Filtering

## 3.1 Introduction

The minimization of finite-word-length effects in digital filters is of considerable practical significance since it leads to implementations with optimal signal-to-noise ratio. State-space structures for digital filters have been described by Mullis and Roberts [47,48,49] Hwang [50,51,52], Jackson, Lindgren and Kim [53,54] and others [55,56]. They have shown that the state-space technique is not only a useful tool in the formulation and analysis of linear system problems, but state-space structures can also possess very low roundoff noise and coefficient sensitivity, and that these structures can be designed to eliminate the possibility of overflow oscillations and limit cycles [57,58,59]. In particular, given any design with state matrices ($\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}$), a state transformation can be applied to produce a new state-space realization, such as normal structure, which is free of overflow oscillations and has

minimum roundoff noise also. The most important advantage of state-space realization is the ability to produce improved nonlinear performance. In general, this is obtained at the expense of an increase in the number of multipliers. Jackson in [60] and Roberts in [61,62] describe the state-space technique in detail.

Recently, many researchers have implemented recursive filters by using VLSI technology. For the direct form IIR filtering, H.T. Kung has done some pioneering work with his concept of systolic arrays and proposed the systolic arrays for filtering in [16]. S.Y. Kung presented another systolic array architecture for ARMA filtering and used a wavefront concept to develop a more efficient structure [63,2]. A high throughput systolic implementation of the second order recursive filter was shown in [64]. Rao and Kailath gave a model identification approach to VLSI filter design [65]. In [66], high speed stored product recursive digital filters were suggested by Dubois and Steenaart. For the state-space digital filtering, some VLSI implementations have been shown in [67,68,69].

In this chapter, several new systolic architectures for 1-D recursive filters based on state-space matrix realization will be introduced. In section 3.2, the digital state-space filtering technique is briefly reviewed. A very simple systolic array for direct form state-space realizations is given in section 3.3. Then, an advanced state calculation array implementation is described in section 3.4. In this structure, the state matrices are converted to new ones so that advanced state vectors can be calculated and fed back to the array. The throughput and efficiency can be increased $N+1$ times with the hardware increased twice, as compared to the direct implementation of the transformed matrix multiplication. Another implementation can be found by realizing a cascade of second-order state-space filters. This is shown in section 3.5. In a cascade of second-order structures, each transformed filter is implemented by parallel processing elements. This structure results in a higher throughput rate and requires less hardware as compared to the direct implementation of the transformed matrix multiplication. All the structures proposed here are 100% efficient and can

work at a high sampling frequency.

## 3.2 Digital State-Space Filtering

### 3.2.1 State Variable Description

There are important problems involving digital filter structures that are quite naturally formulated using the state variable description. With this matrix description, and linear algebra, we can apply powerful analytical methods. Furthermore, from the computational point of view, the state variable approach is invaluable.

For a linear shift-invariant system (LSI), there are several different representations. An important class of LSI systems is the class described by a transfer function $H(z)$ that is a rational function of $z$, i.e.,

$$H(z) = \frac{Y(z)}{U(z)} = \frac{\Sigma_{k=0}^{N} b_k z^{-k}}{1 - \Sigma_{k=1}^{N} a_k z^{-k}} \tag{3.1}$$

Equation 3.1 is corresponding to the following linear difference equation (or recursive equation) in the time domain

$$y(n) = \Sigma_{k=0}^{N} b_k u(n-k) + \Sigma_{k=1}^{N} a_k y(n-k) \tag{3.2}$$

This recursive digital filter can also be described by the familiar state equations

$$X(n+1) = \mathbf{A}X(n) + \mathbf{B}u(n) \tag{3.3}$$

$$y(n) = \mathbf{C}X(n) + \mathbf{D}u(n) \tag{3.4}$$

where, $X(n), X(n+1)$ are state variables and defined as the outputs of delay operators in Figure 3.1. $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are state variable parameter matrices with $N \times N$, $N \times 1, 1 \times N$, and 1, respectively for a single-input, single-output $N$th order IIR filter. By direct substitution, the solution of the state equation (3.3) for a stable filter is:

$$X(n) = \Sigma_{l=1}^{\infty} \mathbf{A}^{l-1} \mathbf{B} u(n-1) \tag{3.5}$$

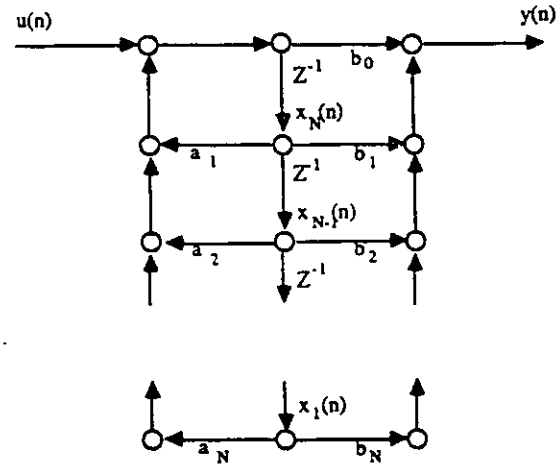



Figure 3.1: Signal flowgraph of 1-D IIR filter.

To obtain the transfer function from the state variable description, we simply use $z$-transform and eliminate the internal variables $X(n)$. The associated transfer function matrix is given by

$$H(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \tag{3.6}$$

and the eigenvalues of $\mathbf{A}$ are the roots of the polynomial

$$\hat{a}(z) = det(z\mathbf{I} - \mathbf{A}) \tag{3.7}$$

which (by Cramer's Rule) is the denominator of the transfer function $H(z)$ as expressed in (3.6). In other words, the poles of transfer function $H(z)$ are the eigenvalues of the matrix $\mathbf{A}$.

Recalling that the input and output sequences are related by the convolution equation

$$y(n) = h(n) * u(n) = \Sigma_{l=-\infty}^{\infty} h(l)u(n-l). \tag{3.8}$$

Since

$$\begin{aligned} y(n) &= \mathbf{C}X(n) + \mathbf{D}u(n) \\ &= \Sigma_{l=1}^{\infty} \mathbf{C}\mathbf{A}^{l-1}\mathbf{B}u(n-l) + \mathbf{D}u(n) \end{aligned} \tag{3.9}$$

we have the unit-impulse response in terms of the state matrices as the following:

$$h(n) = \begin{cases} 0 & n < 0 \\ \mathbf{D} & n = 0 \\ \mathbf{CA}^{n-1}\mathbf{B} & n > 0 \end{cases} \tag{3.10}$$

The state equation $X(n)$ represents $N$ internal variables. We can change the coordinate system for $X(n)$ without changing the external description (the unit-impulse response). Such a transformation will, however, change the internal description $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$. Therefore, different realizations of the same transfer function can be obtained by nonsingular transformations of the system state vector $X(n)$. Thus, if $\mathbf{T}$ is a real invertible $N \times N$ matrix, and we let

$$X'(n) = \mathbf{T}X(n), \tag{3.11}$$

then the filter

$$X'(n+1) = \mathbf{A}'X'(n) + \mathbf{B}'u(n) \tag{3.12}$$

$$y(n) = \mathbf{C}'X'(n) + \mathbf{D}u(n) \tag{3.13}$$

where

$$\mathbf{A}' = \mathbf{TAT}^{-1} \tag{3.14}$$

$$\mathbf{B}' = \mathbf{TB} \tag{3.15}$$

$$\mathbf{C}' = \mathbf{CT}^{-1} \tag{3.16}$$

will give the same linear transfer function as the original filter.

### 3.2.2 Sufficient Conditions for Optimal Filters

In the design of fixed point digital filters, there are three primary finite-world-length effects:

1. Roundoff noise caused by the rounding of products within the filter realization;

2. Overflow oscillations and limit cycles due to nonlinear quantization effects;

3. Changes in the input/output description of the filter due to approximating the real parameters with a finite wordlength binary representation.

Of the three finite-word-length effects, roundoff noise is the most important. The third effect, loosely called coefficient sensitivity, is a deterministic effect and can be bounded in some sense by the roundoff noise. Thus, we will discuss the conditions for minimum roundoff noise and freedom of overflow oscillations here.

- "Overflow-stable" Realizations

  The term "overflow stable" denotes realizations that are guaranteed to be free from overflow oscillations due to the overflow nonlinearity. Let length or norm of a vector be defined as

$$\|X\| = (X^t X)^{1/2} = (\Sigma_{i=1}^{N} x_i^2)^{1/2}. \tag{3.17}$$

  With this definition of length, the "gain" or "amplifying power" or norm of $\mathbf{A}$ can be defined as

$$\|\mathbf{A}\| = \max_{X \neq 0} \frac{\|\mathbf{A}X\|}{\|X\|} = \max_{X \neq 0} \left(\frac{X^t \mathbf{A}^t \mathbf{A} X}{X^t X}\right)^{1/2} \tag{3.18}$$

  In other words, $\|\mathbf{A}\|$ is the maximum increase in the length of the vector. If $\|\mathbf{A}\| < 1$, then all vectors $X$ decrease in length under multiplication by $\mathbf{A}$. Thus, if we can find a structure with a system matrix $\mathbf{A}$ such that $\|\mathbf{A}\| < 1$, then the zero-input overflow oscillations cannot occur.

  In general, the class of state variable descriptions for which there exists a positive diagonal matrix $\hat{\mathbf{D}}$ such that $\mathbf{Q} = \hat{\mathbf{D}} - \mathbf{A}^t \hat{\mathbf{D}} \mathbf{A}$ is a positive definite matrix is overflow-stable. In the case of second order filters, this result reduces to the following for a two state system. If the eigenvalues of $\mathbf{A}$ have magnitude less than unity, $\hat{\mathbf{D}}$ exists for which $\mathbf{Q}$ is positive definite if and only if

$$a_{12} a_{21} \geq 0 \tag{3.19}$$

or if

$$a_{12}a_{21} \leq 0, \quad then \quad |a_{11} - a_{22}| + det|\mathbf{A}| \leq 1. \tag{3.20}$$

where $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$. Among filters that satisfy this condition (equation 3.19 or 3.20) are normal filters, minimum roundoff noise filters, ladder of lattice filters and a multitude of others.

- Minimum Roundoff Noise Realizations

When overflows in internal registers are negligible, roundoff noise is the dominant component of output error in a digital filter. Referring to Figure 3.2, let the $N \times 1$ vector $f(n)$ be the unit-impulse response from the input to the variables, point f in the figure, $g(n)$ be the unit-impulse response from the point g to the output. The state covariance matrices $\mathbf{K}$ and $\mathbf{W}$ are then given by:

$$\mathbf{K} = \Sigma_{n=1}^{\infty} f(n)f^t(n) = \Sigma_{n=0}^{\infty}(\mathbf{A}^n\mathbf{B})(\mathbf{A}^n\mathbf{B})^t = \mathbf{AKA}^t + \mathbf{BB}^t \tag{3.21}$$

$$\mathbf{W} = \Sigma_{n=1}^{\infty} g(n)g^t(n) = \Sigma_{n=0}^{\infty}(\mathbf{CA}^n)^t(\mathbf{CA}^n) = \mathbf{A}^t\mathbf{WA} + \mathbf{C}^t\mathbf{C}. \tag{3.22}$$

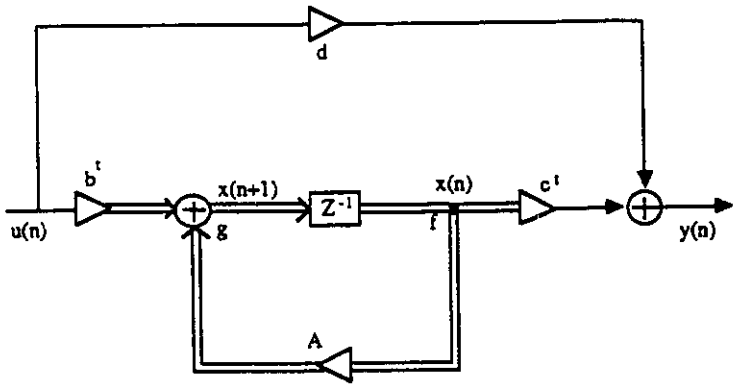



Figure 3.2: The filter diagram in state-space form.

Then for scaled digital filters, the output roundoff noise of the filter, $\sigma_n^2$, in terms of diagonal elements of $\mathbf{K}$ and $\mathbf{W}$ is

$$\sigma_n^2 = \frac{\delta^2 q^2}{12} \Sigma_{i=1}^{N} w_{ii} k_{ii}. \tag{3.23}$$

The necessary and sufficient conditions for a minimum noise filter are given by the diagonal elements of $\mathbf{K}$ and $\mathbf{W}$ [62,53]:

$$k_{ii} w_{ii} = k_{jj} w_{jj} \quad i, j = 1, 2, ...N. \tag{3.24}$$

In the case of second order filters, let $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} c_1 & c_2 \end{bmatrix}$, the conditions for minimum noise are equivalent to:

$$a_{11} = a_{22} \tag{3.25}$$

$$b_1 c_1 = b_2 c_2 \tag{3.26}$$

$$the\ filter\ \ is\ \ l_2\ \ scaled. \tag{3.27}$$

Notice that $b_1$ and $b_2$ here represent the elements of the vector $\mathbf{B}$. They are not the cofficients in (3.1) and (3.2).

### 3.2.3 State Transformation and Optimization

In order to minimize the effects of the finite-word-length, given any design with state matrices $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$, linear state transformations then must be applied to the network. Since the state-space structure with $\mathbf{A}$ matrix of the norm (or coupled) form $\mathbf{A} = \begin{bmatrix} a_1 & a_2 \\ -a_2 & a_1 \end{bmatrix}$ was found to be a low-sensitivity recursive structure, and this structure also satisfies the minimum roundoff noise constraint (3.24), the network can be normalized first. The transformation procedure can be described as follows [62].

1. Using a nonsingular transformation matrix $\mathbf{T}_1 = \begin{bmatrix} -\beta & 0 \\ -\alpha & 1 \end{bmatrix}$, where $\alpha, \beta$ are the real and imaginary parts of eigenvalues of $\mathbf{A}$, the corresponding normal form is obtained:

$$\mathbf{A}'_1 = \mathbf{T}_1\mathbf{A}\mathbf{T}_1^{-1} = \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \tag{3.28}$$

$$\mathbf{B}'_1 = \mathbf{T}_1\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3.29}$$

$$\mathbf{C}'_1 = \mathbf{C}\mathbf{T}_1^{-1} = r\begin{bmatrix} cos\phi & sin\phi \end{bmatrix} \tag{3.30}$$

where $r^2 = c_1^2 + c_2^2, \phi = tan^{-1}(\frac{c_2}{c_1})$. Notice that the condition (3.24) is achieved now.

2. In order to satisfy (3.25), a second transformation is needed. Let $\mathbf{T}_2 = R(\frac{\phi}{2}) = \begin{bmatrix} cos(\frac{\phi}{2}) & sin(\frac{\phi}{2}) \\ -sin(\frac{\phi}{2}) & cos(\frac{\phi}{2}) \end{bmatrix}$, the network will be transformed to:

$$\mathbf{A}'_2 = \mathbf{T}_2\mathbf{A}'_1\mathbf{T}_2^{-1} = \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \tag{3.31}$$

$$\mathbf{B}'_2 = \mathbf{T}_2\mathbf{B}'_1 = \begin{bmatrix} sin(\frac{\phi}{2}) \\ cos(\frac{\phi}{2}) \end{bmatrix} \tag{3.32}$$

$$\mathbf{C}'_2 = \mathbf{C}'_1\mathbf{T}_2^{-1} = r\begin{bmatrix} cos(\frac{\phi}{2}) & sin(\frac{\phi}{2}) \end{bmatrix} \tag{3.33}$$

Hence, the condition (3.25) is achieved.

3. Finally, the network has to be $l_2$ scaled. To scale $(\mathbf{A}'_2, \mathbf{B}'_2, \mathbf{C}'_2)$, the covariance matrix $\mathbf{K}_2$ has to be calculated. If $\mathbf{K}'_2 = \begin{bmatrix} d_1^2 & k_{12} \\ k_{21} & d_2^2 \end{bmatrix}$, then, let $\mathbf{T}_3 = \frac{1}{\delta}\begin{bmatrix} d_1^{-1} & 0 \\ 0 & d_2^{-1} \end{bmatrix}$; $\delta$ is a subjectively chosen value which will effect the

scaling degree.

$$\mathbf{A'}_3 = \mathbf{T}_3\mathbf{A'}_2\mathbf{T}_3^{-1} = \begin{bmatrix} \alpha & -\beta d_2 d_1^{-1} \\ \beta d_1 d_2^{-1} & \alpha \end{bmatrix} \tag{3.34}$$

$$\mathbf{B'}_3 = \mathbf{T}_3\mathbf{B'}_2 = \frac{1}{\delta}\begin{bmatrix} d_1^{-1} sin(\frac{\phi}{2}) \\ d_2^{-1} cos(\frac{\phi}{2}) \end{bmatrix} \tag{3.35}$$

$$\mathbf{C'}_3 = \mathbf{C'}_2\mathbf{T}_3^{-1} = \delta r \begin{bmatrix} d_1 cos(\frac{\phi}{2}) & d_2 sin(\frac{\phi}{2}) \end{bmatrix} \tag{3.36}$$

The filter $(\mathbf{A'}_3, \mathbf{B'}_3, \mathbf{C'}_3)$ meets all the conditions of optimality and thus is the minimum roundoff noise filter.

This transformation can be applied to any given $N$th-order IIR filter. However, after the transformation, the number of non-zero elements in the state matrices is increased considerably. In general, a transformed Nth-order state-space structure would require $(N+1)^2$ processing elements to implement the system matrices.

## 3.3 Direct Form State-Space Implementation

For a linear system, state variable descriptions are derived from signal flow graph (SFG) descriptions. Recalling that a recursive digital filter can be described by SFG (Figure 3.1), and, for convenience, the state update equation and the output equation can be rewritten as following:

$$X(n+1) = \mathbf{A}X(n) + \mathbf{B}u(n) \tag{3.37}$$

$$y(n) = \mathbf{C}X(n) + \mathbf{D}u(n) \tag{3.38}$$

where, $X(n), X(n+1)$ are state variables and defined at the output of the delay operators in Figure 3.1. $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are the system matrices with $N \times N, N \times 1, 1 \times N, 1 \times 1$ respectively for a single input, single output $N$th order filter.

Computation of these two equations using a square systolic array [70,69] is shown in Figure 3.3. This is an array of $(N+1) \times (N+1)$ cells, each of which contains one

multiplier and one adder. $N+1$ clock times after input $u(n)$ enters the array, the output $y(n)$ comes out at the bottom of the array and then $X(n+1)$ is available. A problem is that due to the delay of $N+1$ clocks most of the array remains idle waiting for the output $X(n+1)$ to become available. Thus this array can process only one sample every $N+1$ clocks and the efficiency is $\frac{1}{N+1}$. More efficient and higher speed structures have to be investigated.

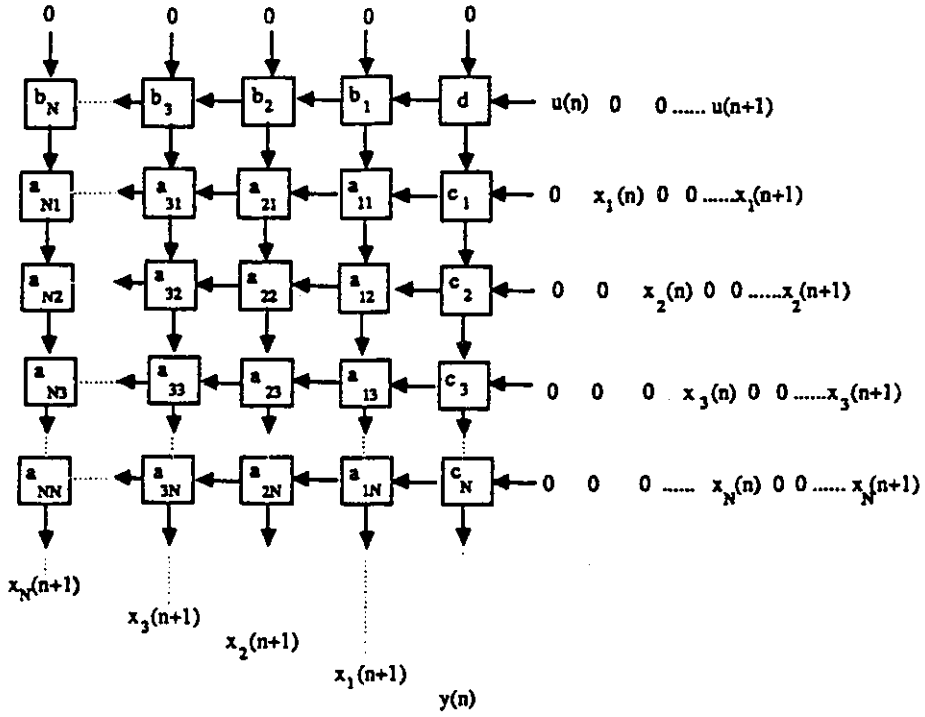



Figure 3.3: Two-dimensional systolic array for Nth-order 1-D IIR state-space recursive filters (a direct implementation).

The implementation of 1-D IIR filters discussed in this section is obtained starting from the state-space representation [71]. By examining the relationship between the current state and the next state in the Figure 3.1, the corresponding state-space

representation is:

$$\begin{bmatrix} x_1(n+1) \\ x_2(n+1) \\ \dots \\ \dots \\ x_N(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_N & a_{N-1} & \dots & \dots & a_1 \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \\ \dots \\ \dots \\ x_N(n) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dots \\ \dots \\ 1 \end{bmatrix} u(n) \quad (3.39)$$

$$y(n) = \begin{bmatrix} c_1 & c_2 & \dots & \dots & c_N \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \\ \dots \\ \dots \\ x_N(n) \end{bmatrix} + b_0 u(n) \quad (3.40)$$

where $c_1 = b_N + b_0 a_N, c_2 = b_{N-1} + b_0 a_{N-1}, \dots, c_N = b_1 + b_0 a_1$, and $a_i, b_i$ are the coefficients in (3.1) and (3.2).

As can be seen from (3.3), there are many zeroes and ones in **A** and **B**. Actually, only $x_N(n+1)$ has to be computed, and the other variable components are obtained by shifting the current components, i.e., $x_1(n+1) = x_2(n)$ etc. Therefore, a simplified array requiring only $2(N+1)$ processor cells to implement the state-space direct form realization [71,72,73] is shown in Figure 3.4.

The state variables $x_1(1), x_2(1), \dots x_N(1)$ are initialized as zeroes. Due to one clock delay for each cell, the input state vector components have to be time-skewed. And since $x_1(2) = x_2(1) = 0, x_1(3) = x_2(2) = x_3(1) = 0, \dots,$

$$x_1(N) = x_2(N-1) = x_3(N-2) = \dots = x_N(1) = 0 \quad (3.41)$$

Thus, during the first $N$ clocks, the components of the state vector are all zeroes. After a delay of $N+1$ clocks, the $x_N(2) = u(1)$ starts appearing at the output and the output signal $y(1) = a_0 u(1)$ follows. As $x_i(n) = x_{i+1}(n-1)$, and because of input data skew, the generated component $x_N(2)$ must be broadcasted to all the entries of the array. Then, $x_N(3)$ is obtained and fed back to the array.

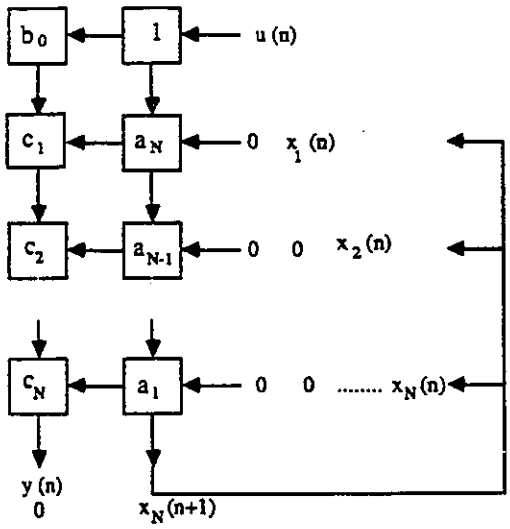



Figure 3.4: Systolic array implementation for direct form state-space recursive filtering.

This systolic array is, in fact, for the direct form IIR filtering. It has the same hardware complexity as the canonical form digital filter, but the realization compares favorably with other realizations in systolic array form. Another important feature of this structure is that after an initial delay of $N+1$ clocks, the throughput rate is one sample per clock time $T_c$, and the structure is then 100% efficient. It shows that the state-space technique is a very useful tool in analysis and realization of linear systems.

## 3.4 Advanced State Calculation Array

The main goal of using state-space technique is to realize high performance fixed point IIR digital filters. The optimal realization can be obtained by similarity transformations. However, after transformation, the number of non-zero elements in the state matrices is increased considerably. As mentioned above, an Nth-order state-space structure requires $(N+1)^2$ processing elements to implement it. The

speed of this implementation will be limited by a delay of $N+1$ clocks, the efficiency is $\frac{1}{N+1}$ [69,74,71]. For improving the efficiency of the square array and speeding up the processing, an implementation of the block-state filter was given in [68]. In that structure, the state matrix A was converted to triangular or "quasi-triangular" form via an unitary or orthogonal similarity transformation. Different types of processing cells were used and multiple input/output lines were needed. Here we give another realization by using the advanced state calculation technique [72], since the problem of the square array is that the next input state vector is not available until the output state vector is computed. If an input vector of size $m$ ($m$ consecutive input samples) can be entered at once, and the state matrices are changed correspondingly, the state vectors can be updated in advance. For the causal system, the initial states are zero. As a result, the array doesn't have to wait for the output state vectors. It can be described as follows.

When $m = 1$, corresponding to the single-input case, the system can be described by :

$$\begin{bmatrix} X(n+1) \\ y(n) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} X(n) \\ u(n) \end{bmatrix} \tag{3.42}$$

Then for multiple inputs, $m = N$, by using recursive substitution, we have:

$$\begin{aligned} \begin{bmatrix} X(n+N) \\ y(n+N-1) \end{bmatrix} &= \begin{bmatrix} \mathbf{A}^N & \mathbf{A}^{N-1}\mathbf{B} & \dots & \mathbf{AB} & \mathbf{B} \\ \mathbf{CA}^{N-1} & \mathbf{CA}^{N-2}\mathbf{B} & \dots & \mathbf{CB} & \mathbf{D} \end{bmatrix} \begin{bmatrix} X(n) \\ u(n) \\ u(n+1) \\ \dots \\ u(n+N-1) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}_m & \mathbf{B}_m \\ \mathbf{C}_m & \mathbf{D}_m \end{bmatrix} \begin{bmatrix} X(n) \\ U(n) \end{bmatrix} \end{aligned} \tag{3.43}$$

where

$$\mathbf{A}_m = \mathbf{A}^N, \tag{3.44}$$

$$\mathrm{B}_m = \left[ \begin{array}{cccc} \mathrm{A}^{N-1}\mathrm{B} & \ldots & \mathrm{AB} & \mathrm{B} \end{array} \right], \tag{3.45}$$

$$\mathrm{C}_m = \mathrm{CA}^{N-1}, \tag{3.46}$$

$$\mathrm{D}_m = \left[ \begin{array}{cccc} \mathrm{CA}^{N-2}\mathrm{B} & \ldots & \mathrm{CB} & d \end{array} \right], \tag{3.47}$$

and $U^t(n) = \left[ \begin{array}{cccc} u(n) & u(n+1) & \ldots & u(n+N-1) \end{array} \right]$.

Since matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ have dimensions of $N \times N, N \times 1, 1 \times N$ and $1 \times 1$, the new state matrix is $(N+1) \times 2N$. Therefore, if $N$ input data can access the array at once with one state vector $X(n)$, instead of computing the next state $X(n+1)$, $X(n+N)$ can be calculated. As a result, when $X(n+N)$ is required to enter the array, it is available as output of the array at the same time.

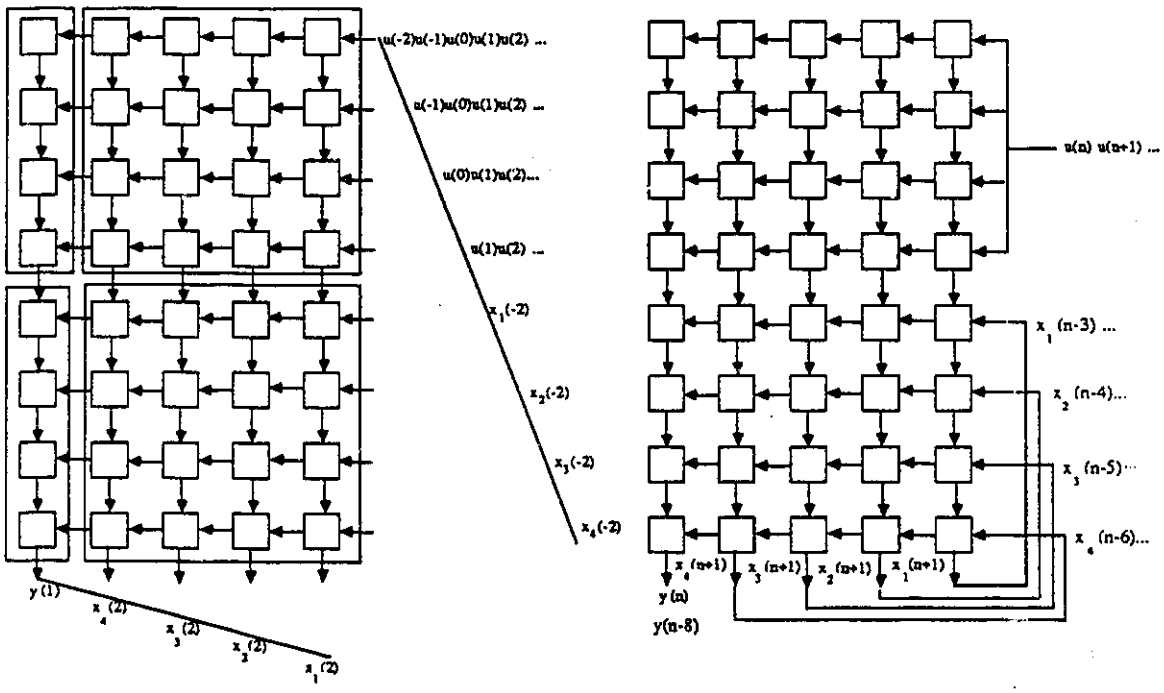



Figure 3.5: The advanced state calculation array for a 4th-order state-space recursive filtering. (a) principle diagram; (b) input/output design.

Figure 3.5 shows the implementation of a 4th-order state-space filter. This systolic array contains $5 \times 8$ processing elements. each of which is still a simple multiplier/adder. There are four parts in this structure for calculating matrix-vector multiplications $\mathbf{A}_m X(n), \mathbf{B}_m U(n)$ and vector-vector multiplications $\mathbf{C}_m X(n), \mathbf{D}_m U(n)$ respectively. Assume $X(n) = 0$ for $n \leq 1$ and $u(n) = 0$ for $n \leq 0$. Then $X(n) = \begin{bmatrix} x_1(n) \\ x_2(n) \\ x_3(n) \\ x_4(n) \end{bmatrix}$ and $U(n) = \begin{bmatrix} u(n) \\ u(n+1) \\ u(n+2) \\ u(n+3) \end{bmatrix}$. Initially, $X(-2)$ and $U(-2)$ enter the array for computing $X(2), y(1)$. Note that all the inputs are zero except $u(1)$. One clock later, $X(-1)$ and $U(-1)$ are ready to enter the array and so on. After a delay of 4 clocks, $X(2)$ is available at the bottom of the array and it will be fed back to the corresponding entries. Then $X(3)$ follows. Consequently, after the array starts to work, one new sample data enters the array and one filtered data sample leaves the array at every clock. From Figure 3.5, we see that due to input data skew and the data vector updating scheme, the input data to all the entries at any clock period are the same. Thus, instead of an input data vector, only one sample of data is required to be broadcast to all the entries of the upper part array. No input data buffer is required. The input state vectors should be zeroes for the causal filters and the outputs of the array can be initialized to zeroes before the calculated state value appears. Thus, the output of the state vector can be connected directly to the corresponding input entry as shown in Figure 3.5(b). Neither delays nor other circuits are needed.

This structure can improve the performance of the IIR filters by loading the transformed matrices into the array. The throughput rate and efficiency of the array is raised from $\frac{1}{N+1}$ to 1. Furthermore, the array is both modular and simple in design. Only the simplest processing element (multiplier/adder) is used, and there is only one input and one output required. It is a typical systolic configuration. The drawback of this structure is an increase in the number of processing elements.

According to the design principles described in chapter 3, this structure is realizable and interesting.

## 3.5 Cascade Form of Second-Order Structures

As indicated above, the number of multipliers is increased considerably by transformations. For improving efficiency and raising speed, more processing elements have to be used in the structure described in section 3.3. One way of reducing the number of multipliers in minimum roundoff noise realizations is to apply the theory to small-order subfilters. Since the cascade form is more robust under coefficient quantization than the parallel form and therefore is more commonly used[75], a cascade form of second-order structures will be discussed in this section [72]. If we decompose $H(z)$ into $r$ second-order filters, $r = \frac{N}{2}$, each section can be transformed to achieve minimum noise structure as described in section 3.2. However, a connection of them is not globally optimal, since downstream filters are not correctly scaled. Another method, so-called block-optimal, can be used. Let $(\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i, \mathbf{D}_i)$ be the matrix of ith section, then the overall structure will be described by $(\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i, \mathbf{D}_i, i = 1, 2, \ldots, \frac{N}{2})$. A cascade of two second-order sections has the state variable description

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & 0 \\ \mathbf{B}_2\mathbf{C}_1 & \mathbf{A}_2 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2\mathbf{D}_1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} \mathbf{D}_2\mathbf{C}_1 & \mathbf{C}_2 \end{bmatrix} \quad \mathbf{D} = \mathbf{D}_1\mathbf{D}_2 \quad (3.48)$$

The transformation steps are given below [62]:

1. Find $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ for the overall structure.
2. Calculate $(\mathbf{K}, \mathbf{W})$ for the overall structure.
3. Extract $2 \times 2$ diagonal blocks from $(\mathbf{K}, \mathbf{W})$ and optimize each $(\mathbf{K}_i, \mathbf{W}_i)$.
4. Apply the individual optimizing transformation to the original second order sections $(\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i, \mathbf{D}_i)$.

As a result, each section is not a normal structure, but the overall system is the minimum noise structure with fewer multipliers. The second-order structure can be implemented by $3 \times 3$ PEs as shown in Figure 3.6. Therefore, $9 \times \frac{N}{2} = 4.5N$ are required for an Nth-order filter as opposed to $(N+1)^2$ PEs for an optimum Nth-order state-space structure. From Figure 3.6, we see that as in the Nth-order case, the array has to wait for the output state vectors. Due to the 2 clock delay, the efficiency is 50% and every input/output sample must be interleaved with zeroes.

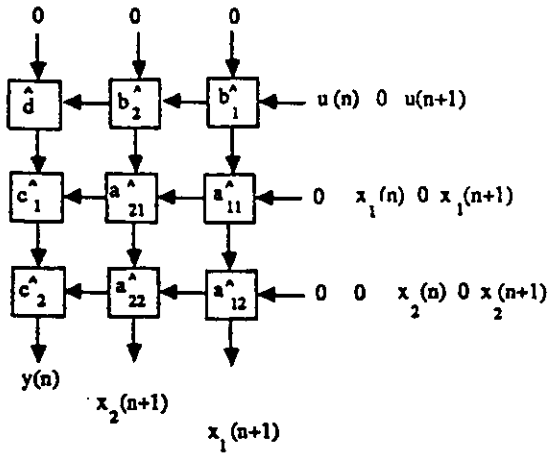



Figure 3.6: A square array for second-order state-space recursive filters.

The state equations show that $y(n), x_1(n+1), x_2(n+1)$ are independent. They can be calculated simultaneously. Each of them is generated by three multiplication/additions and those operations can be executed concurrently. It is reasonable to design the structure by exploiting the parallelism in this algorithm. It is shown in Figure 3.7. The input data is broadcast to three multipliers. At the beginning of the process, the state vector is initialized to zero. The transformed coefficients are loaded into processors. When the input data enters, 9 multipliers start to work at the same time. After one clock delay $x_1(n+1), x_2(n+1)$ and $y(n)$ appear from the three adders respectively. The output $y(n)$ is transmitted to the next section for further processing. The new state components will be fed back to the entries

of the current section for the next recursion. The clock period is equal to one multiplication time and two addition times, resulting in high speed operation with an efficiency of 100%.

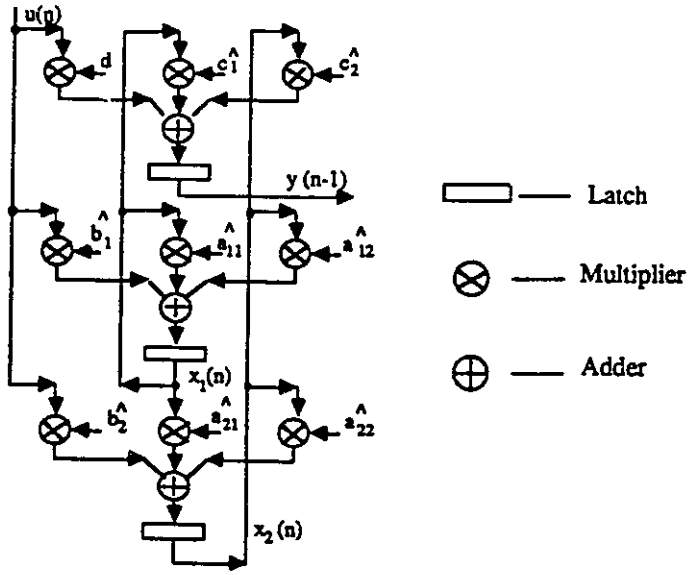



Figure 3.7: A high speed structure for second-order state-space filters.

The overall structure for an Nth-order state-space recursive filter, the cascade form of second-order structures is shown in Figure 3.8. For a single input, single output filter, after $N/2$ clock delay, the first output sample is obtained. Then at each clock, one sample can be processed. The communication is local. The modular design is naturally suitable to this structure.

## 3.6 Comparison and Conclusion

In [16] and [63,2], H.T. Kung and S.Y. Kung presented two systolic array architectures for the implementations of 1-D IIR direct form filters. Both are 50% efficient, and each sample must be interleaved with a blank. If the state matrices are implemented by the square array proposed by Urquhart and Wood, the efficiency is

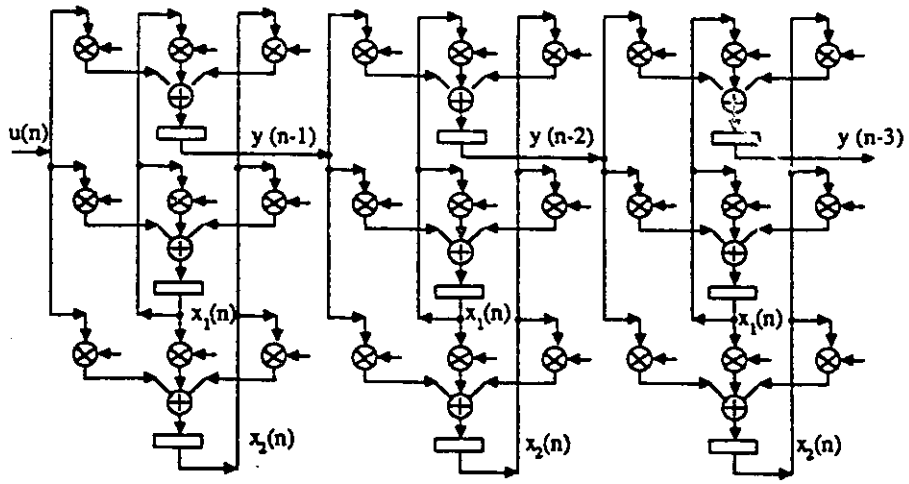



Figure 3.8: The cascade form of second-order state-space structures (3 sections).

$\frac{1}{N+1}$. In this chapter, we have presented three array structures for 1-D IIR filters based on state-space technique. In terms of throughput rate, efficiency, latency, numerical property and hardware complexity, the comparisons among the various architectures are given in Table 3.1, where the throughput rate is defined as the number of samples processed per clock time.

| Architecture | Multipliers | Latency | Efficiency | Throughput rate | Numerical property |
|---|---|---|---|---|---|
| H.T. Kung's | $2(N+1)$ | 1 | 50% | 0.5 | unimproved |
| S.Y. Kung's | $2(N+1)$ | 1 | 50% | 0.5 | unimproved |
| Square state space array | $(N+1)^2$ | $N+1$ | $\frac{1}{N+1}$ | $\frac{1}{N+1}$ | improved |
| Direct state space form | $2(N+1)$ | $N+1$ | 100% | 1 | unimproved |
| Advanced state calculation | $2N(N+1)$ | $N+1$ | 100% | 1 | improved |
| Cascade form | $4.5N$ | $\frac{N}{2}$ | 100% | 1 | improved |

Table 3.1: Comparison of different systolic arrays for 1-D IIR filters.

It is shown that using the state-space approach to analyze the recursive filters,

a very simple systolic array can be obtained. By state transformation and normalization, the numerical properties of IIR filters can be improved and the number of multipliers is increased. In order to realize real-time processing, the advanced state calculation array and the cascade form of second-order structures are given by exploiting the parallelism in the algorithm. Both of them are 100% efficient. They can work at high speed and provide high performance. The structure is less expensive. Three structures described here can be used for real-time digital filtering.

# Chapter 4

# High Speed Architectures for Two-Dimensional State-Space Recursive Filtering

## 4.1 Introduction

The numerical performance in a finite-word-length implementation of a given 2-D transfer function is very dependent on the structure of its realization in a manner similar to the 1-D case. Recently, the interest in synthesis and design of 2-D state-space fixed-point digital filter structures with minimum roundoff noise has increased. In [76,77], Kawamata and Higuchi analyzed the variance of roundoff noise of 2-D separable denominator digital filters in the state-space formulation and proposed the synthesis method of 2-D separable denominator digital filters with minimum roundoff noise and no overflow oscillations. In [78,79], the roundoff noise problems of general 2-D digital filters were studied. The general descriptions of the noise matrix and the covariance matrix were derived. A general synthesis procedure was developed which leads to an optimal local state-space 2-D digital filter realization. Lin Kawamata and Higuchi in [80] presented an optimal design-oriented stability

condition for 2-D state-space digital filters. A general formulation for optimal design of 2-D recursive digital filters is obtained. In [81] El-Agizi and Fahmy gave a criterion which sufficiently guarantees the absence of overflow oscillations in 2-D digital filters in the state space. It has been shown that as in the 1-D case, 2-D state space structures can realize IIR digital filters with minimum roundoff noise, absence of overflow oscillations and low sensitivity. Therefore, state-space digital filters have considerable significance in practice. However, this technique leads to structures having more multipliers than other general realizations.

In the past few years, real-time image processing using 2-D digital filters has become a rapidly growing field in the industrial and biomedical environments, as the need for the fast processing of large amounts of data became evident. The term "real-time image processing" means that images are processed at a speed such that the data rate of the processed images is the same as that of the input images. With a display size of $512 \times 512$ pixels, real-time implies a serial data stream at the rate of 7.86M pixels/s or one pixel every 127 ns. It is impossible to achieve such a processing speed with sequential computing. During recent years, several high speed architectures using parallel processing techniques for 2-D digital direct form filtering have been proposed. In [82,83], Ty and Venetsanopoulos proposed the distributed arithmetic architecture, minimum cycle time filter architecture, multiprocessor element architecture, and VLSI architecture by decomposition. A video rate architecture for a fully recursive filter was presented in [84]. In [85,86], Parhi and Messerschmitt gave concurrent architectures for recursive digital filtering. A systolic array for real-time state-space filtering is given in [69,74]. All these architectures are capable of processing a $512 \times 512$ in real-time or near real-time.

Parallel processing as an efficient form of information processing emphasizes the exploitation of concurrent computations in the process, and these can be executed at different levels. In order to decrease the processing time for each pixel, special computational structures, such as an array of processors, are required. The pixels

are pipelined into processors sequentially. We refer to this approach as "pixel-level speed-up" or "local speed-up". On the other hand, architectures with pipelining and a high degree of parallelism are able to process many pixels simultaneously. The time for processing each pixel may not very short. But, the total processing time of whole images can be decreased in a global sense. We refer to this as "frame-level speed-up". Furthermore, when several processing arrays are available, multiple frames can be processed in parallel. We refer to this method as "sequence-level speed-up". The "frame-level speed-up" and the "sequence-level speed-up" are called "global speed-up" in this thesis. Suppose that $L$ images of size $M \times N$ are to be processed. $l^2$ clock periods per pixel are required for image state-space filtering, where $l$ depends on the order of the filter. The total processing time, which is hereafter referred to as the time complexity, will be $O(L \times M \times N \times l^2)$. When a "local speed-up" technique is used, employing $l^2$ or more elements, the time complexity can be $O(L \times M \times N)$. But in many cases, $M, N$ or $L$ are much larger than $l$. Only with a "lobal speed-up" technique, can the processing time be greatly decreased.

In this chapter, VLSI architectures for high speed 2-D state-space recursive filters are proposed. In section 4.2, state-space realizations for 2-D recursive systems are presented. Subsequently, the problems relating to VLSI implementation are examined. Then, the "local speed-up" implementations are shown in section 4.3. They process each pixel in one multiplication/addition clock time. A systolic array for direct form state-space representation, a square systolic array for transformed state-space filters and an advanced state update architecture are presented sequentially. With the diagonal picture scanning, the throughput rates of the direct form array and the advanced state update array are 1 pixel/clock period and the efficiency is 100%. The time complexity is $O(L \times M \times N)$. In section 4.4, based on concurrency in 2-D recursive filters, a "global speedup", frame-level, systolic architecture is developed.

First, by using DG (dependence graph) mapping method [2] and examining

the relationship between input and output, a column processor is designed. Then, a number of column processors are connected to form a two-dimensional array structure, each working on one column of samples, the whole structure processing multiple columns of samples concurrently. The time complexity for this structure is $O(L \times M \times l)$ by using $N \times l$ PEs (processing elements). A comparison between existing structures and the ones developed in this chapter is made in section 4.5.

## 4.2 2-D State-Space Models

In the last decade, Attasi [87,88], Fornasini-Marchesini [89,90], Givone-Roesser [91,92], Mitra *et al.* [93], Chan [94], Bisiacco [95], and Kurek [96], proposed different 2-D state-space models. State-space models have excellent prospects in image processing for several reasons. First, they are general models, more general than any other 2-D models which are used extensively in image processing in recent years. Second, these models which are in spatial-domain are suitable and are natural vehicles for several modern system theoretical applications. Third, in the state space models one can provide physical meaning to state variables as state images. Finally, state variable approach is easy to extend to 3-D or M-D system and to other signal processing problems.

### 4.2.1 Roesser's model and Its State Diagram

Kung *et al.* [97] have presented a comparison between some different state-space models. It is shown that Roesser's model includes Fornasini-Marchesini model and Attasi's model. It is one of the general 2-D state-space models. Roesser's model was first introduced by Givone and Roesser in [91]. Since then most work in recent years is due to [92], represented by Roesser, the model is known as Roesser's model. It is also called local state-space (LSS) model. For a single-input single-output 2-D

digital filter, the general form of Roesser's model is given by:

$$X_{11}(m,n) = \mathbf{A}X(m,n) + \mathbf{B}u(m,n) \quad (4.1)$$

$$y(m,n) = \mathbf{C}X(m,n) + \mathbf{D}u(m,n) \quad (4.2)$$

where $X(m,n)$ is the current state vector and $X_{11}(m,n)$ denotes the next state vector. When written in more detailed form it becomes:

$$\begin{bmatrix} X^h(m+1,n) \\ X^v(m,n+1) \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X^h(m,n) \\ X^v(m,n) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u(m,n) \quad (4.3)$$

$$y(m,n) = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} X^h(m,n) \\ X^v(m,n) \end{bmatrix} + \mathbf{D}u(m,n) \quad (4.4)$$

where $m = 0,1,...,M-1, n = 0,1,...,N-1$. $u(m,n)$ and $y(m,n)$ are the input and the output, respectively. $X^h(m,n)$ is the horizontal state-space vector component of dimension $m_h \times 1$, and $X^v(m,n)$ is the vertical component of dimension $n_v \times 1$. $\mathbf{A}$ is a $(m_h + n_v) \times (m_h + n_v)$ matrix. $n$ and $m$ are the numbers of delay operators in the horizontal direction and in the vertical direction respectively of a general 2-D recursive filter described by:

$$y(m,n) = \Sigma_{i=0}^{N_h}\Sigma_{j=0}^{N_v} b_{ij}u(m-i,n-j) + \Sigma_{i=0}^{N_h}\Sigma_{j=0}^{N_v}{}_{(i,j)\neq(0,0)} a_{ij}y(m-i,n-j). \quad (4.5)$$

To aid in understanding the relations between input images, state images and output images, the state diagram of Roesser's model is given in Figure 4.1. From this diagram, it is clear that the horizontal state component conveys all the past information for the current pixel along the same line. The vertical state component conveys all the past information for this pixel up to the previous line. A basic and important difference between 1-D and 2-D state-space realizations is that the 2-D state is local rather than global. That is, in 2-D case, the state $X(m,n)$ does not contain enough information to find $y(p,q)$ for all $(p,q) > (m,n)$ even when $u(p,q)$ is given. From $X(m,n)$, we can find $y(m,n)$ as well as $X^h(m+1,n)$ and $X^v(m,n+1)$. Then, to find $y(m+1,n)$, we need $X^h(m+1,n)$ and $X^v(m+1,n)$ which can not be found from $X(m,n)$. Thus, $X(m,n)$ is said to be a state only in the local sense.

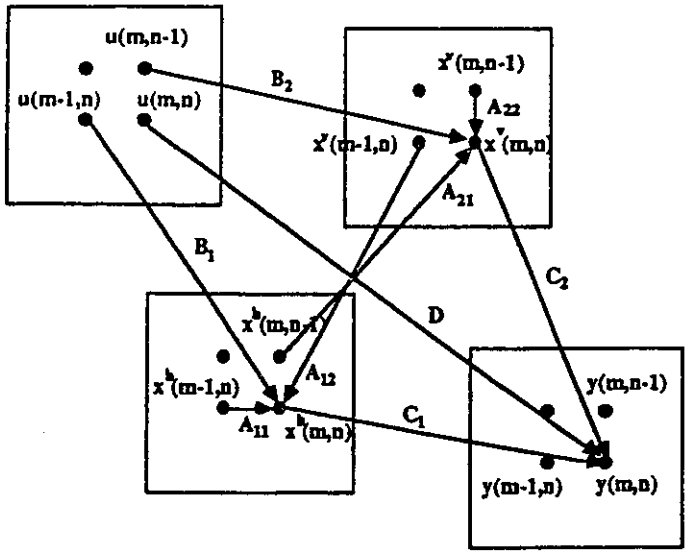



Figure 4.1: The state diagram of Roesser's model.

### 4.2.2 Transfer Function and Transformation

The application of the 2-D Z-transformation to equations 4.1 and 4.2 yields the transfer function:

$$H(z) = \mathbf{C}[\Gamma(z) - \mathbf{A}]^{-1}\mathbf{B} + \mathbf{D} \tag{4.6}$$

where $\Gamma(z) = \begin{bmatrix} z_1 I_m & 0 \\ 0 & z_2 I_n \end{bmatrix}$, and $I_m$ and $I_n$ are identity matrices with dimensions of $m_h$ and $n_v$, respectively.

To obtain a filter with minimum roundoff noise, low coefficient sensitivity and free of overflow oscillations, the coordinate transformations have to be used to obtain other realizations. For a given design with state matrices$(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$, the state matrix transformations are used to produce "well-designed" state-space structures, which minimize the finite-word-length effects [60,62].

Let $X'(m,n) = \mathbf{T}X(m,n)$, then $\mathbf{A}' = \mathbf{TAT}^{-1}, \mathbf{B}' = \mathbf{TB}, \mathbf{C}' = \mathbf{CT}^{-1}, \mathbf{D}' = \mathbf{D}$. To guarantee that the linear system response (i.e., the transfer function) does not

change, a transformation matrix T for 2-D systems has to have the form [98]:

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_1 & 0 \\ 0 & \mathbf{T}_2 \end{bmatrix} \tag{4.7}$$

where $\mathbf{T}_1$ and $\mathbf{T}_2$ are $m_h \times m_h$ and $2n_v \times 2n_v$ matrices, respectively. The transformations can be selected so as to improve one or several aspects of a filter's nonlinear behavior, such as eliminating the limit cycle oscillations only, or scaling the network for no overflow as well as minimizing roundoff noise and sensitivity. For example, given a reachable and observable realization $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ satisfying the stability condition, there exists a 2-D similarity transformation $\mathbf{T}$, such that realization $(\mathbf{TAT}^{-1}, \mathbf{TB}, \mathbf{CT}^{-1}, \mathbf{D})$ minimizes the output noise power subject to dynamic range constraint. The procedure for computing the desired transformation matrix $\mathbf{T}$ is given in [79].

### 4.2.3 Implementation Problem

It is noticed that the matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ obtained directly from the signal flow graph are highly sparse. In general, after transformations, the resultant matrices will not be sparse. Accordingly, a realization with minimum roundoff noise will require more multipliers, at most $l^2(l = m_h + n_v + 1)$multipliers. Therefore, the improvement in the finite-word-length effects is obtained at the expense of an increase in hardware complexity. Let $\mathbf{W} = \begin{bmatrix} w_{ij} \end{bmatrix} = \begin{bmatrix} \mathbf{A}' & \mathbf{B}' \\ \mathbf{C}' & \mathbf{D}' \end{bmatrix}$ represents the system matrix of the local state-space realization, the dimension of $W$ is $l \times l$.

In order to speed up the state space filtering for real time applications, a straightforward implementation is given in [69]. It is a two-dimensional systolic array. According to the state equations 4.3 and 4.4, for each given $X(m,n)$ and $u(m,n)$, we have to find the horizontal state vector, $X^h(m+1,n)$, and the vertical state vector, $X^v(m,n+1)$, as well as computing the output $y(m,n)$. It is shown that due to the recursive property, the pixel at $(m+1,n)$ or $(m,n+1)$ can not be processed until

$X^h(m+1,n)$ or $X^v(m,n+1)$ are available. The problem is that with the regular scanning method, the input data have to be interleaved by zeroes. Therefore, during the filtering process, most of the processing elements in the array will be idle. The efficiency is $\frac{1}{l}$. The throughput is one pixel per $l$ clock periods. In [69], a diagonal scanning method is proposed and a modified systolic array is suggested. With the diagonal recursion, the array is 100% efficient and processes one pixel within every clock time. In this chapter, we will give some alternative array architectures.

## 4.3 Local Speed-up Architectures

### 4.3.1 Direct State-Space Form Array

Consider the case for $m = n = 2$:

$$H(z_1, z_2) = \frac{\sum_{i=0}^{2}\sum_{j=0}^{2} b_{ij} z_1^{-i} z_2^{-j}}{1 - \sum_{i=0}^{2}\sum_{j=0_{(i,j)\neq(0,0)}}^{2} a_{ij} z_1^{-i} z_2^{-j}}. \tag{4.8}$$

The LSS realization for this filter can be obtained by mapping its signal flow-graph to the state-space model. As indicated in the Figure 4.2, the state variables are chosen as the outputs of the spatial delay elements.

The local state-space realization is then described by:

$$\begin{bmatrix} x_1^h(m+1,n) \\ x_2^h(m+1,n) \\ x_1^v(m,n+1) \\ x_2^v(m,n+1) \\ x_3^v(m,n+1) \\ x_4^v(m,n+1) \\ y(m,n) \end{bmatrix} = \begin{bmatrix} a_{10} & a_{20} & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \tilde{a}_{11} & \tilde{a}_{21} & a_{01} & 1 & 0 & 0 & a_{01} \\ \tilde{a}_{12} & \tilde{a}_{22} & a_{02} & 0 & 0 & 0 & a_{02} \\ \tilde{b}_{11} & \tilde{b}_{21} & b_{01} & 0 & 0 & 1 & b_{01} \\ \tilde{b}_{21} & \tilde{b}_{22} & b_{02} & 0 & 0 & 0 & b_{02} \\ \tilde{b}_{10} & \tilde{b}_{20} & b_{00} & 0 & 1 & 0 & b_{00} \end{bmatrix} \begin{bmatrix} x_1^h(m,n) \\ x_2^h(m,n) \\ x_1^v(m,n) \\ x_2^v(m,n) \\ x_3^v(m,n) \\ x_4^v(m,n) \\ u(m,n) \end{bmatrix} \tag{4.9}$$

where $\tilde{a}_{ij} = a_{ij} + a_{i0}a_{0j}$ and $\tilde{b}_{ij} = b_{ij} + a_{i0}b_{0j}$. Examining the system matrix in 4.9, we can see that:

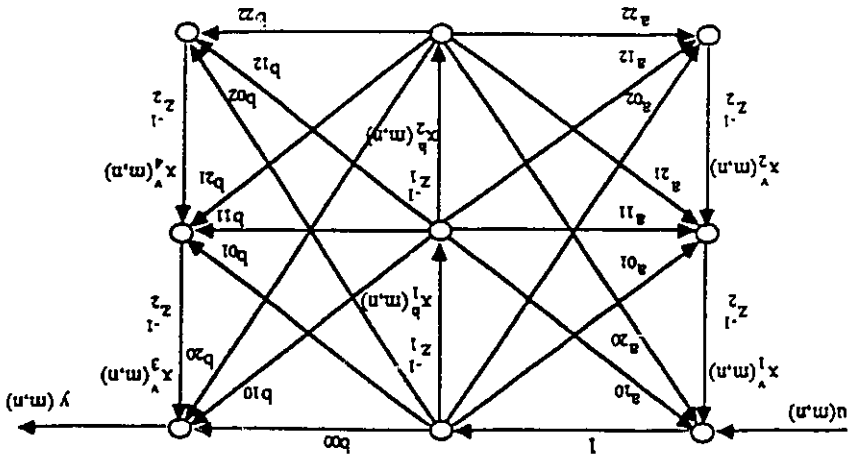


Figure 4.2: Signal flowgraph and state variables for 2-D second-order IIR filtering.

1. The third and seventh column in the $7 \times 7$ matrix 4.9 have identical elements. Hence, the seventh column in the matrix and the seventh entry in the input vector, $u(m,n)$, can be removed with replacing $x_1^v(m,n)$ by $[x_1^v(m,n) + u(m,n)]$;

2. $x_2^h(m+1,n)$ does not need to be calculated since it is always equal to $x_1^h(m,n)$;

3. $x_3^v(m,n)$ and $x_4^v(m,n)$ do not appear in the computation of the first four elements of the output vector. They are only needed for $y(m,n)$.

Taking these remarks into consideration, an array with $3 \times 6$ PEs instead of $7 \times 7$ PEs can be used for the case of the direct form realization [71]. As shown in Figure 4.3, the number of multipliers is equal to that in the canonical form. Since $x_2^h(m+1,n) = x_1^h(m,n)$, a delay is used at the output of the $x_1^h(m+1,n)$ to obtain $x_2^h(m+1,n)$. A few additional adders are used to complete the computation of $y(m,n)$ and $x_1^v(m,n+1)$. Several spatial delay cells are also needed for timing adjustment.

Similarly, due to the recursive property, the pixel at $(m+1,n)$ or $(m,n+1)$ can not be processed until $X^h(m+1,n)$ or $X^v(m,n+1)$ are available. With regular

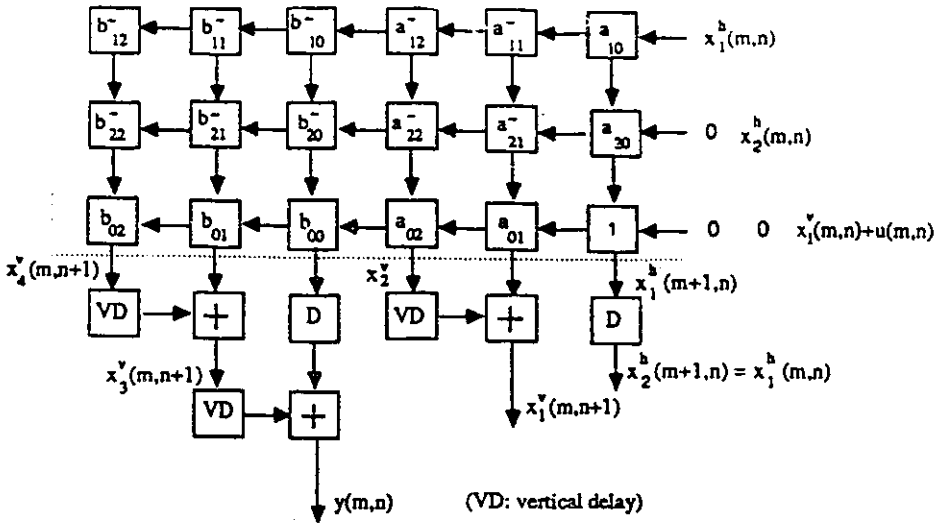



Figure 4.3: A direct state-space form array for second-order IIR filtering.

line by line scanning, the input data have to be interleaved by zeroes. Therefore, the throughput rate of this implementation is one pixel/3 clock times. But when the array works on successive input pixels on a diagonal, the throughput rate is 1 pixel/clock time. This array will be 100% efficient after the initial set up period and will work on successive horizontal strips each of size $N \times 3$. As a result, the time complexity is $O(L \times M \times N)$ clock times for processing $L$ images.

### 4.3.2 Advanced State Update Architecture

The reason for using the diagonal scanning method in implementation [74] is that due to delays of the array, the pixel at $(m+1, n)$ or $(m, n+1)$ can not be processed until $X^h(m+1, n)$ or $X^v(m, n+1)$ are available. One solution to this problem is to use the advanced state update technique developed for the 1-D case [72]. This method is a variation of the basic block processing approach. For block processing the input pixels are grouped into 1-D or 2-D blocks. By modifying the system matrices accordingly, the new array architecture can output a block of data every

clock time. The block processing technique can increase the throughput rate. However not only is the hardware increased considerably, but also multiple input and multiple output have to be arranged. The main idea of the advanced state update architecture is to use a single input to obtain the advanced state vector and a single output data with the transformed matrices and corresponding architecture.

Recalling that the state equations 4.3 and 4.4, corresponding to the single-input single-output case, can be written as:

$$\begin{bmatrix} X^h(m+1,n) \\ X^v(m,n+1) \\ y(m,n) \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ C_1 & C_2 & D \end{bmatrix} \begin{bmatrix} X^h(m,n) \\ X^v(m,n) \\ u(m,n) \end{bmatrix} \tag{4.10}$$

Then, if we use blocks of length $l$ in the horizontal direction, the advanced state vector and the output data can be calculated as follows:

$$\begin{bmatrix} X^h(m+l,n) \\ X^v(m+l-1,n+1) \\ y(m+l-1,n) \end{bmatrix} = \begin{bmatrix} A_{hh} & A_{hv} & B_{hu} \\ A_{vh} & A_{vv} & B_{vu} \\ C_h & C_v & D_u \end{bmatrix} \begin{bmatrix} X^h(m,n) \\ X_l^v(m,n) \\ u_l(m,n) \end{bmatrix} \tag{4.11}$$

where,

$$A_{hh} = A_{11}^l, \tag{4.12}$$

$$A_{hv} = \begin{pmatrix} A_{11}^{l-1}A_{12} & \cdots & A_{11}A_{12} & A_{12} \end{pmatrix}, \tag{4.13}$$

$$B_{hu} = \begin{pmatrix} A_{11}^{l-1}b_1 & \cdots & A_{11}b_1 & b_1 \end{pmatrix}, \tag{4.14}$$

$$A_{vh} = A_{21}A_{11}^{l-1}, \tag{4.15}$$

$$A_{vv} = \begin{pmatrix} A_{21}A_{11}^{l-2}A_{12} & \cdots & A_{21}A_{12} & A_{22} \end{pmatrix}, \tag{4.16}$$

$$B_{vu} = \begin{pmatrix} A_{21}A_{11}^{l-2}b_1 & \cdots & A_{21}b_1 & b_2 \end{pmatrix}, \tag{4.17}$$

$$C_h = c_1A_{11}^{l-1}, \tag{4.18}$$

$$C_v = \begin{pmatrix} c_1A_{11}^{l-2}A_{12} & \cdots & c_1A_{12} & c_2 \end{pmatrix}, \tag{4.19}$$

$$D_u = \begin{pmatrix} c_1A_{11}^{l-2}b_1 & \cdots & c_1b_1 & d \end{pmatrix}. \tag{4.20}$$

and

$$X_l^v(m,n) = \left( X^v(m,n) \;\; X^v(m+1,n) \;\; \cdots \;\; X^v(i+l-1,j) \right)^t, \quad (4.21)$$

$$u_l(m,n) = \left( u(m,n) \;\; u(m+1,n) \;\; \cdots \;\; u(i+l-1,j) \right)^t. \quad (4.22)$$

A 2-D causal filter described by its LSS realization has initial conditions:

$$X^v(m,0) = 0, \quad 0 \le m \le M-1 \quad (4.23)$$

$$X^h(0,n) = 0. \quad 0 \le n \le N-1 \quad (4.24)$$

It is reasonable to assume that for $m < 0, X^v(m,0) = 0, X^h(m,0) = 0$ and $u(m,0) = 0$. Therefore, we can use $X^h(-l+1,0), X_l^v(-l+1,0), u_l(-l+1,0)$ to calculate the state vector $X^h(1,0), X^v(0,1)$ and $y(0,0)$. Similarly, the state vector $X^h(2,0), X^v(1,1)$ and $y(1,0)$ are obtained from $X^h(-l+2,0), X_l^v(-l+2,0)$ and $u_l(-l+2,0)$ and so on. This filter can implemented by a two-dimensional systolic array. After an initial delay, the output data and updated state vector are produced during each clock period. The vertical vector $X^v(m,n)$ is stored in the vertical delay register and the horizontal vector is fed back to the entry of the array directly for calculation of $y(m+l-1,n), X^h(m+l,n)$ and $X^v(l+k-1,n+1)$.

According to the above analysis, an input data vector $u_l(m,n)$ and $l$ vertical state vectors $X_l^v(m,n)$ are required for the advanced state update scheme. In fact, only one input data and one vertical state vector are needed at each clock time. Due to the nature of systolic array, the input data vector and input state vectors have to be time skewed. On the other hand, the next input data vector is the time shifted version of the current one. Hence, the input data to all the entries at any clock time are the same. Instead of an input data vector, only one input data is needed. It is broadcast to all corresponding entries. The vertical state vectors can be treated in the same manner. Since the systolic array should be initialized to zero, the output of the vertical state vector can be connected to the corresponding entries directly. Neither delays nor other circuits are needed.

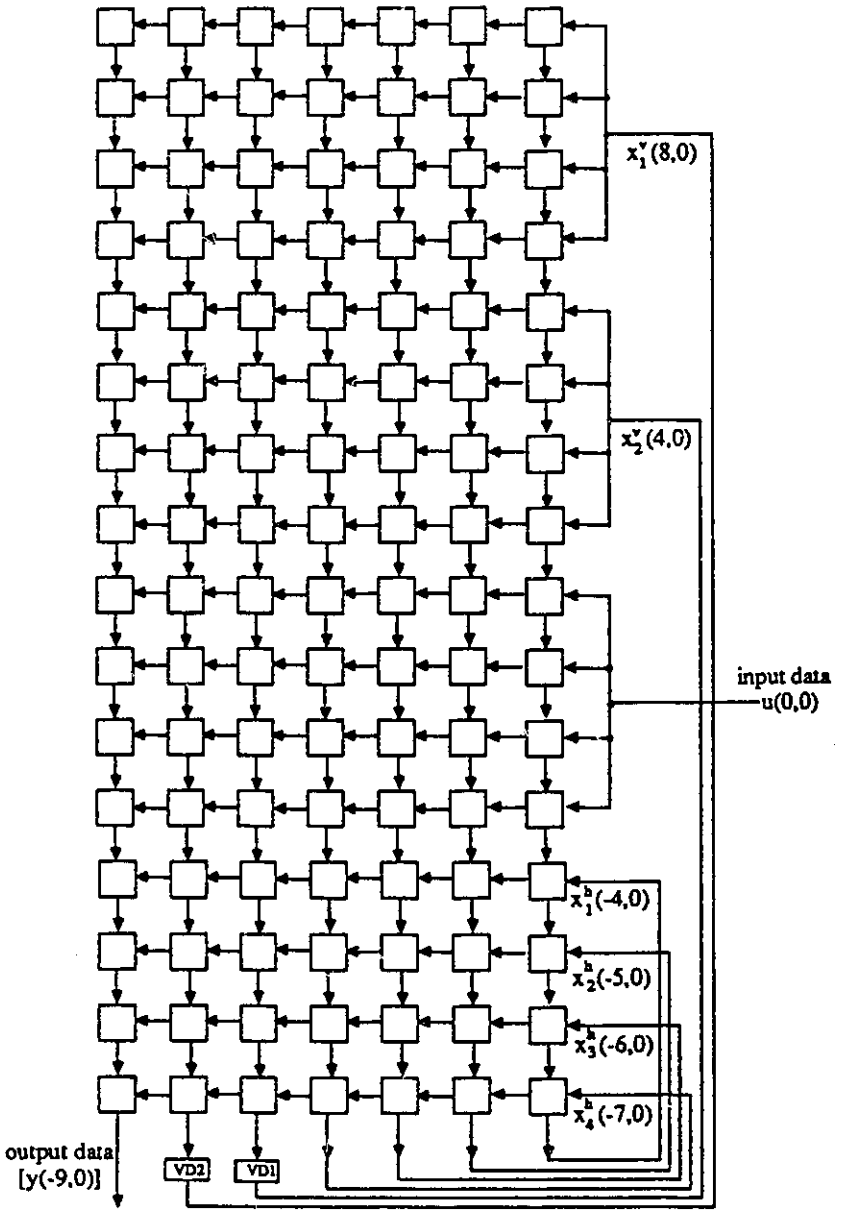



Figure 4.4: The advanced state update architecture for 2-D second-order state-space recursive filtering.

Figure 4.4 shows the implementation of a second-order state-space filter. This systolic array contains $7 \times 16$ processing elements, each of which is a simple multiplier/adder. With an increase in processing elements, this architecture has the following properties:

1. The throughput rate is related to the sample rate, i.e., one pixel/1 clock time, and the efficiency is 100%;
2. The regular line-by-line scanning method is used;
3. The processing elements are of the simplest form (multiplier/adder);
4. The connections between all the PEs are simple and regular;
5. Only one input and one output are required for whole architecture. Therefore, if we can design a large VLSI chip for this architecture, there is no problem with input and output pins.

## 4.4 Global Speed-Up Architecture

It is well known that two-dimensional recursive digital filter algorithms contain inherent concurrency, which can be exploited for pipelining and/or parallelism. Note that in the first-quadrant 2-D state-space filtering, the samples along the diagonal lines can be processed independently of each other, since these computations are not constrained by any precedence relations (see Figure 4.5). For example, the samples $y(4,0), y(3,1), y(2,2), y(1,3)$ and $y(0,4)$ form a set of independent computations, and can be processed in parallel. The systolic array implementation of state-space filtering given in [74] uses this concurrency property. However, it processes images pixel- by-pixel along diagonal direction sequentially. Furthermore, the structure is not easy to be expanded for even high speed processing.

In this section, a high speed structure is suggested by exploiting the spatial parallelism and using many processors concurrently. This structure is referred to as

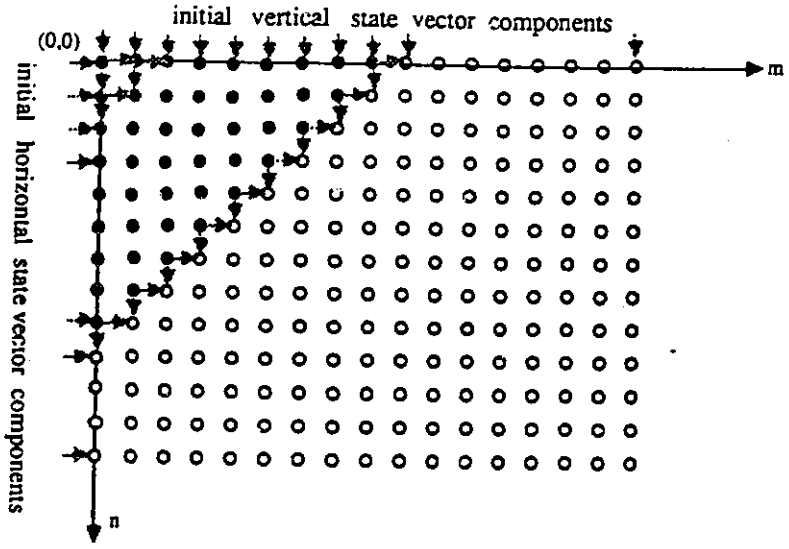



Figure 4.5: Spatial concurrency in 2-D systems.

global speed-up (frame-level) structure [99,100,101,102,103]. As shown in section 4.2, the state equations result in matrix-vector operations for each pixel. According to the mapping method given in [2], we can map the dependence graph (DG) of a state-space filter shown in Figure 4.6(a) (second-order) to different Signal Flow Graphs (SFGs). If we use a projection vector $d = (1 \quad 0)^t$ and systolic schedule, $s = (1 \quad 1)^t$, for this dependence graph, the corresponding systolic array is obtained in Figure 4.6(b).

Since in this application input data length is very long, the state vector for each pixel is different and dependent on the previous processing. The components of each vector can not stay in the cells of this column array for whole filtering process. They remain at the cells of the array for one pixel processing time only.

Referring the data dependency and time schedule, the basic cell and connections between cells of the array are given in Figure 4.7. Since it is a column array and is suitable for processing images along the vertical direction, it is referred to as a column processor. This processor contains 7 processing cells, each consisting of one multiplier, one adder, one shift register, one latch, and two D flip-flops. In this

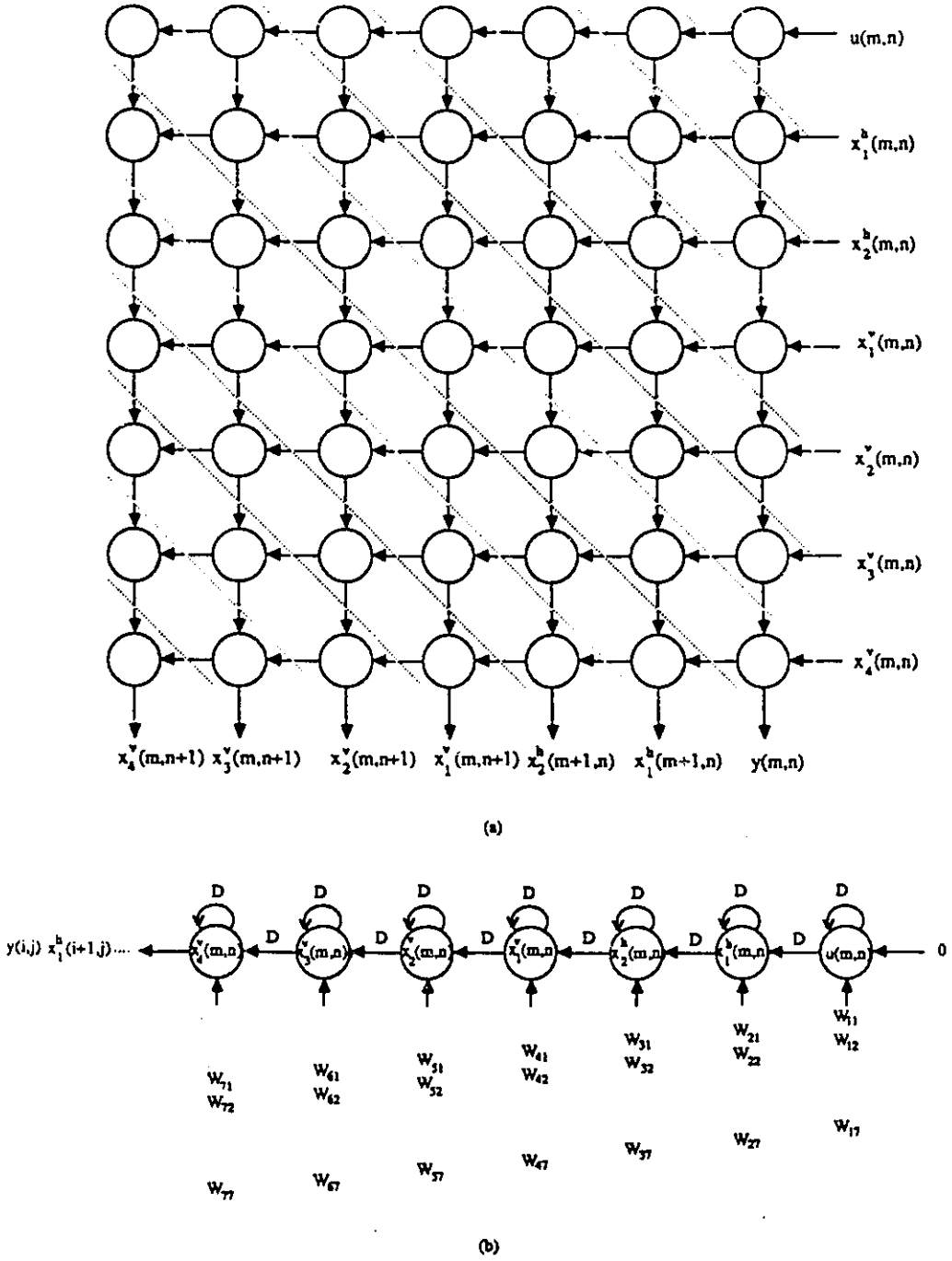



Figure 4.6: Mapping 2-D state-space filtering algorithm onto a systolic array.

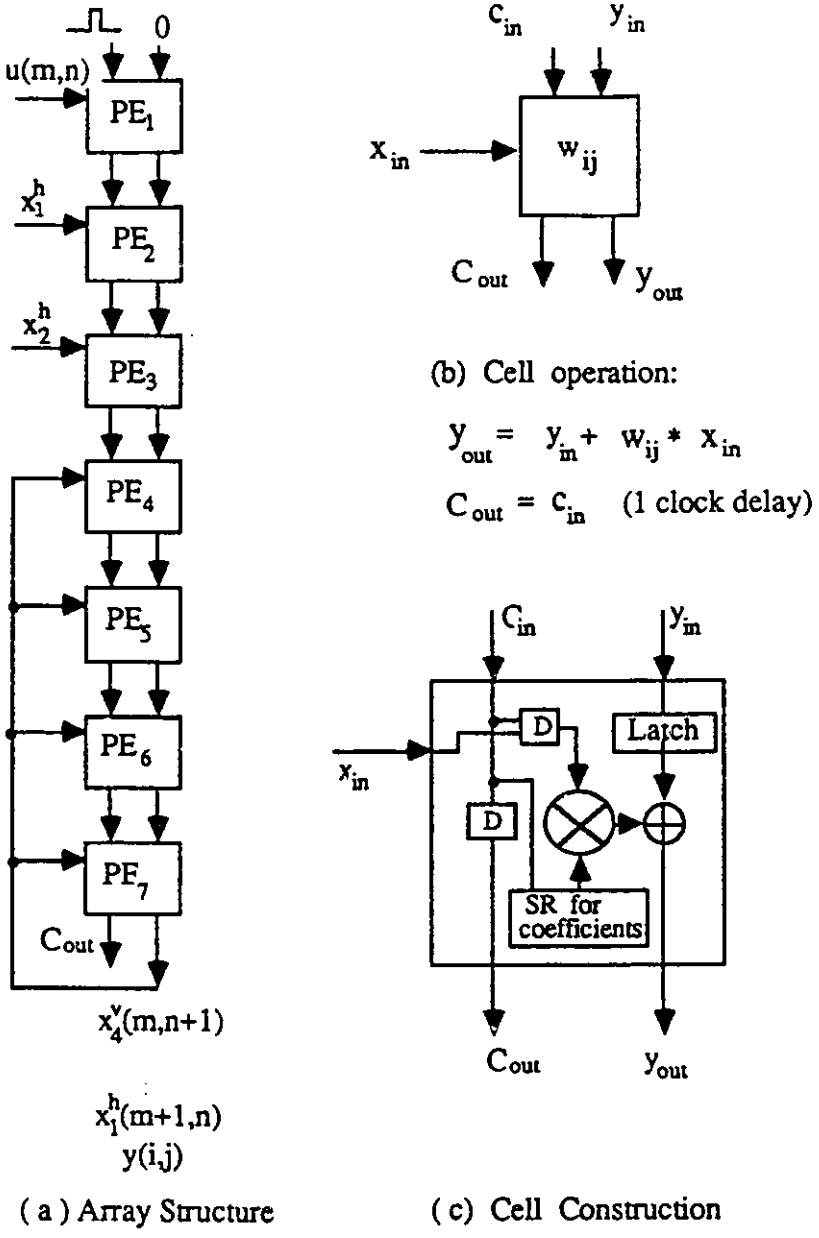



Figure 4.7: The column processor for 2-D second-order state-space filtering.

design, the input $u(m,n)$ and the components of the state vector remain at the cells of the array for 7 clock periods. The coefficients of each row in the matrix circulate around each cell of the array. The partial results move systolically from cell to cell in the up-to-down direction. After a delay of 7 clocks, the final values of $y(m,n), x_1^h(m,n), x_2^h(m,n), x_1^v(m,n)...)$ are obtained from the bottom at the rate of one output per clock. And, $u(m,n+1), x_1^h(m,n), x_2^h(m,n+1)$ (initial condition), $x_1^v(m,n+1), ...x_4^v(m,n+1)$(previous output) are loaded into the cells one after another by a control pulse, which is applied to each cell with a delay. Thus, $u(m,n)$ enters the array first, then, $u(m,n+1), u(m,n+2)...$ follow.

The "global speed-up" structure consists of column processors. Consider the structure shown in Figure 4.8. There are 7 column processors, i.e., $C_0, C_1, ..., C_6$, in this design. Each column processor processes the pixels in one column of images. Based on the analysis of the previous section, pixels at $(m+1,n)$ can not be processed until the horizontal components, $X^h(m+1,n)$, are available. The pixels at $(m,n+1)$ have to wait for $X^v(m,n+1)$ to be completed. Thus, in this structure the horizontal components are transferred from one column processor to the next column processor, while the vertical components are generated in each column processor and return to this array for processing the pixels in the same column.

Initially, $u(0,0)$ enters the first column array $C_0$. 7 clock periods later, $y(0,0)$ is obtained. Then, $x_1^h(1,0)$ is available. It is followed by $x_2^h(1,0), x_1^v(0,1), ..., x_4^v(0,1)$. $x_1^h(1,0), x_2^h(1,0)$ are thus sent to $C_1$. $x_1^v(0,1), ..., x_4^v(0,1)$ are reloaded to $C_0$ immediately. When $y(0,0)$ emerges from $C_0$, $u(1,0)$ enters $C_1$ and $u(0,1)$ enters $C_0$, respectively. Each column processor starts to work 7 clock periods later than the previous one. As a result, the pixels at diagonals, i.e., $u(1,0), u(0,1)$ are being processed concurrently, $u(2,0), u(1,1), u(0,2)$ enter the array at the same time, etc. After initiation, 7 pixels can be processed simultaneously. Thus a throughput rate of 1 pixel/clock period is achieved. This structure processes multiple pixels at diagonal direction concurrently and has the following features:

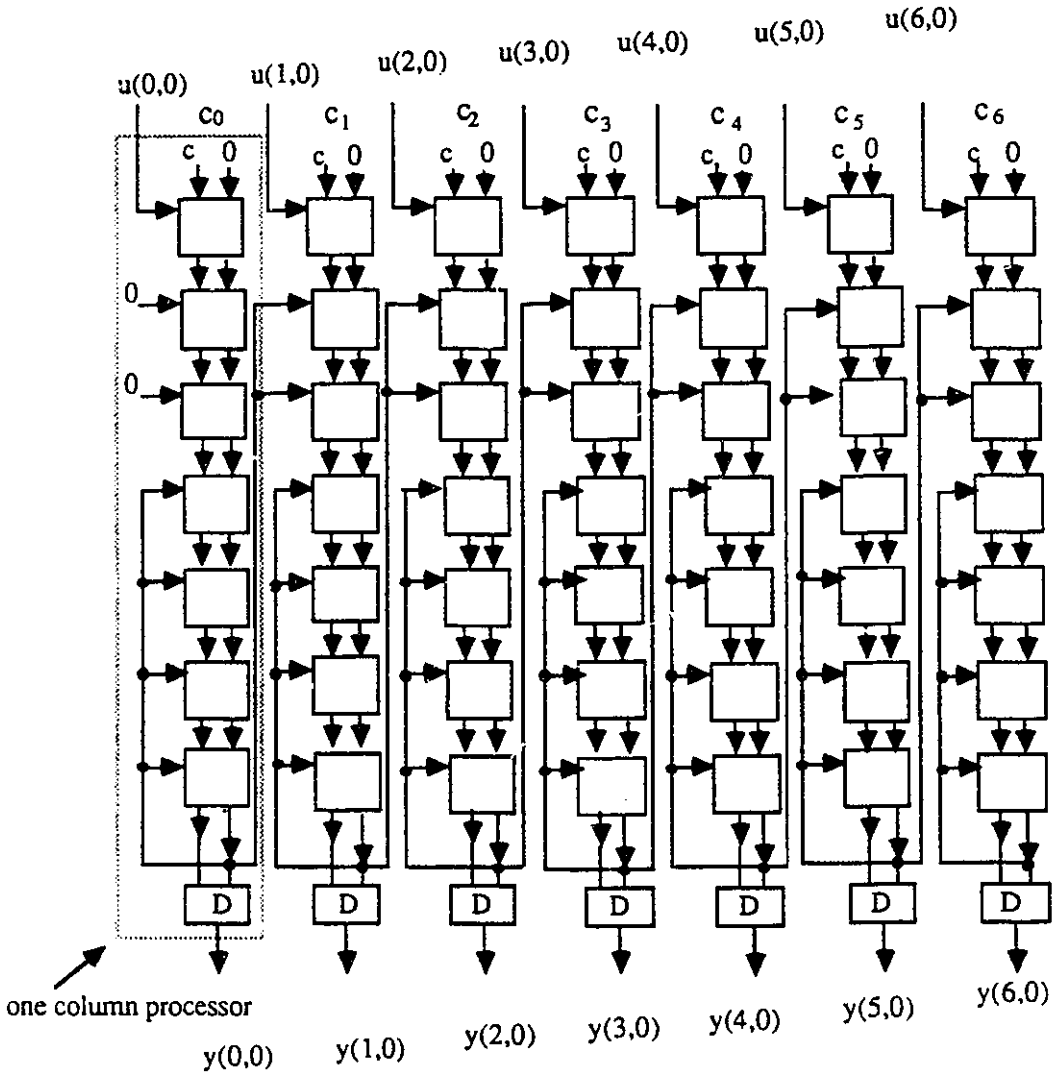



Figure 4.8: The global speedup structure for filtering multiple columns of images concurrently.

1. It is easier to arrange the inputs and outputs. Each column of images inputs to the corresponding processor with a skew of $l$ clock periods and the outputs emerge from that array sequentially. The images do not need remapping and the regular line by line scanning method can be used, which is important in practice. However, a frame buffer is still required.

2. The structure is simple, regular and realizable. Each column processor contains 7 multiplication/addition cells and one input, one output. Based on up-to-date technology, it is feasible to design one VLSI chip for this processor. Furthermore, modular design is suitable for this structure. The system is easily to be changed to adapt to the throughput requirements of the different applications.

3. The processing speed is high and easily adjustable. As shown in Figure 4.8, one pixel is processed during each clock period. If $N \times l$ PEs are used, the throughput rate can be up to $N$ pixels per $l$ clock periods. The processing time required for one image is $M \times l + (N-1) \times l$ clock-periods. $L$ images can be processed in $L \times M \times l + (N-1) \times l$ clock periods. When $L \times M \gg (N-1)$, the time complexity, i.e., the total time required for filtering $L$ pictures, is $O(L \times M \times l)$ by using $N \times l$ PEs. It is obvious that after initiation, the efficiency is 100%.

4. Local memory for storage of the state vector is not always needed. When the vertical components of the state vector are generated, they are returned to the array for the next line processing immediately. Similarly, the horizontal components will be transferred to the successive arrays as soon as they are available.

5. No peripheral circuits, such as multiplexing components, are required, since for each picture, the initial state values are used only at the beginning of the

process. As soon as the new state values are computed by the array, these will be used for the next computation.

## 4.5 Comparisons and Conclusions

In the past few years, several high speed architectures for digital image filtering were proposed. In [82,83], Ty and Venetsanopoulos proposed the distributed arithmetic architecture, minimum cycle time filer architecture, multiprocessor element architecture, and VLSI architecture by decomposition. A video rate architecture for a fully recursive filter was presented in [84]. All these architectures are capable of processing a $512 \times 512$ pixel image in real time or near real time. The structure is the direct form 2-D IIR filter. In [86], Parhi and Messerschmitt gave several possible implementations of pipelining, one- and two-dimensional block processing for both direct form and state-space form. In [74], a systolic array implementation for state-space filtering with diagonal scanning was introduced. In this chapter, we have presented three new architectures specially for high speed state-space filtering. The direct state-space form array has the same complexity as the direct form IIR filters and can process one pixel at every clock time with the diagonal scanning method. The advanced state update architecture is a 2-D systolic array. It can process one pixel within one multiplication/addition clock time with the regular line-by-line scanning method. Since the global speed-up architecture is very modular and flexible, the throughput rate can be easily adjusted to the different requirements. A very high throughput rate can be achieved by using more column processors. A comparison between different architectures is given in Table 4.1.

It is shown that the proposed architectures are attractive. The architectures for transformed state-space filters can process images at very high speed with high numerical performance.

| Architectures | Cycle time | Latency | Hardware complexity | Modularity | Comments |
|---|---|---|---|---|---|
| Distributed Arithmetic Architecture[81] | $4T_a + T_{ma} + T_d$ (1 pixel/156ns) | Same as cycle time | Built by TTL chips | No | Direct form recursive filters |
| Minimum-Cycle-Time Architecture[81] | $T_{ma} + 2\,T_a + T_d$ (1 pixel/113ns) | System clock period + $T_a + T_d$ | 18 Processing Elements | No | Direct form recursive filters |
| Multiple-Processing-Element Architecture[81] | $T_m + 4\ T_a + T_d$ (1 pixel/151ns) | Same as cycle time | (not clear) | Yes | Direct form recursive filters |
| VLSI Architecture by Decomposition [81] | $9\ T_{ma} + 9\ T_a$ + (control delay) | Same as cycle time | Two NMOS VLSI chips | Yes | Direct form recursive filters |
| Video Rate Architecture[83] | 375 ns for 3 pixels (1 pixel/125ns) | 375 ns | 18 PEs for second order row processor | Yes | Direct form fully recursive filters |
| Systolic array for Diagonal Scanning [74] | $T_m + T_a + T_l$ (<100 ns) | 7 cycle times | 49 PEs | Yes Systolic array | State-space recursive filters Diagonal scanning |
| Global Speed-Up Architecture | $T_m + T_a + T_l$ (<100ns) | 7 cycle times | 49 PEs | Yes Systolic array | State-space recursive filtering Regular scanning Adjustable rate |
| Advanced State Update Architecture | $T_m + T_a + T_l$ (<100ns) | 9 cycle times | 7 x 16 PEs | Yes Systolic array | State-space recursive filtering Line-by-line scanning One input/output |

Table 4.1: Comparisons among various 2-D high speed architectures. Where, $T_m$: multiplication time; $T_l$: latch set up and propagation time; $T_a$: addition time; $T_{ma}$: memory access time; $T_d$: shift register set up and propagation delay time.

# Chapter 5

# Realization and Implementation of Adaptive Recursive State-Space Filtering

## 5.1 Introduction

In this chapter, we will extend the state-space technique to the area of adaptive filtering and develop the adaptive algorithm for recursive state-space filters.

Recently, there has been growing interest in using adaptive digital filters for noise cancellation, echo cancellation, channel equalization, system identification, and adaptive beamforming or adaptive array processing [104]. This interest is due in part to remarkable advances in VLSI technology. Computationally powerful single-chip digital signal processors (DSP's) are now available for these types of applications.

Typically, adaptive filters are implemented as transversal finite-impulse-response (FIR) filters and the simple LMS (least mean square) adaptation algorithm [105] is used to adjust the zeroes of the filter. This is because FIR adaptive filters are well behaved, i.e., they are guaranteed to remain stable and they usually converge to a

globally optimum condition because the error surface is unimodal [106].

Over the last several years. adaptive infinite-impulse-response (IIR) filtering has been an active area of research [107.108], and it has been considered for a variety of problems in signal processing and communications. It is well known that the fundamental properties of adaptive algorithms are stability, speed of convergence, misadjustment errors, robustness, numerical complexity, and round-off error analysis of adaptive algorithms. The basic consideration to use adaptive IIR filters is that in many cases, lower-order adaptive recursive filters are required to achieve satisfactory performance than the adaptive FIR filters since both the poles and zeroes are adapted. For example, a fifth-order IIR filter requiring 9 multipliers and 8 adders per output sample may match the unknown system as well as a 64th-order FIR filter that requires 64 multipliers and 63 adders per output sample [109].

For this reason, numerous algorithms have been proposed for implementing adaptive IIR direct form filters [110,111,112,113,114,115]. However, the problems such as instability, multimodal performance surface and intensive computation for the gradient calculation limit the applications of the adaptive recursive filtering. Because of the fast convergence and modularity, lattice filters have become popular in the adaptive signal processing community. The disadvantage of the lattice filter algorithms are their numerical complexity and the mathematical sophistication [116,117,118,119].

It has been shown in the previous chapters that the state-space technique is a very useful tool in the formulation and analysis of linear systems. The state-space digital filters are capable of providing improved nonlinear performance. For example, a normal structure for a linear stable filter has lower coefficient sensitivity and is free of overflow oscillations [60,61]. Furthermore the state variable representation results in matrix operations and is easily implemented by systolic arrays [72,103]. Being able to adapt arbitrary state-space filters gives the designer the freedom to explore the performance advantages of different structures [120]. The only problem of

the state-space filtering is an increase in the number of the multipliers which can be resolved by other techniques. Consequently, there has been an interest in extending the gradient-based adaptation algorithm to the state-space recursive filtering [121,122,123,124]. To search for better adaptive filter structures and the advantages of the state-space filtering, in this chapter, an adaptive recursive state-state filtering algorithm is presented, and the performance of the adaptive state-space filters is studied. In the following section, the problem of the adaptive state-space filtering is given first. The LMS algorithm for state-space filtering is developed in section 5.3. The gradients for the parameters are derived directly from the state equations and formulated by matrix derivative linear operations. In section 5.4, to reduce the computational complexity, the parallel form of adaptive state-space filters is presented. The parallel form leads to easier stability monitoring, which is then discussed in section 5.5. A possible roundoff noise performance improvement by using adaptive state-space filtering is shown in section 5.6 by using simulation examples. Then, to study the convergence property of the adaptive state-space filtering, several simulation results are given in section 5.7. In section 5.8, in order to speed up the filtering and the adaptation procedures, a VLSI architecture is suggested for real-time processing.

## 5.2 Problem Statement

As given in Chapter 3, an Nth-order state-space digital filter can be described by:

$$X(n+1) = \mathbf{A}X(n) + \mathbf{B}u(n) \tag{5.1}$$

$$y(n) = \mathbf{C}X(n) + \mathbf{D}u(n) \tag{5.2}$$

where $u(n)$ and $y(n)$ are input signal and output signal. $X(n)$is the state vector. $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are system matrices with appropriate dimensions. For a time-varying state-space filter, equations 5.1 and 5.2 become:

$$X(n) = \mathbf{A}(n)X(n-1) + \mathbf{B}(n)u(n-1) \tag{5.3}$$

$$y(n) = \mathbf{C}(n)\mathbf{X}(n) + \mathbf{D}(n)u(n) \tag{5.4}$$

A coefficient vector of the filter $W(n)$ can be defined as

$$W(n)^t = [cs^t[\mathbf{A}(n)]|\mathbf{B}(n)^t|\mathbf{C}(n)|\mathbf{D}(n)]$$

where $^t$ indicates transposition and $cs[\mathbf{A}(n)]$ denotes the column string of the matrix $\mathbf{A}(n)$.

Figure 5.1 illustrates the general structure and components of an adaptive state-space filter with input $u(n)$ and output $y(n)$. It is comprised of a time-varying state-space filter, characterized by the adjustable coefficients $W(n)$, and a recursive algorithm. The problem to be solved is to find an appropriate algorithm that adjusts $W(n)$ so that $y(n)$ approximates some desired response $d(n)$ as closely as possible and the resulting filter is zero-input asymptotically stable.

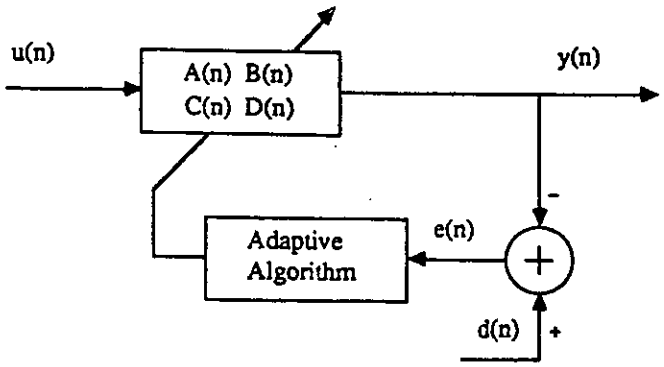



Figure 5.1: Adaptive state-space filter. An adaptive state-space filter is comprised of a time-varying state-space filter and an adaptive algorithm.

## 5.3 Algorithm Development

Fundamentally, there have been two approaches to adaptive IIR filtering that correspond to different formulations of the prediction error; these are known as equation error and output error methods. The equation-error approach can lead to biased

estimates of the coefficients. The output-error approach can converge to a local minimum estimate and its convergence properties are not easily predicted. As a result, there is a trade-off between these two approaches [125].

A variety of adaptation algorithms have been proposed in the past three decades. Among them, the LMS (least mean square) adaptation algorithm is a simple, robust, and easily understandable one. Because of these properties, the LMS algorithm has become popular in adaptive signal processing applications. Hence, we will use the LMS algorithm to develop our adaptive recursive state-space filtering. The LMS algorithm is a stochastic version of a steepest descent algorithm and is for the output-error formulation.

If the error signal with the index $n$ is given by $e(n) = d(n) - y(n)$. With the LMS algorithm, the parameters will be adjusted to minimize the instantaneous squared error $e^2(n)$. The general parameter update equation is given by [105]

$$W(n+1) = W(n) + 2\mu e(n) \bigtriangledown y(n) \tag{5.5}$$

where $\mu$ is a step-size to control convergence of the algorithm, and

$$\begin{aligned} \bigtriangledown y(n) &\equiv \frac{\partial y(n)}{\partial w_i(n)} \\ &\equiv [\mathcal{D}_{cs^t[A]} | \mathcal{D}_{B^t} | \mathcal{D}_C | \mathcal{D}_D]^t y(n) \end{aligned} \tag{5.6}$$

is the gradient vector of $y$ with respect to the vector $W$, where $\mathcal{D}_E T(E)$ is the matrix(vector) derivative linear operator defined by [126]:

$$\mathcal{D}_{E_{r\times s}} T_{\xi\times\nu}(E) \equiv \begin{bmatrix} \mathcal{D}_{e_{11}} T(E) & \cdots & \mathcal{D}_{e_{1s}} T(E) \\ \vdots & & \vdots \\ \mathcal{D}_{e_{r1}} T(E) & \cdots & \mathcal{D}_{e_{rs}} T(E) \end{bmatrix} \tag{5.7}$$

with

$$\mathcal{D}_{e_{ij}} T(E) \equiv \begin{bmatrix} \frac{\partial t_{11}(E)}{\partial e_{ij}} & \cdots & \frac{\partial t_{1\nu}(E)}{\partial e_{ij}} \\ \vdots & & \vdots \\ \frac{\partial t_{\xi 1}(E)}{\partial e_{ij}} & \cdots & \frac{\partial t_{\xi\nu}(E)}{\partial e_{ij}} \end{bmatrix} \tag{5.8}$$

During the adaptation process, at each step, the error signal is calculated and the derivative value for each parameter has to be calculated concurrently with the filtering calculations. Then the parameters are modified to new values. According to the state equations 5.3 and 5.4, the time domain gradients can be calculated as following [127]:

$$\mathcal{D}_{cs^t[\mathrm{A}]}y(n) = cs^t[(\mathrm{I}_N \otimes \mathrm{C})\mathcal{D}_{\mathrm{A}}X(n)] \tag{5.9}$$

$$\mathcal{D}_{\mathrm{B}^t}y(n) = \mathrm{C}\mathcal{D}_{\mathrm{B}^t}X(n) \tag{5.10}$$

$$\mathcal{D}_{\mathrm{C}}y(n) = X^t(n) \tag{5.11}$$

$$\mathcal{D}_{\mathrm{D}}y(n) = u(n) \tag{5.12}$$

where $\mathcal{D}_{\mathrm{A}}X(n)$ and $\mathcal{D}_{\mathrm{B}^t}X(n)$ are calculated from 5.3 as below:

$$\mathcal{D}_{\mathrm{A}}X(n) = \mathcal{D}_{\mathrm{A}}\mathrm{A}(\mathrm{I}_N \otimes X(n-1)) + \mathrm{I}_N \otimes \mathrm{A}(n)\mathcal{D}_{\mathrm{A}}X(n-1) \tag{5.13}$$

$$\mathcal{D}_{\mathrm{B}^t}X(n) = \mathrm{I}_N u(n-1) + \mathrm{A}(n)\mathcal{D}_{\mathrm{B}^t}X(n-1) \tag{5.14}$$

Note that $\mathrm{I}_N$ is the identity matrix of order $N$ and $\mathbf{R}_{s\times t} \otimes \mathbf{Q}_{r\times q}$ is the Kronecker (direct) product of $\mathbf{R}$ and $\mathbf{Q}$ of dimension $sr \times tq$ defined as

$$\mathbf{R} \otimes \mathbf{Q} \equiv \begin{bmatrix} r_{11}\mathbf{Q} & \cdots & r_{1t}\mathbf{Q} \\ \vdots & & \vdots \\ r_{s1}\mathbf{Q} & \cdots & r_{st}\mathbf{Q} \end{bmatrix}.$$

Finally, the updating equations for each element in the matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are obtained:

$$a_{ij}(n+1) = a_{ij}(n) + 2\mu e(n)\mathrm{C}(n)\mathcal{D}_{a_{ij}}X(n) \tag{5.15}$$

$$b_i(n+1) = b_i(n) + 2\mu e(n)\mathrm{C}(n)\mathcal{D}_{b_i}X(n) \tag{5.16}$$

$$c_i(n+1) = c_i(n) + 2\mu e(n)x_i(n) \tag{5.17}$$

$$d(n+1) = d(n) + 2\mu e(n)u(n) \tag{5.18}$$

where

$$\mathcal{D}_{a_{i,j}} X(n) = \mathcal{D}_{a_{i,j}} \mathrm{A} X(n-1) + \mathrm{A}(n) \mathcal{D}_{a_{i,j}} X(n-1) \tag{5.19}$$

$$\mathcal{D}_{b_i} X(n) = \mathcal{D}_{b_i} \mathrm{B} u(n-1) + \mathrm{A}(n) \mathcal{D}_{b_i} X(n-1). \tag{5.20}$$

To illustrate this adaptive state-space algorithm, the diagram of the adaptive state-space filter is redrawn in Figure 5.2. Notice that equations 5.3, 5.19 and 5.20 are calculated recursively and therefore the values of $X(n), u(n), \mathcal{D}_{\mathrm{A}} X(n)$ and $\mathcal{D}_{\mathrm{B}} X(n)$ have to be stored. The implementation of this algorithm will be discussed in the following section.

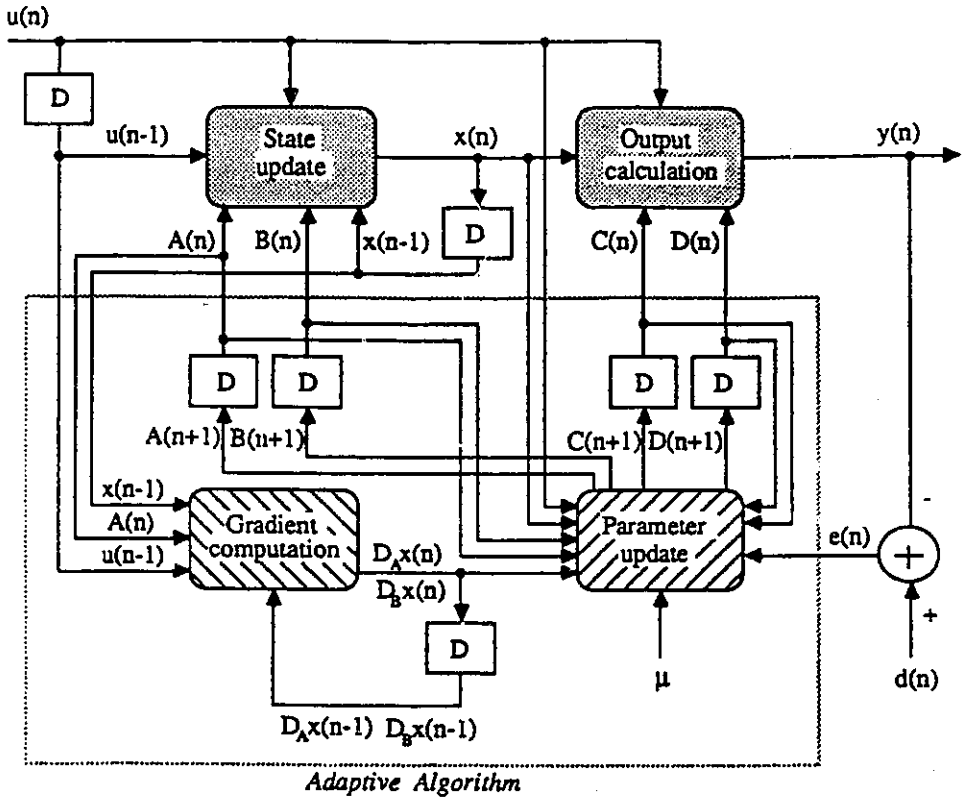



Figure 5.2: Adaptive state-space filter with LMS algorithm.

## 5.4 Parallel Form Realization

As shown in chapter 3, the transformation that obtains the minimum roundoff noise filter results in an increase of computational complexity. In order to reduce the number of multipliers in a minimum roundoff noise realization, the cascade form and the parallel form of second-order state-space filters are suggested. On the other hand, for adaptive IIR filtering, it has been shown that a decomposed filter structure consisting of parallel or cascaded lower order sections is superior for two reasons: 1). Coefficient sensitivities of the parallel and cascade forms are much lower than that for the direct form, and 2). it is simple to monitor stabilities for the parallel and cascade forms [109,125]. Here, we use the parallel form realization for adaptive state-space recursive filters.

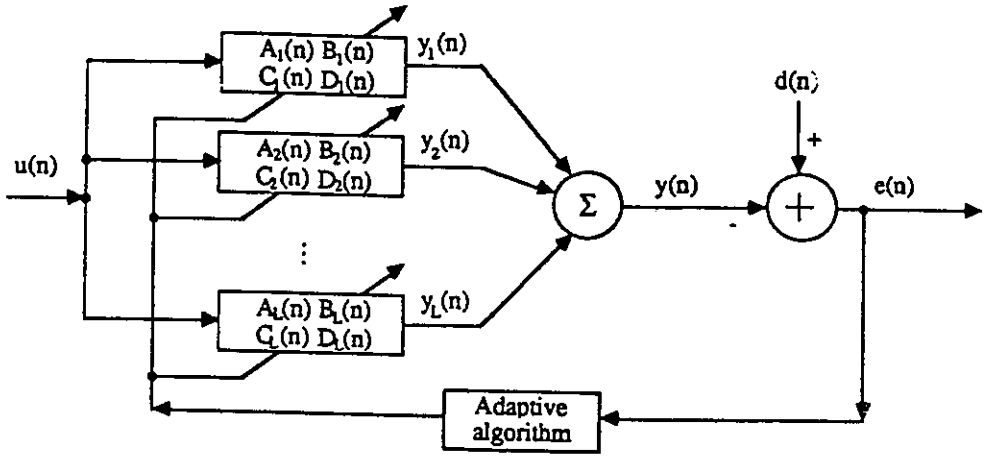



Figure 5.3: Parallel form adaptive state-space filter.

Suppose the original adaptive filter is characterized by an $N$th-order transfer function. The parallel form is derived from a partial fraction expansion of the pole-zero filter, resulting in the sum of $L = N/2$ second-order sections as shown in Figure 5.3. The state-space representation of the parallel combination with $L$

second-order sections is described by:

$$\mathbf{A} = \begin{bmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & A_L \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_L \end{bmatrix},$$
$$\mathbf{C} = \begin{bmatrix} C_1 & C_2 & \cdots & C_L \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} D_1 & D_2 & \cdots & D_L \end{bmatrix}. \tag{5.21}$$

The 0's are $2 \times 2$ matrices filled with zeroes and the $A_i, B_i, C_i, D_i$ are $2 \times 2$ matrices describing each of the second-order sections. The instantaneous output error is now given by

$$e(n) = d(n) - \Sigma_{i=1}^{N/2} y_i(n), \tag{5.22}$$

and this error signal will be used to control the coefficient update.

There are several advantages for using parallel form adaptive state-space filtering. First of all, an important feature of parallel form realization is that the state update matrix is block diagonal with $2 \times 2$ matrices along the diagonal. Such matrices are called quasi-diagonal. As a result, the sections are essentially independent and the gradient of one section does not depend on the coefficients of any other section. Therefore, the gradient components are easy to compute. Second, the stability monitoring is trivial for parallel form, which will be discussed in the next section. Third, because the output roundoff noise of the system is the sum of the noise of each section, minimization of the output noise implies optimizing each section separately. On the other hand, the cascade form is very similar to the parallel form. Stability monitoring is also trivial. However, the complexity of the gradient is significantly greater. This is because the output signal of each section depends on the coefficients of that section as well as all previous sections.

## 5.5 Stability Monitoring

One of the major problems associated with the adaptive IIR filtering is instability. The filter may become unstable during adaptation if one or more poles of the filter accidentally update outside the unit circle and remain there for a significant length of time. As a result, the output can grow without bound. Therefore it is necessary to overcome this problem by monitoring the poles of the transfer function. If it is unstable, the feedback coefficients of the IIR filter should be modified so that all the poles are projected to within the stability region [128]. The problem is that a real-time stability test can be a significant computational burden in many situations. Which structure of the adaptive recursive filter may require fewer computations for the stability test?

Recall the expressions for the state-space filter given in equations 5.3 and 5.4. Using z-transforms, the transfer function of the system can be derived as:

$$H(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}. \tag{5.23}$$

This transfer function $H(z)$ is rational, and the denominator polynomial of the transfer function $H(z)$ can be expressed as:

$$D(z) = det(z\mathbf{I} - \mathbf{A}). \tag{5.24}$$

In other words, the poles of the transfer function are the eigenvalues of the matrix **A**. It shows that the filter is stable if and only if all eigenvalues of matrix **A** have modulus less than one. Therefore, the stability is guaranteed through adding the constraint during the adaptation process:

$$\max_i |\lambda_i| < 1 \tag{5.25}$$

where, $\lambda_i$ is the $i^{th}$ eigenvalue of the matrix **A**. This sufficient condition for stability can be used as upper bound in the process. For a second-order normal structure with $\mathbf{A} = \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix}$, the eigenvalues are $\alpha \pm j\beta$. During adaptation, only two

coefficients are free to change, and the constraint is: $\alpha^2 + \beta^2 \leq 1$. This can easily be checked. For general second order filters, the constraints are:

$$a_{11}a_{22} - a_{12}a_{21} \leq 1. \tag{5.26}$$

For the second-order direct form IIR filter,

$$H(z) = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

if $1 + a_1 - a_2 > 0, 1 - a_1 - a_2 > 0$, and $1 + a_2 > 0$, the system is stable. For high order filters, the existing approaches are either computationally expensive or nonrobust. For the lattice form, the stability requires only that each reflection coefficient have a magnitude less than 1 [125].

It is shown that for the stability test, the lattice form requires fewer computations. The computation complexities are comparable for the state-space structures and the direct form IIR filters. For the second-order normal structure, the stability check is quite simple.

## 5.6 Output Roundoff Noise

As shown in chapter 3, a realization of an IIR digital filter employs three basic operations: multiplication by constants (the filter coefficients), accumulation of the products, and storage into memory (register or latch). The results of accumulations inside the filter must eventually be quantized because the multiplications always increase the number of bits required to represent the products. In this section, we shall consider: given the transfer function $H(z)$ of the filter and the roundoff noise model, how much noise performance improvement of using state-space adaptive filtering over the direct-form adaptive filtering can be obtained?

### 5.6.1 Calculation of Output Roundoff Noise

The roundoff noise model and the formula for calculating the output roundoff noise of a state-space filter are given in [61]. For a state-space filter with matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$, the output noise is expressed in terms of the covariances $\mathbf{K}$ and $\mathbf{W}$.

$$\sigma^2_{total} = \frac{\delta^2 q^2}{12} \Sigma_{i=1}^{p} K_{ii} W_{ii} \tag{5.27}$$

where, $\delta$ is the scaling factor to avoid the overflow, $q$ is called the quantization step size dependent upon the number of bits used. The covariances $\mathbf{K}$ and $\mathbf{W}$ are defined as:

$$\mathbf{K} = \mathbf{A}\mathbf{K}\mathbf{A}^t + \mathbf{B}\mathbf{B}^t \tag{5.28}$$

$$\mathbf{W} = \mathbf{A}^t\mathbf{W}\mathbf{A} + \mathbf{C}^t\mathbf{C} \tag{5.29}$$

It is shown that for the equal word length case, minimization of the roundoff noise implies minimization of the arithmetic mean of $K_{ii}W_{ii}$[61]. Therefore, for comparing the noise performance of different structures, the noise measure is defined as:

$$N = \Sigma_{i=1}^{p} K_{ii} W_{ii} \tag{5.30}$$

where $p$ is the dimension of the matrices.

### 5.6.2 Examples

Since the parallel form is used for our adaptive state-space filtering, the optimization theory is applied to the second-order subfilters, and the examples used here are second-order filters. The output roundoff noise of a higher order filter is minimized if the noise of each section is minimized.

*Example 1.*

The transfer function of the filter is:

$$H_1(z) = \frac{1}{1 + z^{-1} + 0.5z^{-2}}. \tag{5.31}$$

By using the direct-form adaptive filtering algorithm, we can obtain the desired coefficients given in the equation 5.31 after the system convergences. The state variable representation of the system is:

$$\mathbf{A}_d = \begin{bmatrix} 0.0 & 1.0 \\ 0.5 & -1.0 \end{bmatrix} \quad \mathbf{B}_d = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix} \quad \mathbf{C}_d^t = \begin{bmatrix} -0.5 \\ -1.0 \end{bmatrix} \quad \mathbf{D}_d = 1.0 \tag{5.32}$$

According to the optimization procedures described in [62], the transformed minimum roundoff noise realization is obtained as:

$$\mathbf{A}_t = \begin{bmatrix} -0.5 & -0.5 \\ 0.5 & -0.5 \end{bmatrix} \quad \mathbf{B}_t = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \quad \mathbf{C}_t^t = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \quad \mathbf{D}_t = 1.0 \tag{5.33}$$

When we use the proposed state-space adaptive algorithm, the system matrices are initialized to :

$$\mathbf{A}_i = \begin{bmatrix} 0.25 & -0.25 \\ 0.25 & 0.25 \end{bmatrix} \quad \mathbf{B}_i = \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix} \quad \mathbf{C}_i^t = \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix} \quad \mathbf{D}_i = 0.0 \tag{5.34}$$

During the adaptation, the parameter vector of the adaptive filter is updated under a constraint: the adaptive filter is a normal structure. After the system convergences, the system matrices are:

$$\mathbf{A}_r = \begin{bmatrix} -0.49995 & -0.50003 \\ 0.50003 & -0.49995 \end{bmatrix} \quad \mathbf{B}_r = \begin{bmatrix} 0.70410 \\ 0.71025 \end{bmatrix} \quad \mathbf{C}_r^t = \begin{bmatrix} 0.70409 \\ 0.70990 \end{bmatrix} \quad \mathbf{D}_r = 1.00000 \tag{5.35}$$

The $\mathbf{K}, \mathbf{W}$ matrices and the noise measure $N$ are then calculated. They are shown in Table 5.1.

*Example 2.*

The second example is a similar second-order filter with transfer function:

$$H_2(z) = \frac{1}{1 - 1.2z^{-1} + 0.6z^{-2}}. \tag{5.36}$$

For this example. the direct-form adaptive filtering algorithm needs stability check and pole projection during the adaptation process. The state variable representation

| | K | | W | | $N$ |
|---|---|---|---|---|---|
| Direct Form | 2.4000000 | -1.6000000 | 0.6000000 | 0.8000000 | 4.8000000 |
| Filter | -1.6000000 | 2.4000000 | 0.8000000 | 1.4000000 | |
| Transformed | 1.1999770 | 0.3999923 | 0.7999846 | 0.3999923 | 1.9199260 |
| Filter | 0.3999923 | 0.7999846 | 0.3999923 | 1.1999770 | |
| Resulting | 1.1966970 | 0.4017961 | 0.7964718 | 0.3982112 | 1.9198020 |
| Filter | 0.4017961 | 0.8036467 | 0.3982112 | 1.2028500 | |

Table 5.1: The comparison of the output roundoff noise for different realizations in Example 1.

of the system is:

$$\mathbf{A}_d = \begin{bmatrix} 0.0 & 1.0 \\ -0.6 & 1.2 \end{bmatrix} \quad \mathbf{B}_d = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix} \quad \mathbf{C}_d^t = \begin{bmatrix} -0.6 \\ 1.2 \end{bmatrix} \quad \mathbf{D}_d = 1.0 \qquad (5.37)$$

Similarly, by calculations, the transformed minimum roundoff noise realization is obtained as:

$$\mathbf{A}_t = \begin{bmatrix} 0.600 & -0.424 \\ 0.566 & 0.600 \end{bmatrix} \quad \mathbf{B}_t = \begin{bmatrix} -0.267 \\ 0.377 \end{bmatrix} \quad \mathbf{C}_t^t = \begin{bmatrix} 2.267 \\ -1.588 \end{bmatrix} \quad \mathbf{D}_t = 1.000 \quad (5.38)$$

When using the state-space adaptive algorithm, the system matrices are initialized to:

$$\mathbf{A}_i = \begin{bmatrix} 0.25 & -0.25 \\ 0.25 & 0.25 \end{bmatrix} \quad \mathbf{B}_i = \begin{bmatrix} -0.25 \\ 0.25 \end{bmatrix} \quad \mathbf{C}_i^t = \begin{bmatrix} 0.25 \\ -0.25 \end{bmatrix} \quad \mathbf{D}_i = 0.0 \qquad (5.39)$$

During the adaptation process, no stability monitoring is applied. The resulting system is given by the following matrices:

$$\mathbf{A}_r = \begin{bmatrix} 0.60000 & -0.45764 \\ 0.52440 & 0.60000 \end{bmatrix} \quad \mathbf{B}_r = \begin{bmatrix} -0.29541 \\ 0.35192 \end{bmatrix} \quad \mathbf{C}_r^t = \begin{bmatrix} 2.24015 \\ -1.52138 \end{bmatrix} \quad \mathbf{D}_r = 1.00000 \qquad (5.40)$$

The $\mathbf{K}, \mathbf{W}$ matrices and the noise measure $N$ are given in Table 5.2.

From these two examples, it is shown that the adaptive state-space algorithm can result in a realization with less output roundoff noise.

| | K | | W | | $N$ |
|---|---|---|---|---|---|
| Direct Form | 3.5714290 | 2.6785720 | 1.2857150 | -1.6071430 | 13.7755100 |
| Filter | 2.6785720 | 3.5714290 | -1.6071430 | 2.5714290 | |
| Transformed | 0.2509687 | -0.0893540 | 9.1448650 | -3.2313890 | 4.5906540 |
| Filter | -0.0893540 | 0.2528736 | -3.2313890 | 9.0779730 | |
| Resulting | 0.2869766 | -0.0875328 | 8.6024060 | -3.2579570 | 4.5978980 |
| Filter | -0.0875328 | 0.2307536 | -3.2579570 | 9.2271920 | |

Table 5.2: The comparison of the output roundoff noise for different realizations in Example 2.

## 5.7 Convergence Rate Study

There have been relatively few studies of the convergence properties of adaptive IIR filters. Most of the results are derived from work in system identifications. Therefore, computer simulations are still the only method of studying the rate of convergence. Consequentially, the adaptive state-space LMS algorithm is applied to the system identification problems [125].

We assume that the adaptive filter is operating in a parallel form configuration and that it has sufficient order to model the unknown system coefficients. A block diagram of the computer simulations is given in Figure 5.4, where the input to the unknown system and the adaptive filter is computer generated white noise. An Nth-order adaptive filter is required to adjust its coefficients to match an Nth-order reference transfer function. Notice that both the reference filter and the state-space adaptive filter are in the parallel forms of second-order sections.

- State-space adaptation versus direct-form adaptation.

  The transfer function of the reference system is given in 5.31. In this experiment, the direct adaptive IIR LMS algorithm and the adaptive state-space LMS algorithm are applied to identify the reference system. In both cases, the step-size of 0.005 is adopted. First, the state-space model of the reference

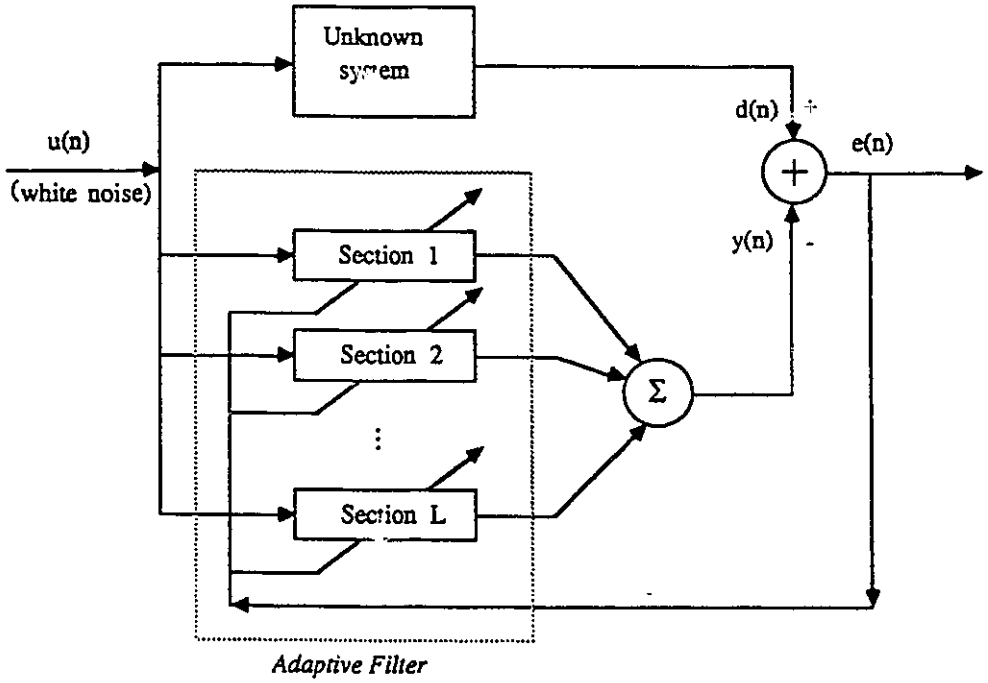



Figure 5.4: Adaptive state-space filtering for system identification. Notice that the adaptive filter is in the parallel form.

system is chosen as a direct state variable representation as shown in equation 5.32. The adaptive filter convergences to the desired system by using the state-space LMS algorithm. In this simulation, four elements of the system matrices are constant, i.e., $a_{11} = 0, a_{12} = 1, b_1 = 0$, and $b_2 = 1$. Other five elements are updated according to the state-space LMS algorithm. Next, the transformed state-space structure with minimum roundoff noise is used. The state-space adaptive algorithm can successfully identify the system as the desired matrices are obtained (see equation 5.34 and 5.35). Third, the direct IIR LMS algorithm is used to identify the coefficients of the transfer function directly. The convergence rates of these three methods are given in Figure 5.5.

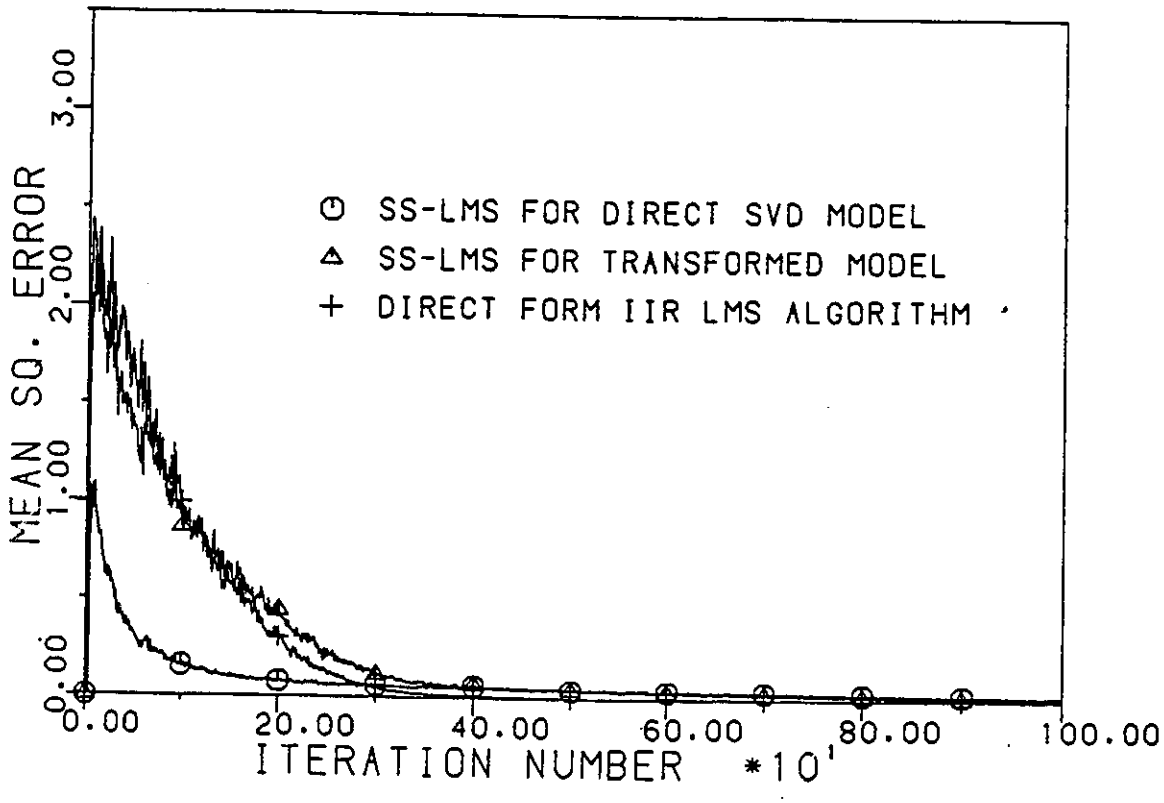



Figure 5.5: Comparison of convergence rates of direct form IIR adaptive algorithm and state-space adaptive algorithm.

- Parallel form adaptation.

The reference system has the transfer function as follows:

$$H(z) = \frac{2 - 0.2z^{-1} + 1.1z^{-2}}{1 - 0.2z^{-1} - 0.1z^{-2} + 0.3z^{-4}} \tag{5.41}$$

It is easy to show that

$$H(z) = H_1(z) + H_2(z) \tag{5.42}$$

where $H_1(z)$ and $H_2(z)$ are given in 5.31 and 5.36.

The optimized state-space matrices for $H_1(z)$ and $H_2(z)$, and the resulting matrices by using state-space adaptive algorithm independently are given in the previous section. The learning curves for $H_1(z)$ is shown in Figure 5.5. To learn the parallel form adaptive filtering, two second-order sections are connected in parallel form in both reference filter and the adaptive filter. The difference signal of the overall system is used to control the coefficient updating of two adaptive sections. The initial values of two sections are kept the same as in the previous experiments. To compare, the mean-squared errors vs iteration numbers for $H_1(z), H_2(z)$ and the parallel form of $H(z)$ are shown in Figure 5.6. It shows that the convergence rate for the parallel form of two sections is slower than that for the separate cases.

- Initialization.

  In our simulations, the state vectors are initialized to zeroes. We found that the parameters of the adaptive filter have to be initialized properly. First, for the direct form adaptive IIR filters, since one feedback coefficient is set to a constant, i.e., $a_0 = 1$. The parameter vector can be initialized to zero. In adaptive state-space case, if all matrices are initialized to zeroes, only **D** can be updated and the other matrices will remain zero. Second, The most important point is that the matrices **B** and **C** can not be initialized to zeroes. Otherwise, the system is not controllable.

  The initialization values of poles affect the convergence rate significantly. We

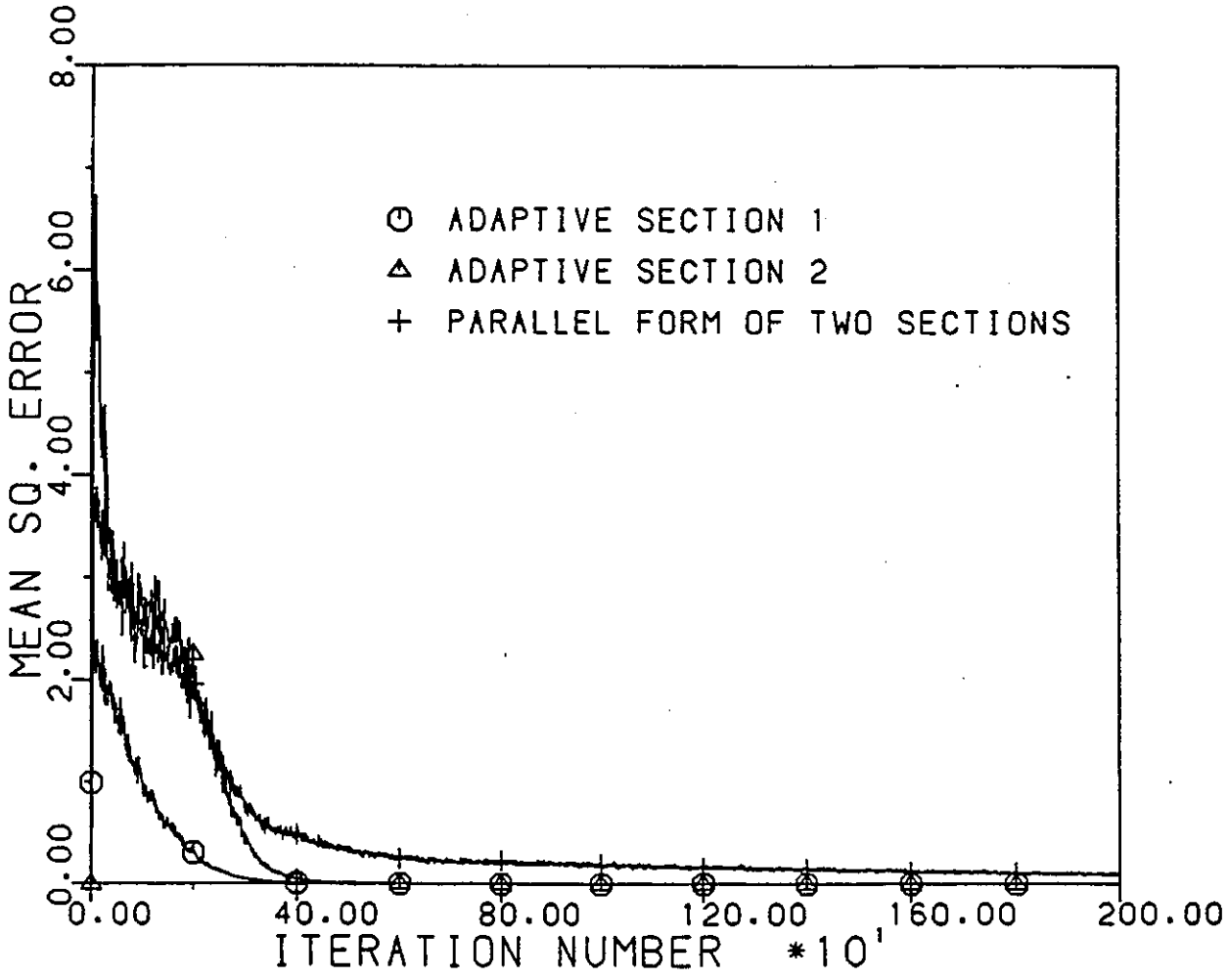



Figure 5.6: Convergence rate of parallel form adaptive state-space filtering.

used three different sets of initial values for the parallel form system identification.

1. The matrices of two sections are initialized to the same values as before. They are:

$$\mathbf{A}_i = \begin{bmatrix} 0.25 & -0.25 \\ 0.25 & 0.25 \end{bmatrix} \quad \mathbf{B}_i = \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix} \quad \mathbf{C}_i^t = \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix} \quad \mathbf{D}_i = 0.0$$

2. Matrices **B** and **C** are not changed, but matrix **A** is changed according to the reference system.

$$\mathbf{A}_{1i} = \begin{bmatrix} -0.25 & -0.25 \\ 0.25 & -0.25 \end{bmatrix}$$

$$\mathbf{A}_{2i} = \begin{bmatrix} 0.25 & -0.25 \\ 0.25 & 0.25 \end{bmatrix}$$

3. The matrices **B** and **C** are also changed.

$$\mathbf{B}_{1i} = \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix} \quad \mathbf{C}_{1i}^t = \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix} \quad \mathbf{B}_{2i} = \begin{bmatrix} -0.25 \\ 0.25 \end{bmatrix} \quad \mathbf{C}_{2i}^t = \begin{bmatrix} 0.25 \\ -0.25 \end{bmatrix}$$

The convergence rates for these three cases are shown in Figure 5.7. It is obviously that when the poles are initialized to close the desired positions, the convergence rate is much faster than the other cases.

## 5.8 VLSI Implementation

In this section, a VLSI architecture for the second-order section is suggested. When the higher order filter is needed, this architecture module is repeated and the error signal will be distributed to each section. For an adaptive state-space filter, the

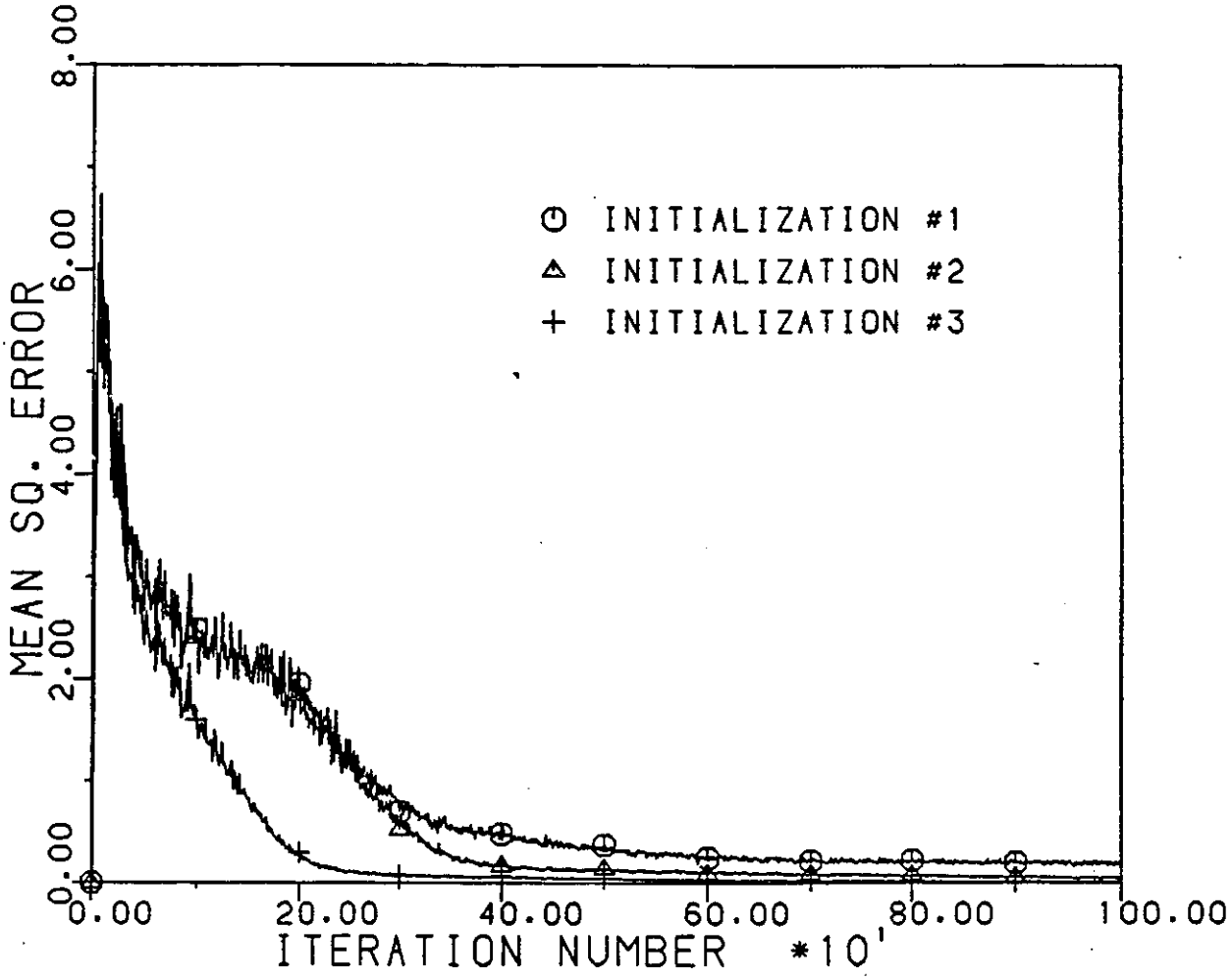



Figure 5.7: Initialization effects on the convergence rate.

computations involved in the filter calculation part and in the parameter update part are matrix operations. During each iteration, the gradients for matrices A and B have to be calculated. The implementation includes three parts:

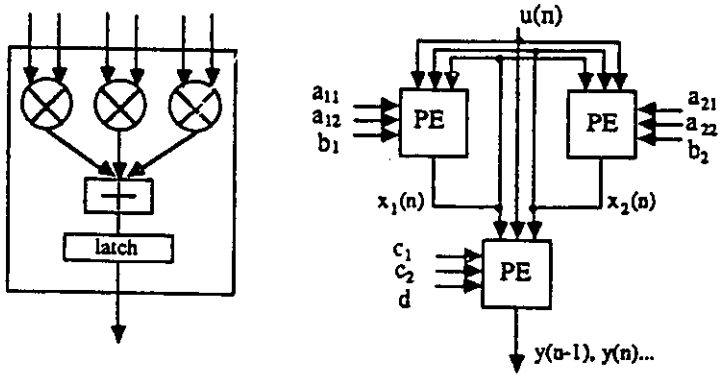
1. Filter Calculation.

   Since the current output value, $y(n)$, depends on the current state variables, $X(n)$, and the current error, $\epsilon(n)$, will be used to control the update of system parameters so that the next output can be adjusted, the state equations used in the process are rewritten here:

$$X(n) = \mathbf{A}(n-1)X(n-1) + \mathbf{B}(n-1)u(n-1) \tag{5.43}$$

$$y(n) = \mathbf{C}(n)X(n) + \mathbf{D}(n)u(n) \tag{5.44}$$

   And, the current state and the output can not be calculated concurrently. Therefore, we can not use a square systolic array to implement this filter part. A two step structure can be used as shown in Figure 5.8. In the first step, the components of state vector are obtained. In the second step, the output value is calculated. All the processing elements are the same and each step needs one clock time.

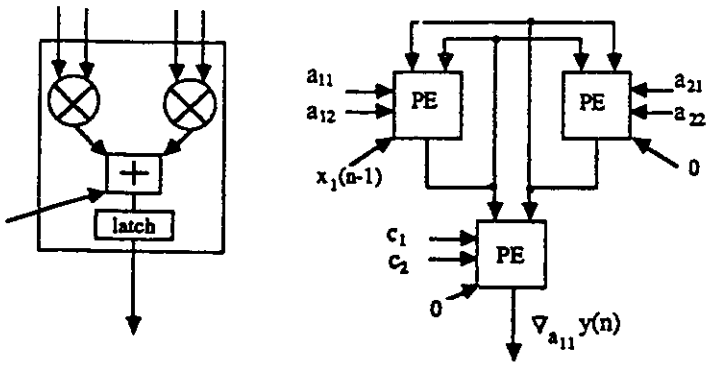



(a) Processing element (b) Array configuration

Figure 5.8: Array processor for state update and output calculation.

2. Gradient Computation.

As indicated in (5.5) and (5.6), the gradient vector of $y$ respect to $W$ has to be found for updating the vector $W$. The gradients of $y$ respect to $\mathbf{C}$ and $\mathbf{D}$ can be obtained directly from the current state $X(n)$ and the current input $u(n)$, respectively. However, as shown in (5.13) and (5.14), the gradients for elements of matrices $\mathbf{A}$ and $\mathbf{B}$ depend on the previous state, $X(n-1)$, previous matrix $\mathbf{A}(n-1)$, and previous gradients. It takes two steps to calculate these gradients. Therefore, they can be computed concurrently with the filter calculations. Similar processing elements as the filter part will be used in this gradient computation part (see Figure 5.9).

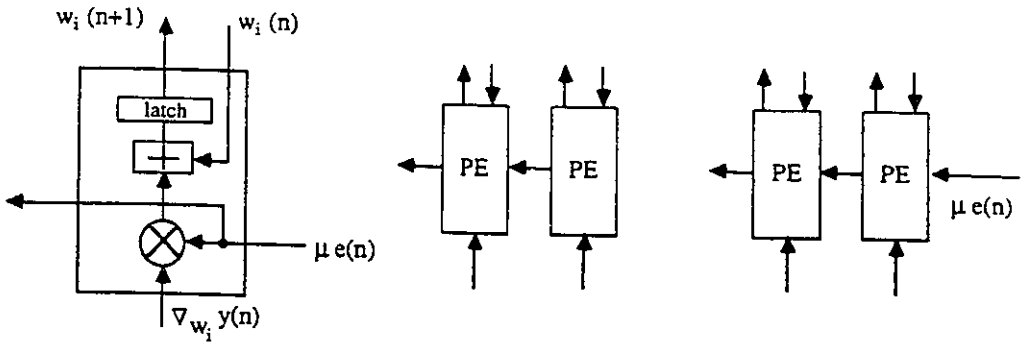



(a) Processing element (b) Array configuration

Figure 5.9: Array processor for gradient calculation.

3. Parameter Update.

When the gradients are ready, the error signal, and the current state variables are also available. The parameter update calculation can be executed in a linear array. In this array, the processing elements perform simple multiplication and summation. It takes one clock time. The error signal is distributed to all PEs so that all the parameters are updated at the same time. This array is given in Figure 5.10.




(a) Processing element

(b) Array configuration

Figure 5.10: The array processor for parameter update.

The overall structure for second order adaptive state-space filter consists of three array processors given above and one delay for input data. Other delays in Figure 5.2 are included in the design of the array processors. As described above, for each iteration, three clock times are required. Based on the current VLSI technique, the throughput rate of 1 sample/100 ns can be expected. If the sufficient condition for stability is checked during the process, we simply add one processing element and one more clock time is needed. For a higher order adaptive state-space filter, the system will be in the parallel form of such second-order structures.

## 5.9 Conclusion

In this chapter, The LMS algorithm is extended to the state-space filtering. The gradients for the parameters in the parameter vector are derived directly from the state equations. When the parallel form is used, the computation complexities for filter operations and for gradient calculations are reduced. The stability monitoring is simple since it is the second-order filter. From the simulation results, it can be seen that the adaptive state-space filters can provide a better numerical performance than the direct form adaptive IIR filters, as in the non-adaptive case. The convergence properties are impressive. Finally, by using the suggested VLSI

processors real-time adaptive filtering can be expected.

# Chapter 6

# Kalman Filtering for Image Restoration and VLSI Implementation

## 6.1 Introduction

Image restoration is an estimation process that attempts to recover a high quality image from its blurred and noised rendition. Restoration is applied to remove (1) systematic degradations such as blurring due to optical system aberrations, atmospheric turbulence, motion and diffraction; and (2) statistical degradations such as noise and measurement errors. These two types of degradations lead to conflicting requirements on the restoration filter. It is difficult to improve image quality by emphasizing high quality spatial frequencies while suppressing noise at the same time. The overall quality of image restoration algorithms greatly benefits from (1) better models for the image formation process, including both systematic and statistical degradations; (2) better modeling of objects to be restored by using nonstationary or local statistical models; (3) better modeling of image quality criteria and properties of the human observer and visual system [129,130].

Kalman filtering has been one of the most widely applied techniques in the area of modern control, signal processing, and communication applications. It is formulated using the state-space approach, in which a dynamical system is described by a set of variables called the state. The state contains all the necessary information about the behavior of the system such that, given the present and future values of the input, we may compute the future state and output of the system [131] recursively. The extension of Kalman filtering to the 2-D case, and its application to image restoration, has been receiving a great deal of attention in recent years. The Kalman filter is considered an optimal estimator because it provides a minimum variance estimate [132]. The main problems in extending the standard 1-D recursive state filtering techniques to the 2-D case are (1) how to establish a suitable 2-D recursive model by defining a proper state vector; (2) how to reduce the dimensionality of the resulting state vectors by reasonable approximation; (3) how to speed up the Kalman filtering procedure by processing signals in parallel.

Considerable effort has been devoted to image modeling and reduction of the orders of the Kalman filter for image restoration. Early applications of Kalman filtering to the restoration of blurred and noisy images were restricted to 1-D Kalman filters (processing each line independently) using 1-D image models [133,134]. In [135] Nahi and Assefi modeled images by state-space techniques. Aboutalib and Silverman [133] considered the case of images that are degraded by linear motion blur and additive noise. The original image is modeled as the output of a line scanner, and the blurring process is modeled by a 1-D linear dynamic model. Later, this approach was extended to the general motion blur [134]. Woods and Rademan [136] formulated a 2-D scalar Kalman processor, called the reduced update Kalman filter (RUKF), using a 2-D AR image model with nonsymmetric half plane (NSHP) causal support. The RUKF scheme was shown to offer significant reduction in the total computation load. Woods and Ingle [137] then extended the RUKF to the case of degradation due to both blur and noise. Murphy and Silverman [138] formulated

a vector Kalman filter based on a 2-D semicausal, Gauss-Markov image model, where the filtering is performed one line at a time. More recently, Angwin and Kaufman [139] proposed the reduced order model Kalman filter (ROMKF), which uses a NSHP image model and is based on a low-order state-space model of an image. The low dimension of this system results in decreased computation time. In addition, several other authors have proposed different 2-D Kalman filtering schemes for restoration of images degraded by both blur and noise. For example, Suresh and Shenoi [140] proposed the Kalman strip filtering algorithm, in which the blur was modeled by a 2-D state-space structure. Wu [141] employed three-dimensional state-space models to develop another strip filtering model for the degraded image with a nonsymmetric half-plane support. Azimi-Sadjadi and Wong [142] presented a two-dimensional block Kalman filtering. This 2-D block state-space model takes into account the correlations of the image data in successive neighboring blocks and reduces the edge effects. However, the optimal Kalman filter for strip observations as well as the one for the block observations are characterized by complexities and large computational requirements.

The processing of a Kalman filter requires matrix/vector operations such as multiplication, addition, subtraction, and inversion. Among these, matrix inversion is the most difficult to implement. Especially in the 2-D case, the dimensions of the matrices are much larger than the 1-D case. Fortunately, with the rapid development of VLSI integrated circuits, it is feasible to implement Kalman filters by parallel array architectures. Recently, VLSI implementations of the Kalman filters have been given by several authors [2,143,144,145,146,132]. The results in [143] and [144] handled only the measurement updating, while [145,146] proposed array designs for both the measurement and time updating. Sung in [145] presented a parallel implementation of the square-root Kalman filters, in which systolic-type VLSI processor arrays were used as basic building blocks to accelerate the matrix operations and an inter-array pipelining scheme was used through the overlapping

of execution between successive processor arrays. Kung in [2] exploited the inherent triangular structure of the matrix and derived another systolic array architecture. In [132], Yeh provided a method to map Kalman filter to Faddeev algorithm, and then map this algorithm to systolic processors. However, up to now, it is still a problem to realize the Kalman filter for image restoration by simple means and at real-time speed.

In this chapter, a state-space Kalman filter which models both the image and the linear spatial invariant (LSI) blur by 2-D state-space structures is proposed, and a VLSI implementation for the diagonal scanning scheme is presented [147,148]. We start with a brief review of digital image restoration. Next, for simplifying the filtering procedure, 2-D state-space structures for autoregressive (AR) image generation model and LSI blur model with the quarter-plane region of support are introduced. Then, these two state-space structures are cascaded to form a composite state-space dynamic model and the Kalman filter equations are established. Finally, to achieve a real-time processing speed, an efficient VLSI implementation with the diagonal scheme is presented. The experiments are given to show that the proposed algorithm and the implementation could be very useful for real-time image restoration.

## 6.2 Image Restoration

The images acquired by electronic, electron-optical, or optical means are often suffering degradations. The degradations may take different forms such as blur caused by atmospheric turbulence, diffraction, motion, misfocus, sensor noise, measurement errors, etc. It is necessary to improve the quality of the images for many applications.

Image restoration is one of the commonly used technologies for image improvement [132,149,150,151]. It tries to filter the observed images and to approximate

an ideal degradation-free image as closely as possible. In general, there are three steps to obtain a restored image. The first step is to establish the models of the image. The second is to identify the parameters of the models from some available set of similar images. The third one is then to recover the image by using certain algorithm based on the models and the parameters. When the characteristics of the blur, measurement noise and image generation process are known, it is then possible to use this priori knowledge to filter the degraded image. When sample images are not available, or images are nonhomogeneous, or image models are not necessarily true autoregressive fields, an adaptive estimator has to be used to identify the parameters and estimate the images simultaneously. In this research, we define the image restoration problem as follows: Given a noisy and blurred image, find an estimate of the ideal image using a priori information about the blur and/or the noise and/or the ideal image.

In many practical imaging situations, blur is a distortion which affects the image in a deterministic manner, while noise is a stochastic phenomenon which corrupts the image. Therefore, the observed image pixels for a blurred and noisy image can be modeled as the output of a linear filter, which is described by

$$y(m,n) = \Sigma_{i,j\in R_1} h_{ij}(m,n) f(m-i,n-j) + v(m,n) \tag{6.1}$$

where $h_{ij}(m,n)$ represents the space-varying degradation function or point spread function (PSF) with support denoted by $R_1$; $f(m,n)$ represents the uncorrupted image of size $M \times N$; $y(m,n)$ is the observed image; and $v(m,n)$ represents the observation noise. The noise which is a stochastic phenomenon, in most practical situation may be considered to be white Gaussian. For LSI blur, equation (6.1) can be written as:

$$y(m,n) = \Sigma_{i,j\in R_1} h_{ij} f(m-i,n-j) + v(m,n) \tag{6.2}$$

The original image can be modeled as an autoregressive Gauss-Markov process

driven by uncorrelated Gaussian white noise, described by

$$f(m,n) = \Sigma_{k,l \in R_2} c_{kl} f(m-k, n-l) + u(m,n) \tag{6.3}$$

In this equation, $f(m,n)$ represents the original image; $c_{kl}$ represents space-invariant model coefficients; $u(m,n)$ is the noise process accounting for the error in the model; and $R_2$ represents the support region. The block diagram representation of the input-output model of the degraded images is given in Figure 6.1. Here, we assume the parameters of the image model, the blur model and the noise are known. The image restoration problem is then to estimate $f(m,n)$ from the observed image $y(m,n)$ according to some optimality criterion.

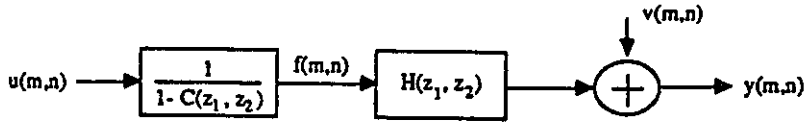



Figure 6.1: Block diagram representation of the image generation model and the degradation model

Most commonly used filters for image restoration are: inverse/pseudoinverse filters, Wiener filter, FIR filter, recursive (Kalman) filter, etc. It has been shown that the Kalman filter is an optimal estimator for image restoration. In the following sections, we will construct a state-space blur model and a state-space image generation model based on the models used here and incorporate the blurred model and image model into a state dynamic model. The state-space Kalman filter is then established and the system matrices are derived according to the blur and image parameters.

## 6.3 State-Space Modeling

### 6.3.1 Image Generation Model

In order to use Kalman filter to restore the image, we first have to know the correlations between pixels of the original image. When the previous pixels are estimated, the current pixel can be predicted based on the correlation. Then, using the current observation, we correct the prediction to get the optimal estimation.

The image generation model has been proposed by several authors. Basically, the image is modeled as an autoregressive (AR) Gaussian-Markov process driven by uncorrelated Gaussian white noise as shown in equation (6.3). If the AR model support is causal, where causality is defined on the basis of pixel-by-pixel scanning of images from left to right and top to bottom, the value at pixel location $(m,n)$ (the present) is expressed in terms of the values of the pixels scanned up to this location (the past). The nonsymmetric half plane causal support and semicausal support (quarter-plane) are widely used [138,152]. Here, the Roesser's 2-D state-space model [92] will be used to represent the image AR model with a quarter-plane region of support.

Consider an image process that starts from the upper-left-hand corner of the image and then proceeds horizontally line-by-line. The current output pixel of the image process only depends on the past pixels [153]. Assume the image pixels are produced recursively by a first-quadrant causal system (see Figure 6.2).

Then, we have Roesser's state equations for equation 6.3

$$\begin{bmatrix} R_u(m+1,n) \\ S_u(m,n+1) \end{bmatrix} = \begin{bmatrix} A_u^{11} & A_u^{12} \\ A_u^{21} & A_u^{22} \end{bmatrix} \begin{bmatrix} R_u(m,n) \\ S_u(m,n) \end{bmatrix} + \begin{bmatrix} B_u^1 \\ B_u^2 \end{bmatrix} u(m,n); \quad (6.4)$$

$$f(m,n) = \begin{bmatrix} C_u^1 & C_u^2 \end{bmatrix} \begin{bmatrix} R_u(m,n) \\ S_u(m,n) \end{bmatrix} + D_u u(m,n). \quad (6.5)$$

where $m = 0,1,...,M-1$, $n = 0.1,....,N-1$. $u(m,n)$ and $f(m,n)$ are the driven noise and the output image, respectively. $R_u(m,n)$ is the horizontal state-space vector

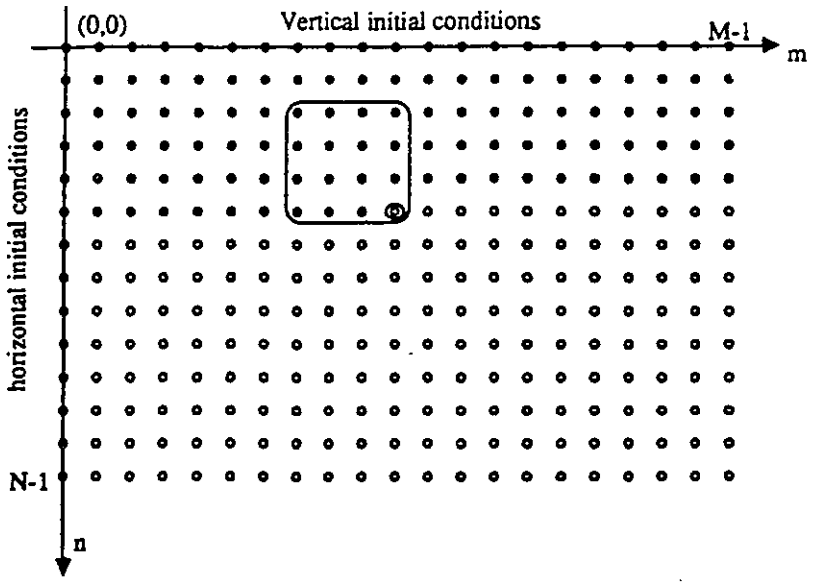



Figure 6.2: Input-output relationship of an image process with a first-quadrant support region $R_2$.

component of dimension $m_u \times 1$, and $S_u(m,n)$ is the vertical component of dimension $n_u \times 1$. $\mathbf{A}_u, \mathbf{B}_u, \mathbf{C}_u$, and $\mathbf{D}_u$ are $(m_u+n_u)\times(m_u+n_u), (m_u+n_u)\times 1, 1\times(m_u+n_u)$, and $1 \times 1$ matrices. $m_u$ and $n_u$ are the number of dependent pixels respectively in the horizontal and in the vertical directions related to the current pixel. Notice that the current state vector is calculated by the following equation:

$$\begin{bmatrix} R_u(m,n) \\ S_u(m,n) \end{bmatrix} = \mathbf{A}^{10} \begin{bmatrix} R_u(m-1,n) \\ S_u(m-1,n) \end{bmatrix} + \mathbf{A}^{01} \begin{bmatrix} R_u(m,n-1) \\ S_u(m,n-1) \end{bmatrix} + \mathbf{B}^{11} \begin{bmatrix} u(m-1,n) \\ u(m,n-1) \end{bmatrix} \tag{6.6}$$

where

$$\mathbf{A}^{10} = \begin{bmatrix} A_u^{11} & A_u^{12} \\ 0 & 0 \end{bmatrix}, \tag{6.7}$$

$$\mathbf{A}^{01} = \begin{bmatrix} 0 & 0 \\ A_u^{21} & A_u^{22} \end{bmatrix}, \tag{6.8}$$

$$\mathbf{B}^{11} = \begin{bmatrix} B_u^1 & 0 \\ 0 & B_u^2 \end{bmatrix}. \tag{6.9}$$

When the image coefficients in equation (6.3) are found, the system matrices in (6.4) and (6.5) can be easily derived by using a signal flow graph mapping method [148]. For example, if

$$\begin{aligned} f(m,n) &= c_{10}f(m-1,n) + c_{01}f(m,n-1) + \\ &\quad c_{11}f(m-1,n-1) + u(m,n), \end{aligned} \tag{6.10}$$

and the state variables are chosen as the outputs of the delay operators, as shown in the signal flow graph Figure 6.3, the matrices $\mathbf{A}_u, \mathbf{B}_u, \mathbf{C}_u$, and $\mathbf{D}_u$ for Roesser's model are as follows:

$$\mathbf{A}_u = \begin{bmatrix} A_u^{11} & A_u^{12} \\ A_u^{21} & A_u^{22} \end{bmatrix} = \begin{bmatrix} c_{10} & 1 \\ c_{11} & c_{01} \end{bmatrix}; \tag{6.11}$$

$$\mathbf{B}_u = \begin{bmatrix} B_u^1 \\ B_u^2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \tag{6.12}$$

$$\mathbf{C}_u = \begin{bmatrix} C_u^1 & C_u^2 \end{bmatrix} = \begin{bmatrix} c_{10} & 1 \end{bmatrix}; \tag{6.13}$$

$$\mathbf{D}_u = D_u = 1. \tag{6.14}$$

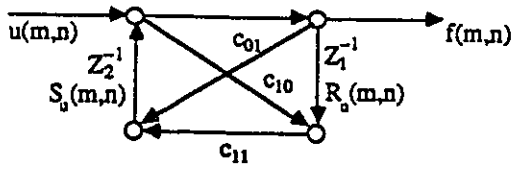



Figure 6.3: Signal flow graph of an AR image model for a first-quadrant support region with first order

### 6.3.2 LSI Blur Model

In this subsection, we will discuss the LSI blur model or the observation model. Notice that $f(m,n)$ is the input and $y(m,n)$ is the output of the system now. The

image restoration algorithm is based on an observation model that establishes the relationship between the input (ideal image) and the output (observed degraded image) of the imaging system. The success of image restoration depends on how good the assumed mathematical model fits the input/output characteristics of the imaging system. Our aim now is to obtain a state-space model of blurs, the deterministic degradation from equation 6.2. 2-D state-space models for linear image processing have been proposed by several authors. Here, we will use Roesser's 2-D state-space model again since it is more general one. Notice that $f(m,n)$ is the scalar input, the pixel element generated by the image process mentioned above. $y(m,n)$ represents the blurred pixel at the same location. A vector $R_f(m,n)$ is defined as the horizontal state, which is assumed to convey information in the horizontal direction. Similarly, a vector $S_f(m,n)$ can be chosen as the vertical state to convey information in the vertical direction. The vectors $R_f(m,n)$ and $S_f(m,n)$ together are called as the local state since they are propagated locally and can only determine the output pixels next to them. For a infinite impulse response (IIR) system or a finite impulse response (FIR) system, the state vectors can be chosen as the outputs of the delay operators. In general Roesser's local state-space equations are:

$$\begin{bmatrix} R_f(m+1,n) \\ S_f(m,n+1) \end{bmatrix} = \begin{bmatrix} A_f^{11} & A_f^{12} \\ A_f^{21} & A_f^{22} \end{bmatrix} \begin{bmatrix} R_f(m,n) \\ S_f(m,n) \end{bmatrix} + \begin{bmatrix} B_f^1 \\ B_f^2 \end{bmatrix} f(m,n); \quad (6.15)$$

$$y'(m,n) = \begin{bmatrix} C_f^1 & C_f^2 \end{bmatrix} \begin{bmatrix} R_f(m,n) \\ S_f(m,n) \end{bmatrix} + D_f f(m,n). \quad (6.16)$$

where, $\mathbf{A}_f, \mathbf{B}_f, \mathbf{C}_f$, and $\mathbf{D}_f$ are system matrices of appropriate dimensions.

For a known linear spatial invariant blur, $h_{ij}$, a local state-space realization can be obtained by inspecting the relationship between the input, the output and the state vectors of the signal flow graph. Then, given values for the boundary conditions (such as all zero) and the input $f(m,n)$, the equations produce an output state vector and an output pixel value, and the image is processed recursively.

## 6.4 Kalman Filter Formulation

### 6.4.1 Kalman Filters

A Kalman filter determines the causal least mean square error (LMSE) estimate recursively. It is based on state-space representation of the dynamic system [154]. In the case of image restoration, image data are used to define the state vectors.

Here we consider a discrete dynamic imaging system represented by:

$$X(m+1,n) = \mathbf{A}X(m,n) + \mathbf{G}W(m,n) + \mathbf{H}U(m,n) \tag{6.17}$$

$$y(m,n) = \mathbf{C}X(m,n) + v(m,n) \tag{6.18}$$

where $X(m,n)$ represents the state vector; $W(m,n)$ denotes a deterministic input vector; $U(m,n)$ is a noise vector; and $\mathbf{A}, \mathbf{C}, \mathbf{G}$ and $\mathbf{H}$ are system matrices. The terms $v(m,n)$ and $y(m,n)$ have been defined previously. The representation of equation (6.17) is a "one-dimensional" state-space difference equation since the state is propagated horizontally.

In equations (6.17) and (6.18) $U(m,n)$ and $v(m,n)$ are uncorrelated zero-mean white noise processes, with the following second-order properties:

$$\mathbf{Q}_u = E(U(m,n)U(m,n)^t); \qquad Q_v = E(v(m,n)v(m,n)^t).$$

When a Kalman filter is used, the estimation of the state vector, $\hat{X}(m,n)$, is obtained. The error covariance $P(m,n)$ is then defined as

$$\mathbf{P}(m,n) = E(e(m,n)e(m,n)^t),$$

where $e(m,n) = X(m,n) - \hat{X}(m,n)$. Given an a priori estimate of the initial state $\hat{X}(0,0)$ and the state error covariance $\mathbf{P}(0,0)$, and given the a priori statistical information of the input noise covariance $\mathbf{Q}_u(m,n)$ and measurement noise covariance $Q_v(m,n)$, an estimate of the state of the system can be obtained sequentially by the Kalman filter, which consists of a predictor and a corrector shown in Figure 6.4.

$K(m,n)$ is the Kalman gain, adaptive to minimize the error covariance $\mathbf{P}(m,n)$. The calculation of $K(m,n)$ is given by

$$\mathbf{P}(m,n|m-1,n) = \mathbf{A}\mathbf{P}(m-1,n|m-1,n)\mathbf{A}^t + \mathbf{H}\mathbf{Q}_u(m-1,n)\mathbf{H}^t \quad (6.19)$$

$$K(m,n) = \mathbf{P}(m,n|m-1,n)\mathbf{C}^t \{\mathbf{C}\mathbf{P}(m,n|m-1,n)\mathbf{C}^t + Q_v(m,n)\}^{-1} \quad (6.20)$$

$$\mathbf{P}(m,n|m,n) = \{\mathbf{I} - K(m,n)\mathbf{C}\}\mathbf{P}(m,n|m-1,n) \quad (6.21)$$

where $\mathbf{P}(m,n|m-1,n)$ is the predicted covariance matrix for $(m,n)$ determined by $\mathbf{P}(m-1,n|m-1,n)$, $\mathbf{Q}(m-1,n)$ and system matrices $\mathbf{A},\mathbf{H}$. $\mathbf{P}(m,n|m,n)$ is the error covariance matrix for $(m,n)$, same as $\mathbf{P}(m,n)$. The notation of $\mathbf{P}(m,n|m,n)$ indicates that it is the final correction of the estimation $\mathbf{P}(m,n|m-1,n)$.

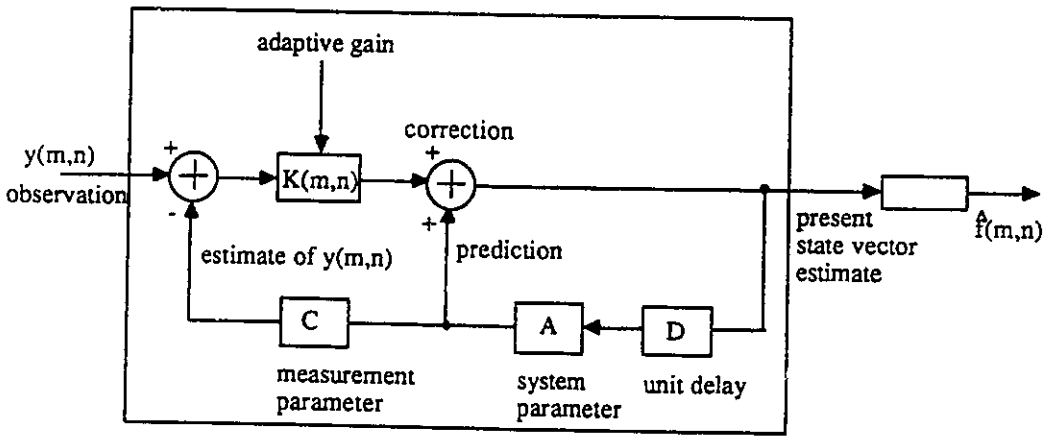



Figure 6.4: Kalman filter for image restoration.

### 6.4.2 State-Space Kalman Filter

To establish a 2-D state-space Kalman filter, we incorporate the image generation model and the LSI blur model into a composite state dynamic model, i.e. a state update equation and a state-dependent output equation. In other words, we have

to determine the elements of the state vector for the whole system such that the similar equations as 6.17 and 6.18 can be obtained.

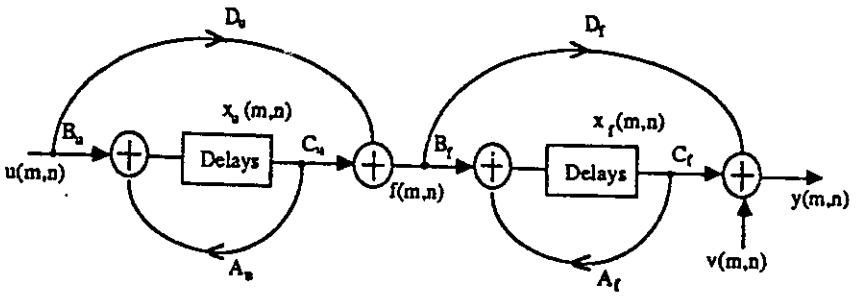



Figure 6.5: The cascade of 2-D state-space models. The first one is for image generation and the second one is for LSI blur model.

As shown in Figure 6.5, the image generation model and the blur model are cascaded to form the whole system, and a white noise $v(m,n)$ is added as the observation noise. Let

$$R(m,n) = \begin{bmatrix} R_u(m,n) \\ R_f(m,n) \end{bmatrix} \tag{6.22}$$

$$S(m,n) = \begin{bmatrix} S_u(m,n) \\ S_f(m,n) \end{bmatrix} \tag{6.23}$$

thus, we have

$$\begin{bmatrix} R(m+1,n) \\ S(m,n+1) \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} R(m,n) \\ S(m,n) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u(m,n) \tag{6.24}$$

$$y(m,n) = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} R(m,n) \\ S(m,n) \end{bmatrix} + Du(m,n) + v(m,n) \tag{6.25}$$

where,

$$A_{11} = \begin{bmatrix} A_u^{11} & 0 \\ B_f^1 C_u^1 & A_f^{11} \end{bmatrix}, A_{12} = \begin{bmatrix} A_u^{12} & 0 \\ B_f^1 C_u^2 & A_f^{12} \end{bmatrix}$$

$$
\begin{aligned}
A_{21} &= \begin{bmatrix} A_u^{21} & 0 \\ B_f^2 C_u^1 & A_f^{21} \end{bmatrix}, A_{22} = \begin{bmatrix} A_u^{22} & 0 \\ B_f^2 C_u^2 & A_f^{22} \end{bmatrix} \\
B_1 &= \begin{bmatrix} B_u^1 \\ B_f^1 D_u \end{bmatrix}, B_2 = \begin{bmatrix} B_u^2 \\ B_f^2 D_u \end{bmatrix} \\
C_1 &= \begin{bmatrix} D_f C_u^1 & C_f^1 \end{bmatrix}, C_2 = \begin{bmatrix} D_f C_u^2 & C_f^2 \end{bmatrix} \\
D &= D_f D_u
\end{aligned}
$$

The elements of these matrices are defined in equations (6.4)(6.5) and (6.15)(6.16).

By comparing equations (6.17)(6.18) and equations (6.24)(6.25), we find that there are two problems to be solved. First, the input random process $u(m,n)$ appears in the observation equation of (6.25) but not in (6.18). Thus, the equations of the conventional Kalman filter gain and the error covariance matrix can not be used directly. Second, since the local state propagates horizontally and vertically, the state at $(m+1,n)$th position depends on the states of positions $(m,n)$ and $(m+1,n-1)$. Therefore, we modify the state vectors of the system by including $u(m,n)$ and the state vectors for $(m,n)$ and $(m+1,n-1)$ in it. We define:

$$
R(k) = \begin{bmatrix} R(m,n) \\ R(m+1,n-1) \end{bmatrix} \tag{6.26}
$$

$$
S(k) = \begin{bmatrix} S(m,n) \\ S(m+1,n-1) \end{bmatrix} \tag{6.27}
$$

$$
U_1(k) = \begin{bmatrix} u(m,n) \\ u(m+1,n-1) \end{bmatrix} \tag{6.28}
$$

$$
R(k+1) = \begin{bmatrix} R(m+1,n) \\ R(m+2,n-1) \end{bmatrix} \tag{6.29}
$$

$$
S(k+1) = \begin{bmatrix} S(m+1,n) \\ S(m+2,n-1) \end{bmatrix} \tag{6.30}
$$

$$U_1(k+1) = \begin{bmatrix} u(m+1,n) \\ u(m+2,n-1) \end{bmatrix} \tag{6.31}$$

$$U_2(k) = \begin{bmatrix} u(m+1,n) \\ u(m+2,n-1) \end{bmatrix} \tag{6.32}$$

The state vectors are then defined as

$$X(k) = \begin{bmatrix} R(k) \\ S(k) \\ U_1(k) \end{bmatrix} \tag{6.33}$$

$$X(k+1) = \begin{bmatrix} R(k+1) \\ S(k+1) \\ U_1(k+1) \end{bmatrix} \tag{6.34}$$

Now, the problem left is that the state vector $X(k+1)$ can not be represented in terms of $X(k)$ since $R(m+2,n-1)$ and $S(m+2,n-1)$ are not completed related to $X(k)$. We approximate these values by their most recent estimates with the uncertainty represented in a noise term. Let

$$X_2(k) = \begin{bmatrix} R(m+2,n-1) \\ S(m+2,n-1) \end{bmatrix} \tag{6.35}$$

$$W(k) = \begin{bmatrix} \hat{R}(m+2,n-1) \\ \hat{S}(m+2,n-1) \end{bmatrix} \tag{6.36}$$

Then we have:

$$X_2(k) = W(k) + W_2(k) \tag{6.37}$$

where $W(k)$ indicates the best available estimate of the state vector, $X_2(k)$, and $W_2(k)$ is the estimation noise included to account for uncertainty in this approximation. It follows that equations 6.24 and 6.25 can be changed to:

$$X(k+1) = \mathbf{A}X(k) + \mathbf{G}X_2(k) + \mathbf{F}U_2(k) \tag{6.38}$$

$$y(k) = \mathbf{C}X(k) + v(k) \tag{6.39}$$

where

$$\mathbf{A} = \begin{bmatrix} A_{11} & 0 & A_{12} & 0 & B_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & A_{21} & 0 & A_{22} & 0 & B_2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.40}$$

$$\mathbf{G} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}^t \tag{6.41}$$

$$\mathbf{F} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^t \tag{6.42}$$

$$\mathbf{C} = \begin{bmatrix} C_1 & 0 & C_2 & 0 & D & 0 \end{bmatrix} \tag{6.43}$$

where $^t$ denotes the transpose. Let $m_u$ and $n_u$ be the numbers of delays in the horizontal and vertical directions in the image generation model, respectively. Let $m_f$ and $n_f$ be the numbers of delays in the horizontal and vertical directions in the linear blur model, respectively. The dimensions of $R(k), S(k)$ and $U_1(k)$ are $2(m_u + m_f) \times 1$, $2(n_u + n_f) \times 1$ and 2, respectively. The dimensions of matrices $A, G, F$ and $C$ are $2(m_u + m_f + n_u + n_f + 1) \times 2(m_u + m_f + n_u + n_f + 1), 2(m_u + m_f + n_u + n_f + 1) \times (m_u + m_f + n_u + n_f), 2(m_u + m_f + n_u + n_f + 1) \times 2$, and $1 \times 2(m_u + m_f + n_u + n_f + 1)$, respectively. For the submatrices, the dimension of $A_{11}$ is $(m_u + m_f) \times (m_u + m_f)$, of $A_{12}$ is $(n_u + m_f) \times (n_u + n_f)$, of $A_{21}$ is $(n_u + n_f) \times (m_u + m_f)$, of $A_{22}$ is $(n_u + n_f) \times (n_u + n_f)$, of $B_1$ is $(m_u + m_f) \times 1$ and $B_2$ is $(n_u + n_f) \times 1$, respectively. Finally, by substituting $W(k)$ and $W_2(k)$ for $X_2(k)$, define $U(k) = \begin{bmatrix} W_2(k) & U_2(k) \end{bmatrix}^t$, and $\mathbf{H} = \begin{bmatrix} \mathbf{G} & \mathbf{F} \end{bmatrix}$, the desired equations as 6.17 and 6.18 are obtained:

$$X(k+1) = \mathbf{A}X(k) + \mathbf{G}W(k) + \mathbf{H}U(k) \tag{6.44}$$

$$y(k) = \mathbf{C}X(k) + v(k) \tag{6.45}$$

Notice that although the representation of equation is a one-dimensional state-space difference equation, actually, the state propagates on both horizontal and vertical directions due to the definition of the state.

Now, we can use the conventional Kalman filtering formula to estimate the state $X(k)$ and restore the image as follows.

$$\hat{X}(k|k-1) = \mathbf{A}\hat{X}(k-1|k-1) + \mathbf{G}W(k) \quad (6.46)$$

$$\mathbf{P}(k|k-1) = \mathbf{A}\mathbf{P}(k-1|k-1)\mathbf{A}^t + \mathbf{H}\mathbf{Q}_u(k-1)\mathbf{H}^t \quad (6.47)$$

$$K(k) = \mathbf{P}(k|k-1)\mathbf{C}^t\{\mathbf{C}\mathbf{P}(k|k-1)\mathbf{C}^t + Q_v(k)\}^{-1} \quad (6.48)$$

$$\mathbf{P}(k|k) = \{\mathbf{I} - K(k)\mathbf{C}\}\mathbf{P}(k|k-1) \quad (6.49)$$

$$\hat{X}(k|k) = \hat{X}(k|k-1) + K(k) \times \{y(k) - \mathbf{C}\hat{X}(k|k-1)\} \quad (6.50)$$

$$\hat{f}(k) = \begin{bmatrix} C_u^1 & C_u^2 \end{bmatrix} \begin{bmatrix} \hat{R}_u(k) \\ \hat{S}_u(k) \end{bmatrix} \quad (6.51)$$

where $\hat{f}(k)$ is the optimal estimate of the image. The index $k$ is equal to $nM+m$, in other words, this representation is based on the line-by-line scanning method. We refer to this algorithm as state-space Kalman filter with the line scanning (SSKFL).

## 6.5 Parallel VLSI Implementation

As shown above, the Kalman filter is computationally intensive since in each estimation, many matrix operations are performed. As described by equations (6.46)-(6.51), the filter is a scalar processor. The result of $\{\mathbf{C}\mathbf{P}(k-1|k-1)\mathbf{C}^t + \mathbf{Q}_v\}^{-1}$ is a scalar. As a result, the matrix inversion is avoided. However, matrix-matrix multiplications and matrix-vector multiplications still prevent the potential application of this Kalman filter to high speed processing.

In order to accelerate the Kalman filter processing speed, highly parallel VLSI systolic structures have been proposed by several authors to perform the Kalman filtering as mentioned before. Notice that the Kalman filter is a recursive filter. The

estimation of $X(k+1)$ can not be started before the estimation of $X(k)$ has been finished. Without special arrangement, even though the systolic arrays can be obtained to perform the matrix operations by using the direct mapping method given in chapter 2, the throughput rate and the utilization of the array are limited. In this section we will give a dedicated fast implementation for our proposed algorithm. The implementation of this Kalman filtering consists of two array processors: one is for the state vector update, and the other one is for the Kalman gain calculation.

Before we give the design of array architectures, two important features have to be discussed. First, in real processing, not all components of the state vector $X(k)$ have to be updated. For example, only the state components for pixel at $(m,n)$ need to be calculated, while the state components for pixel at $(m+1,n-1)$ are already generated. Hence, we will save the state vector components $R(m+1,n-1)$ and $S(m+1,n-1)$ into a buffer when they are generated, and retrieve them from the buffer when they are needed. Therefore, the computations for equations 6.46 and 6.50 can be reduced. For the same reason, the implementations of 6.47, 6.48 and 6.49 can also be simplified.

Second, for the first-quadrant image model, the pixels along the diagonal lines are computationally independent. They can be processed concurrently. Thus, to achieve high speed and high utilization of the systolic array structures, the diagonal scanning method can be used [74]. That is, image is divided into horizontal strips. The width of the strip, $W$, depends on the number of delays between the input and the output of the implementation, which is given by:

$$W = \max(\textit{latency of state update, the latency of Kalman gain calculation}).$$

Let us consider the process of one diagonal segment within one strip, which starts from the nth line. As shown in Figure 6.6, when the state vector for pixel $(m-1,n)$, $X(m-1,n) = \begin{pmatrix} R(m-1,n) & S(m-1,n) & U_1(m-1,n) \end{pmatrix}^t$, enters the array processor, the prediction of the state component, $\hat{R}(m,n)$ can be calculated. At the same time, $\hat{S}(m,n)$ is generated based on the initialized or previous estimated

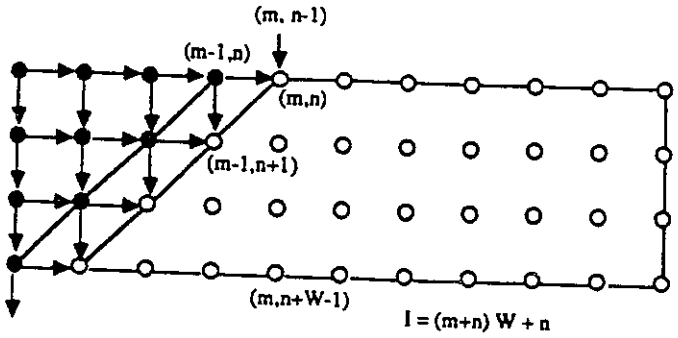



Figure 6.6: The diagonal estimate process within each horizontal strip (W=4).

values of the state vectors of pixel $(m, n-1), X(m, n-1)$. Next, $X(m-2, n+1)$ is ready to enter the array for producing $\hat{R}(m-1, n+1)$. And, at same time, the state vector $X(m-1, n)$ can be used to calculate the vertical component $\hat{S}(m-1, n+1)$, and so on. When the boundary of the strip is reached, the generated state vectors will be stored in buffers for processing the next strip. The estimated state vectors of the last strip stored in the buffers are popped to the array for processing the first pixel of the next diagonal segment. It takes one cycle, which is equal to $W$ clock times, to process one diagonal segment. It is clear that every cycle one initial state vector for the first line of the strip needs to be retrieved from the storage, one generated state vector for the last line has to be stored. This means $X(k), X(k+(M-1)), X(k+2(M-1)), ..., X(k+(W-1)(M-1))$ can be estimated concurrently. This can be more clearly explained by introducing a new index $I$. If $m = 0, 1, ...M-1$ is the column coordinate and $n = 0, 1, ...W-1$ is the line coordinate for one strip, the index $I$ is equal to $(m+n)W+n$ within each strip. If the entire image is divided into $S$ strips (as shown in Figure 6.7), then the estimation index $I'$ will be given by

$$I' = [m + \lfloor \frac{n}{W} \rfloor M + (n \; Mod \; W)]W + (n \; Mod \; W).$$

For the new index $I$, equations 6.46 to 6.51 will be rewritten as follows:

$$\hat{X}(I|I-W) = \mathrm{A}\hat{X}(I-W|I-W) + \mathrm{G}W(I) \tag{6.52}$$

$$\mathbf{P}(I|I-W) = \mathbf{A}\mathbf{P}(I-W|I-W)\mathbf{A}^t + \mathbf{H}\mathbf{Q}_u(I-W)\mathbf{H}^t \quad (6.53)$$

$$K(I) = \mathbf{P}(I|I-W)\mathbf{C}^t\{\mathbf{C}\mathbf{P}(I|I-W)\mathbf{C}^t + Q_v(I)\}^{-1} \quad (6.54)$$

$$\mathbf{P}(I|I) = \{\mathbf{I} - K(I)\mathbf{C}\}\mathbf{P}(I|I-W) \quad (6.55)$$

$$\hat{X}(I|I) = \hat{X}(I|I-W) + K(I) \times \{y(I) - \mathbf{C}\hat{X}(I|I-W)\} \quad (6.56)$$

$$\hat{f}(I) = \begin{bmatrix} C_u^1 & C_u^2 \end{bmatrix} \begin{bmatrix} \hat{R}_u(I) \\ \hat{S}_u(I) \end{bmatrix} \quad (6.57)$$

According to this expression, $X(I), X(I+1), ..., X(I+W-1)$ can be estimated concurrently or sequentially without waiting time. After a delay of $W$ clock times, $X(I+W), X(I+W+1), ..., X(I+W+W-1)$ can be estimated in the same way. For a high utilization of the array, we will estimate the state vectors, $X(I), X(I+1), ..., X(I+W-1)$ sequentially. This algorithm is referred to as the state-space Kalman filter with the diagonal scanning (SSKFD).

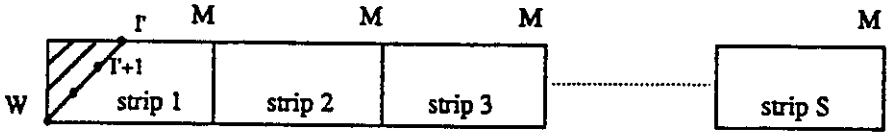



Figure 6.7: The index I' for the entire image which is divided into S strips

Third, the Kalman gain evaluation is determined by the model parameters and the statistics of the random process, $\mathbf{Q}_u$ and $Q_v$. Hence, $K(k)$ can be calculated before estimation is carried out since it does not depend on the measurements [154]. Furthermore, since the Kalman gain converges to a steady-state solution after a certain number of iterations, it can be evaluated off-line until it converges and then can be used in the real-time state estimation. When the Kalman gain is needed to be calculated on-line as the estimation proceeds, it is time consuming to implement 6.47, 6.48 and 6.49 since they can not be performed in parallel. Fortunately, when the diagonal estimation scheme is used, the gain $K(I)$ is related to $K(I-W)$ not to $K(I-1)$. The Kalman gains, $K(I), K(I+1), ..., K(I+W-1)$ can be calculated

in parallel or sequentially without waiting time based on the results of the previous diagonal segment.

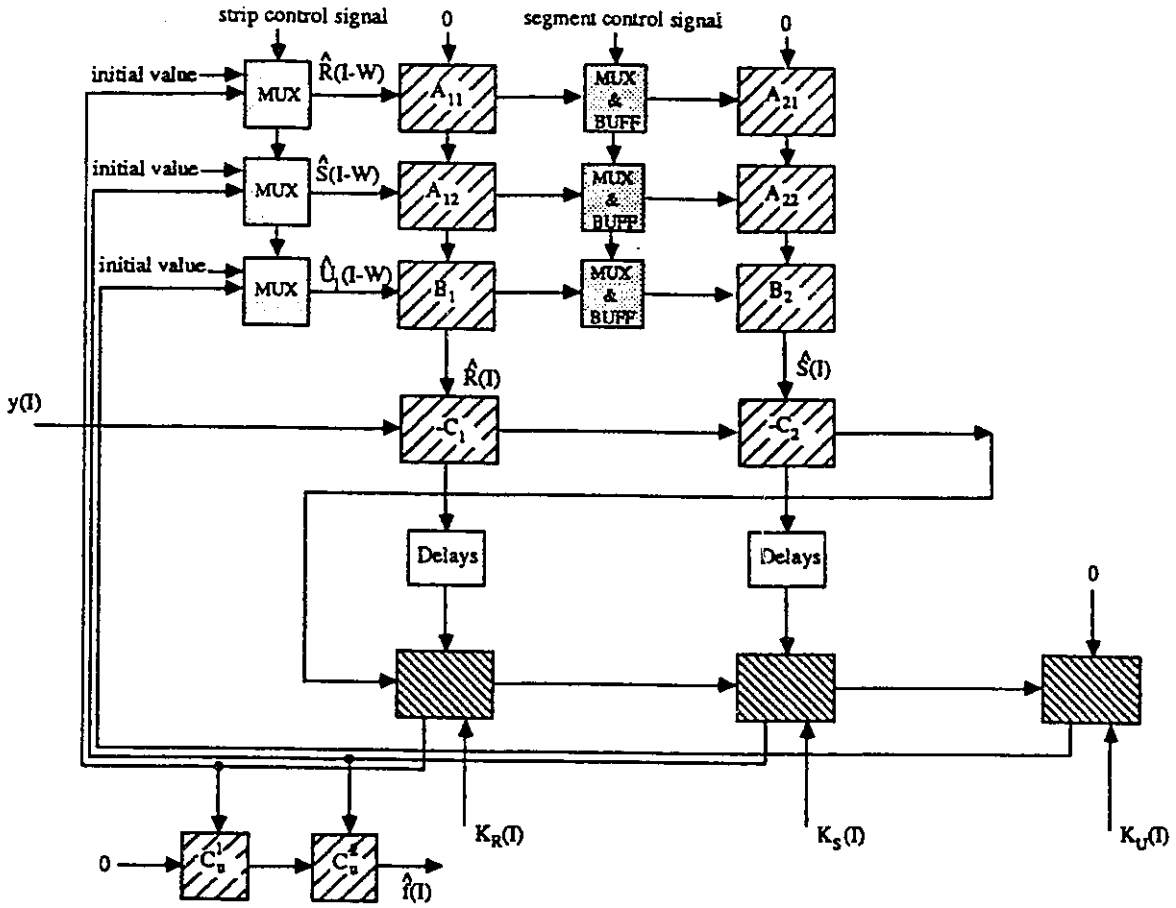



Figure 6.8: The array architecture for state update part of the Kalman filtering

The implementation of the state update is given in Figure 6.8, which performs equations 6.52, 6.56 and 6.57. In this design, we assume that vertical and horizontal state vectors for the boundary are initialized to zero. Two two-dimensional systolic arrays are used to generate the predicted estimate of the state vectors. Next, a one-dimensional array is used to obtain the measured error between the observation value, $y(m,n)$, and the predicted value. Then another 1-D array is followed to correct the estimated states to the optimal values. To begin the operations, the state vector of pixel $(m-1,n)$, $X(I-W) = \left[ \begin{array}{ccc} R(I-W) & S(I-W) & U_1(I-W) \end{array} \right]^t$,

enters the array, the prediction of the state component,$\hat{R}(I)$ can be calculated. At the same time, $\hat{S}(I)$ is generated by the initialized or previous estimated values of the state vectors of pixel in the last strip. Then, $X(I-W+1)$ enters the array for producing $\hat{R}(I+1)$. And, the state vector $X(I-W)$ propagates to the right for calculating $\hat{S}(I+1)$, and so on. When the boundary of the strip is reached, the generated state vectors have to be stored in buffers for processing the next strip. The estimated state vectors of the last strip stored in the buffers are popped to the array for processing the next segment. Notice that in this design at every clock one pixel is estimated and the clock period depends on the time in which the $K(I)$ can be generated. The functions of the basic processing elements for the state update array processor are given in Figure 6.9.

The architecture of the Kalman filter gain calculation part is given in Figure 6.10. It is a direct realization of 6.53, 6.54 and 6.55. We use two-dimensional and one-dimensional systolic arrays to implement the matrix-matrix multiplications, matrix-vector multiplications. Note that the latency for this implementation is not small. But, the inputs for one segment can be pipelined into the array, the outputs come out from the array at every clock time. When the delay of the state update part is equal to the delay of the gain calculation part, one pixel can be estimated at every clock time. It results in real-time processing. The detailed implementation of the Kalman gain calculation processor is given in Figure 6.11. As an illustration, we suppose the dimensions of the matrices are $3 \times 3$.

In the proposed Kalman filter array processors, besides the delays, there are one division processing element and one transposition unit. All the others are systolic arrays. The basic processing elements of systolic arrays perform the multiplication and accumulation operations. After the initialization, the processor utilization efficiency is 100%.

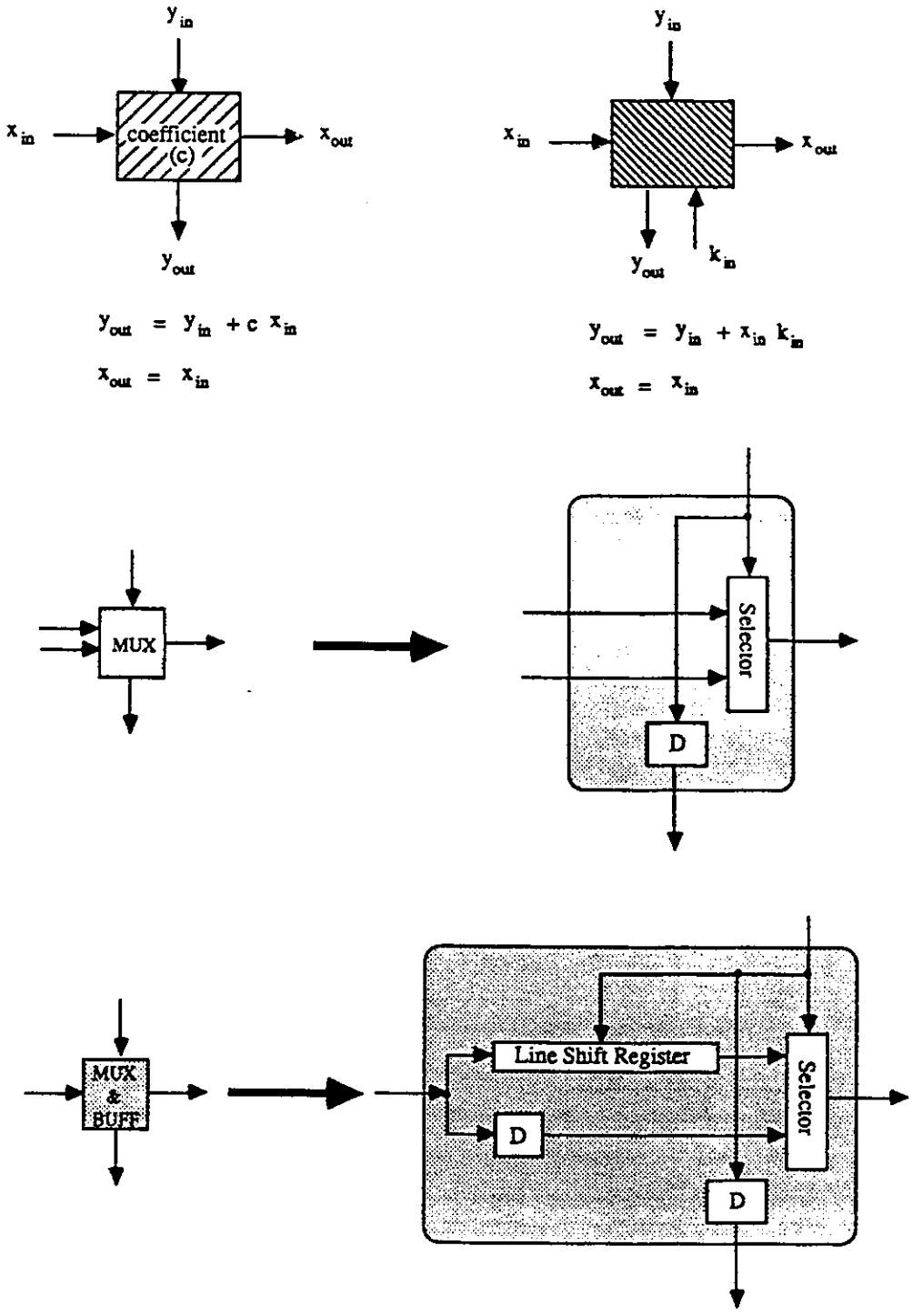



Figure 6.9: The detailed operations of the basic processing elements for the state update array.

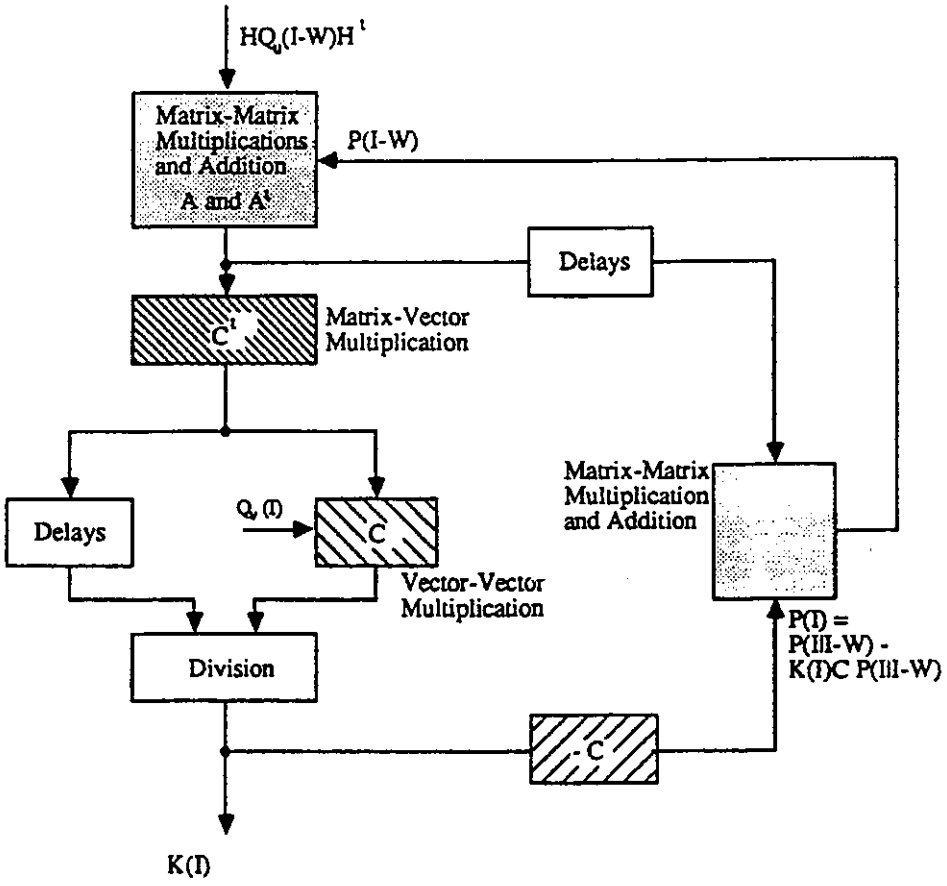



Figure 6.10: The implementation of Kalman gain calculation.

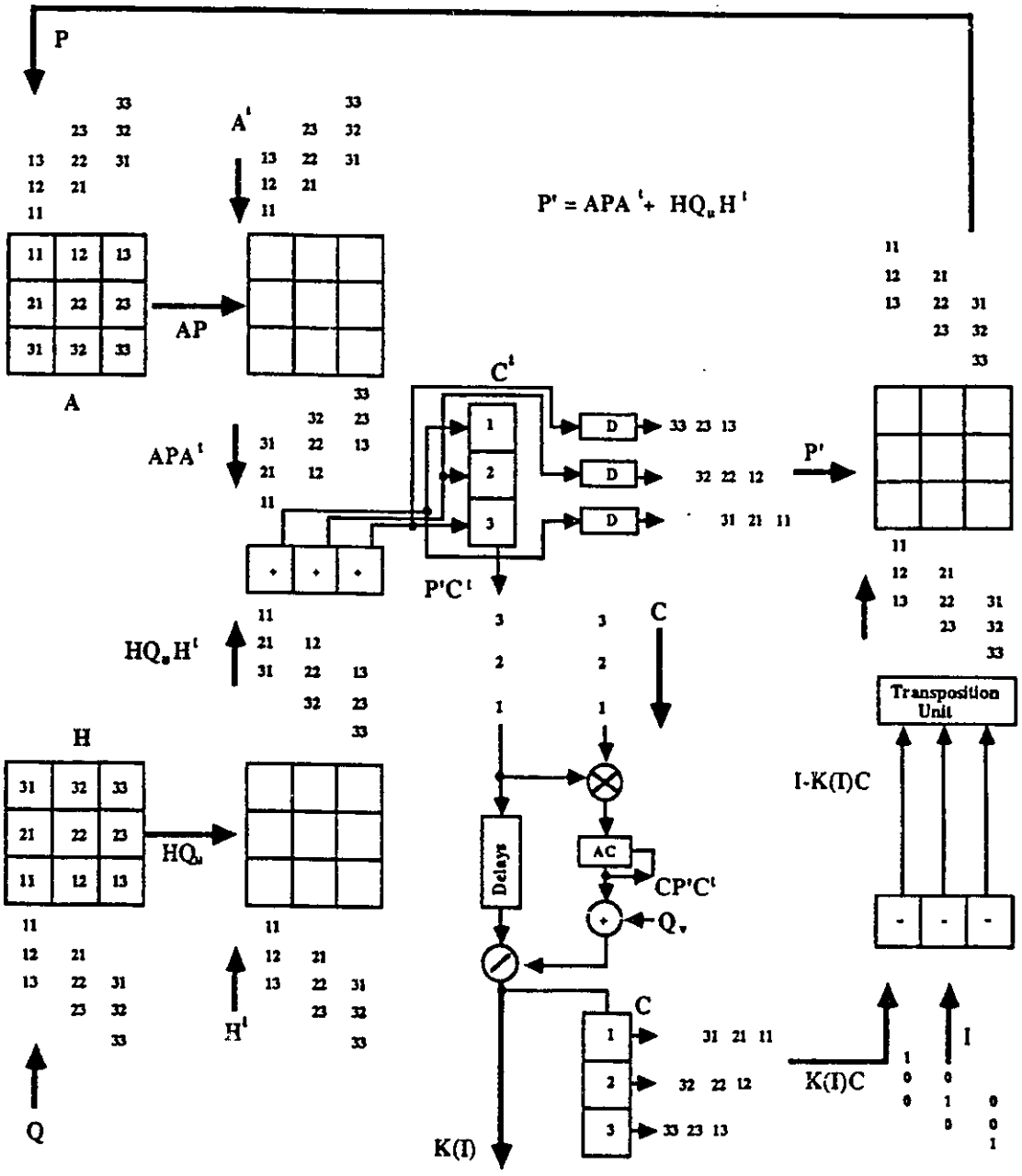



Figure 6.11: The detailed implementation of the Kalman gain calculation processor.

## 6.6 Simulation Results

In order to test the validity of the proposed 2-D state-space Kalman filter scheme and VLSI architecture, we have performed several experiments on real images, a $128 \times 128$ decimated cameraman image and a $256 \times 256$ cameraman image, shown in Figure 6.12 and Figure 6.13, respectively. The image model parameters are obtained by using general least square fit procedure [151] and given in the Table 6.1.

Figure 6.12: The original 128x128 cameraman image, which is magnified by computer with zoom factor = 2.

| | *Coefficients* | | | |
|---|---|---|---|---|
| *Image* | $c_{10}$ | $C_{01}$ | $c_{11}$ | $Q_u$ |
| $128 \times 128$ *Cameraman* | 0.5597539 | 0.7486134 | -0.3117485 | 271.7489 |
| $256 \times 256$ *Cameraman* | 0.7110014 | 0.7446861 | -0.4593296 | 229.4642 |

Table 6.1: The generation model parameters of original images.

The original image was blurred with a certain PSF (point spread function)

Figure 6.13: The original 256x256 cameraman image.

and a zero-mean white random noise as the observation noise was added. In our experiments, the MSE value of an image, $z(m,n)$, is the variance, of the image, which is given by:

$$MSE = \frac{1}{MN}\Sigma_{m,n}(z(m,n) - \overline{z(m,n)})^2 \tag{6.58}$$

where $\overline{z(m,n)}$ is the mean value of the image. The MSD value of an image is the mean squared difference value between the image $z(m,n)$, which can be the blurred image, $y(m,n)$, or the restored image, $\hat{f}(m,n)$, and the original image $f(m,n)$.

$$MSD = \frac{1}{MN}\Sigma_{m,n}(z(m,n) - f(m,n))^2 \tag{6.59}$$

The blurred signal-to-noise ratio (BSNR) is defined as follows:

$$BSNR = 10log_{10}\frac{MSE\ value\ of\ blurred\ image}{variance\ of\ measurement\ noise} \tag{6.60}$$

To compare the performances of different realizations, three methods were used to obtain the restored images for each experiment. These three methods are reduced order models kalman filter (ROMKF) [139], our proposed state-space Kalman filter

with diagonal scanning scheme (SSKFD) and our state-space Kalman filter with line-by-line scanning scheme (SSKFL). The improvements of the Kalman filters, $\eta$, are calculated in decibels according to:

$$\eta_{dB} = 10log_{10}\frac{MSD\ value\ of\ the\ blurred/noised\ image}{MSD\ value\ of\ the\ restored\ image} \tag{6.61}$$

The experimental results are reported as follows:

- Experiment 1.

  In the first experiment, we considered there is no linear blur. Only white noise was added to the original $128 \times 128$ image. For $BSNR = 22db, 16db$ and $10db$, the improvements of ROMKF, SSKFD and SSKFL are very little. The numerical results are given in Table 6.2. The pictorial results are shown in Figure 6.14, Figure 6.15. and Figure 6.16, respectively. It is shown that the performances of three methods are similar. They are not very effective for removing the statistical degradations such as noise and measurement errors.

| *Images* | *Mean* | *MSE* | *MSD* | |
|---|---|---|---|---|
| Noisy image | 118.3246 | 4102.854 | 25.19019 | $BSNR = 22db$ |
| Restored by ROMKF | 117.7950 | 4060.579 | 23.60651 | $\eta = 0.28db$ |
| Restored by SSKFD | 117.7513 | 4062.456 | 23.82764 | $\eta = 0.24db$ |
| Restored by SSKFL | 117.7471 | 4062.419 | 23.80707 | $\eta = 0.25db$ |
| Noised image | 118.4193 | 4145.406 | 97.77112 | $BSNR = 16db$ |
| Restored by ROMKF | 117.8152 | 3995.813 | 76.55389 | $\eta = 1.06db$ |
| Restored by SSKFD | 117.6614 | 3999.540 | 81.64191 | $\eta = 0.78db$ |
| Restored by SSKFL | 117.6514 | 3999.524 | 81.51031 | $\eta = 0.79db$ |
| Noised image | 118.9260 | 4293.528 | 375.6303 | $BSNR = 10.3db$ |
| Restored by ROMKF | 118.0909 | 3775.282 | 210.3007 | $\eta = 2.52db$ |
| Restored by SSKFD | 117.4970 | 3751.064 | 237.4967 | $\eta = 1.99db$ |
| Restored by SSKFL | 117.4859 | 3751.025 | 237.0778 | $\eta = 2.00db$ |

Table 6.2: Comparative results for noisy image without linear blur.

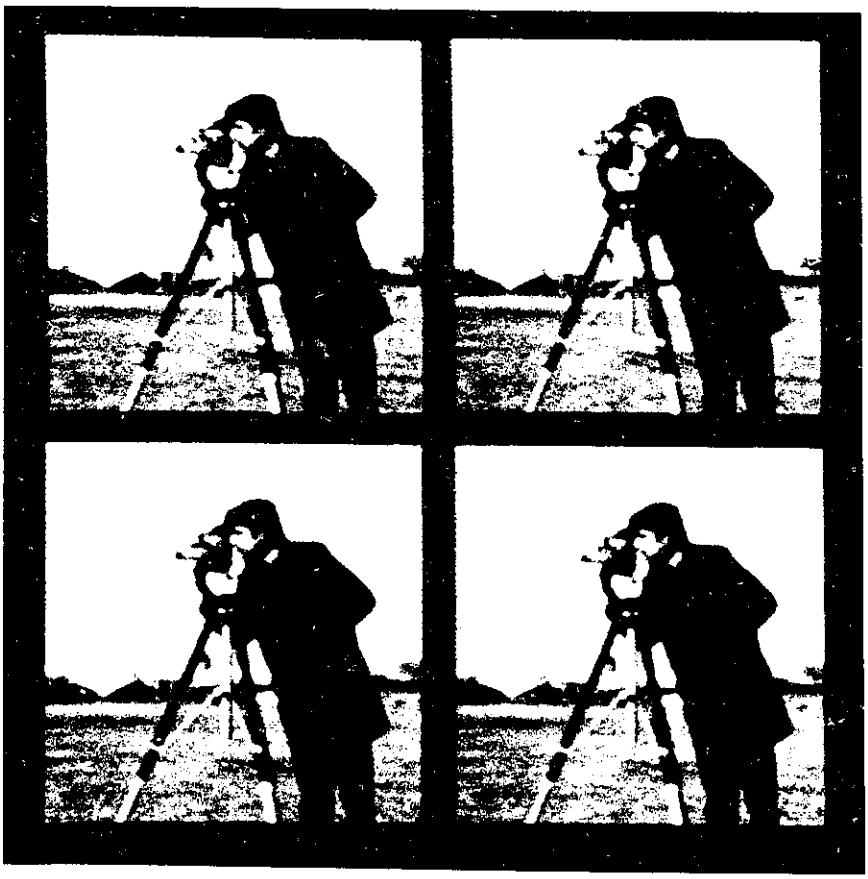


Figure 6.14: The pictorial results for filtering the noisy 128x128 cameraman image, the noise variance is 25.0. The left-up one is the noisy image, the right-up is the restored image by ROMKF, the left-down is the restored image by SSKFD, and the right-down is the restored image by SSKFL.

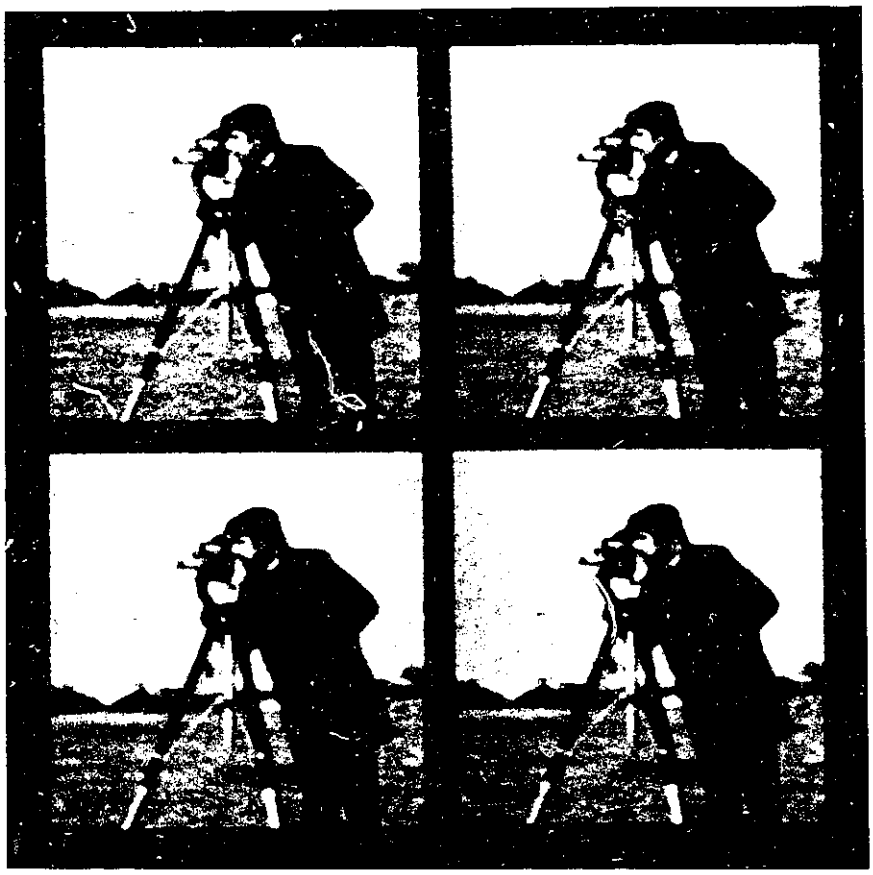


Figure 6.15: The pictorial results for filtering the noisy 128x128 cameraman image, the noise variance is 100.0. The left-up one is the noisy image, the right-up is the restored image by ROMKF, the left-down is the restored image by SSKFD, and the right-down is the restored image by SSKFL.

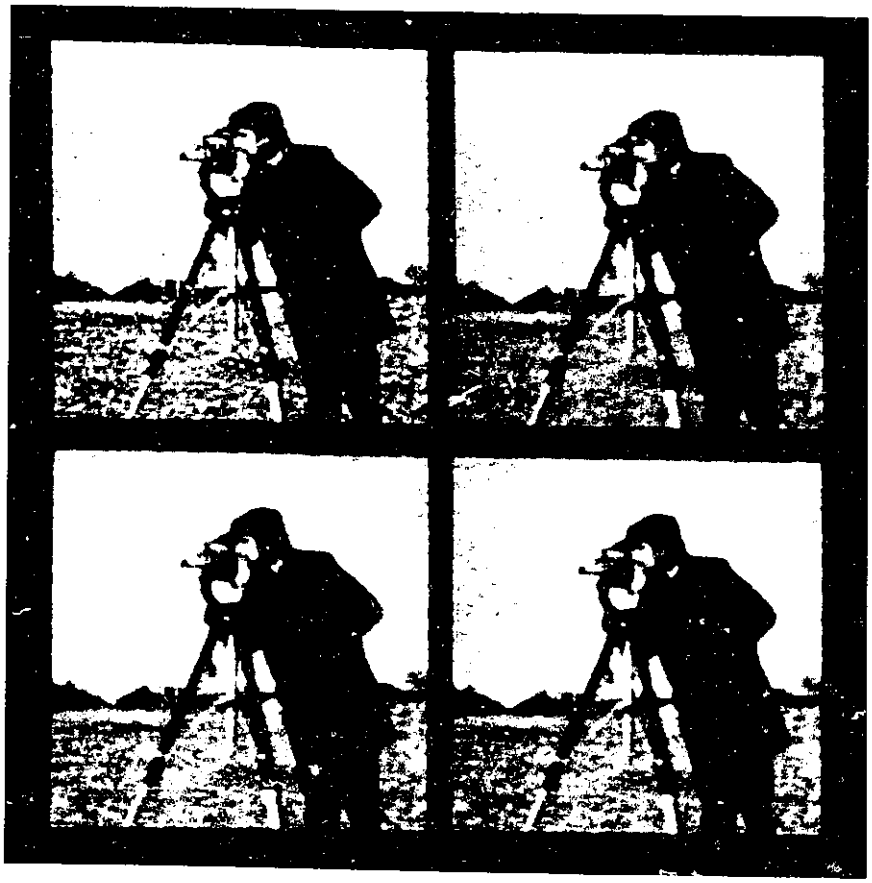


Figure 6.16: The pictorial results for filtering the noisy 128x128 cameraman image, the noise variance is 400.0. The left-up one is the noisy image, the right-up is the restored image by ROMKF, the left-down is the restored image by SSKFD, and the right-down is the restored image by SSKFL.

- Experiment 2.

  In this experiment, we consider the image is only degraded by the linear blur. The linear blur function is chosen as [141]:

$$h_{00} = 0.06. \quad h_{10} = 0.15. \quad h_{01} = 0.15, \quad h_{11} = 0.05.$$

  The numerical performance is given in Table 6.3 and the pictorial results are shown in Figure 6.17.

| *Images* | *Mean* | *MSE* | *MSD* | |
|---|---|---|---|---|
| Blurred by $2 \times 2$ PSF | 48.52795 | 647.7051 | 6325.445 | |
| Restored by ROMKF | 117.5904 | 4006.629 | 223.4454 | $\eta = 14.5db$ |
| Restored by SSKFD | 117.9369 | 3968.936 | 232.3826 | $\eta = 14.4db$ |
| Restored by SSKFL | 117.8571 | 3968.159 | 232.6489 | $\eta = 14.4db$ |

Table 6.3: Comparative results for blurred image without noise.

- Experiment 3.

  In this experiment, we consider the image is degraded by both linear blur and the measurement noise. The linear blur is a $2 \times 2$ PSF, which is the same as the one used in the experiment 2. The white noise has the variance of 6.5, so that the $BSNR = 20db$. The results show that three methods have the same ability to improve the quality of the image. The numerical comparison is given in Table 6.4, and the pictorial results can be observed in Figure 6.18.

- Experiment 4.

  In this experiment, we consider a $3 \times 3$ linear blur function and a white noise. The PSF parameters are adopted from [151]:

  The blurred signal-to-noise ration of the image is kept as $BSNR = 20db$. The numerical results are presented in Table 6.5, and the visual results are given

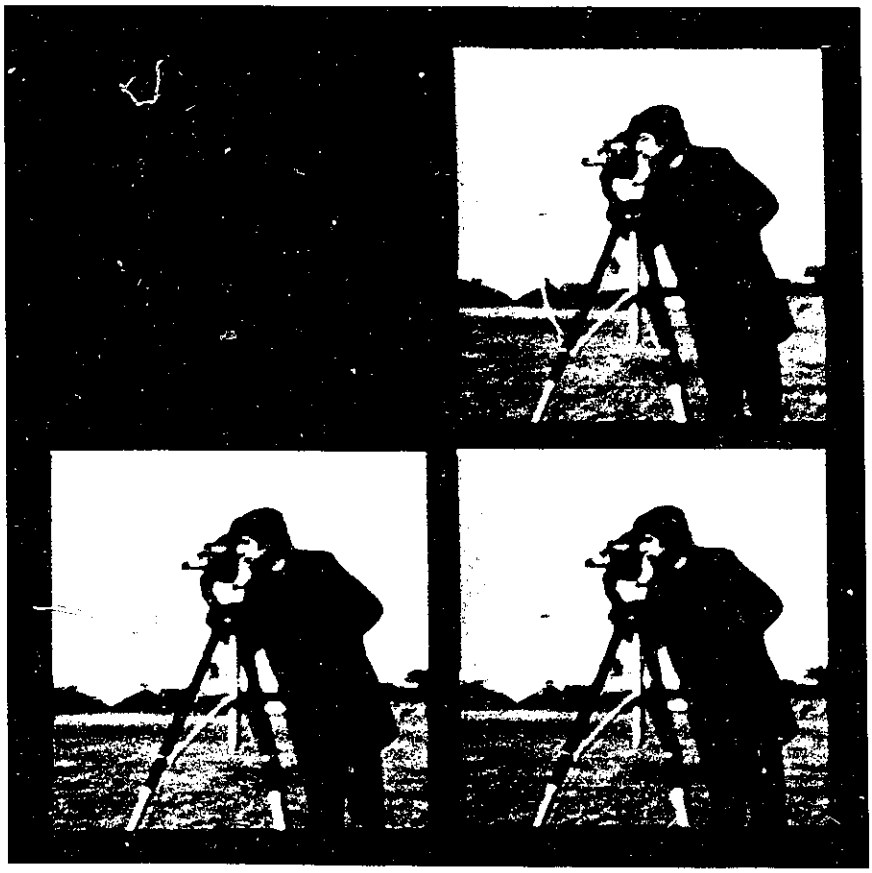


Figure 6.17: The pictorial results for filtering the blurred 128x128 cameraman image. The left-up is the noisy image, the right-up is the restored image by ROMKF, the left-down is the restored image by SSKFD, and the right-down is the restored image by SSKFL.

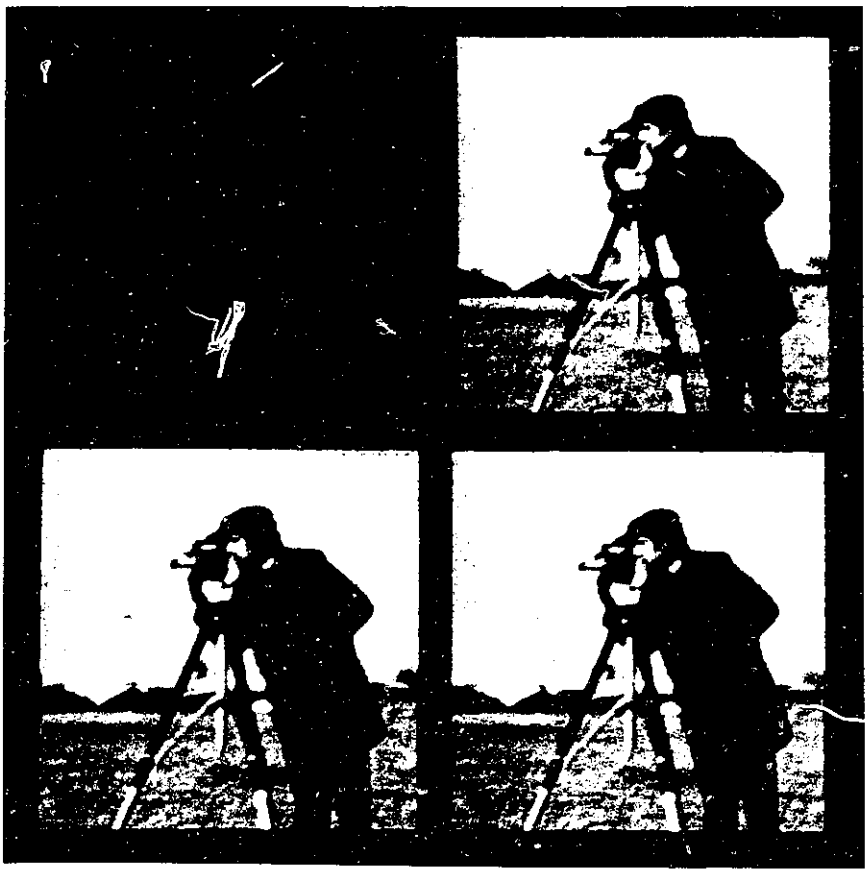


Figure 6.18: The pictorial results for filtering the blurred and noised 128x128 cameraman image. $BSNR = 20db$. The left-up one is the noisy image, the right-up is the restored image by ROMKF, the left-down is the restored image by SSKFD, and the right-down is the restored image by SSKFL.

| *Images* | *Mean* | *MSE* | *MSD* | |
|---|---|---|---|---|
| Degraded by $2 \times 2$ PSF | 48.53534 | 651.9294 | 6334.025 | $BSNR = 20db$ |
| Restored by ROMKF | 117.5443 | 3930.072 | 283.9073 | $\eta = 13.5db$ |
| Restored by SSKFD | 117.7208 | 3929.306 | 302.1602 | $\eta = 13.2db$ |
| Restored by SSKFL | 117.6281 | 3928.159 | 296.2195 | $\eta = 13.3db$ |

Table 6.4: Comparative results for blurred and noisy image.

$$h_{00} = 0.1621 \quad h_{10} = 0.0983 \quad h_{20} = 0.0219$$
$$h_{01} = 0.0983 \quad h_{11} = 0.0596 \quad h_{21} = 0.0133$$
$$h_{02} = 0.0219 \quad h_{12} = 0.0133 \quad h_{22} = 0.0030$$

in Figure 6.19.. These results show comparable performance both visually and with respect to the mean-square error again.

| *Images* | *Mean* | *MSE* | *MSD* | |
|---|---|---|---|---|
| Degraded by $3 \times 3$ PSF | 58.19897 | 923.0527 | 4752.042 | $BSNR = 20db$ |
| Restored by ROMKF | 117.6967 | 3963.260 | 169.2443 | $\eta = 14.5db$ |
| Restored by SSKFD | 117.9247 | 3966.527 | 190.3295 | $\eta = 14.0db$ |
| Restored by SSKFL | 117.9350 | 3966.438 | 190.4330 | $\eta = 14.0db$ |

Table 6.5: Comparative results for blurred and noisy image.

- Experiment 5.

  In order to compare the processing speed, we applied three Kalman filters to the $256 \times 256$ image. The original image is blurred by the same $2 \times 2$ PSF as above experiment and BSNR is equal to $20db$. The results are given in Table 6.6 and Figure 6.20.

- Experiment 6.

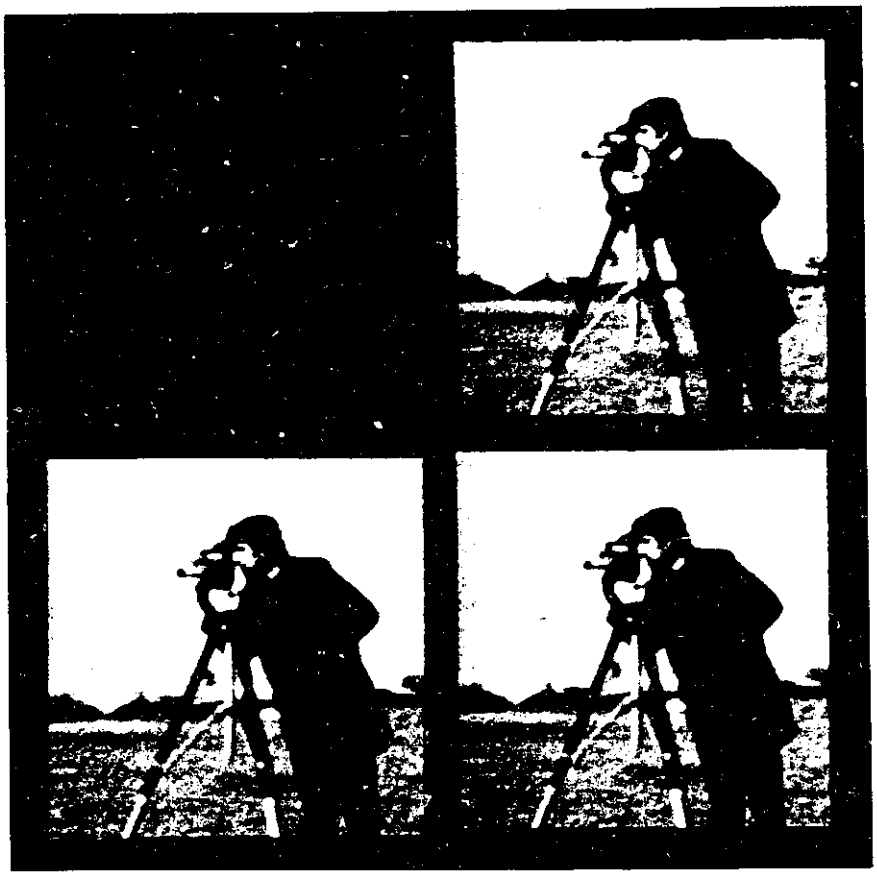


Figure 6.19: The pictorial results for filtering the blurred and noised 128x128 cameraman image. The PSF is $3 \times 3$. The $BSNR = 20db$. The left-up one is the noisy image, the right-up is the restored image by ROMKF, the left-down is the restored image by SSKFD, and the right-down is the restored image by SSKFL.

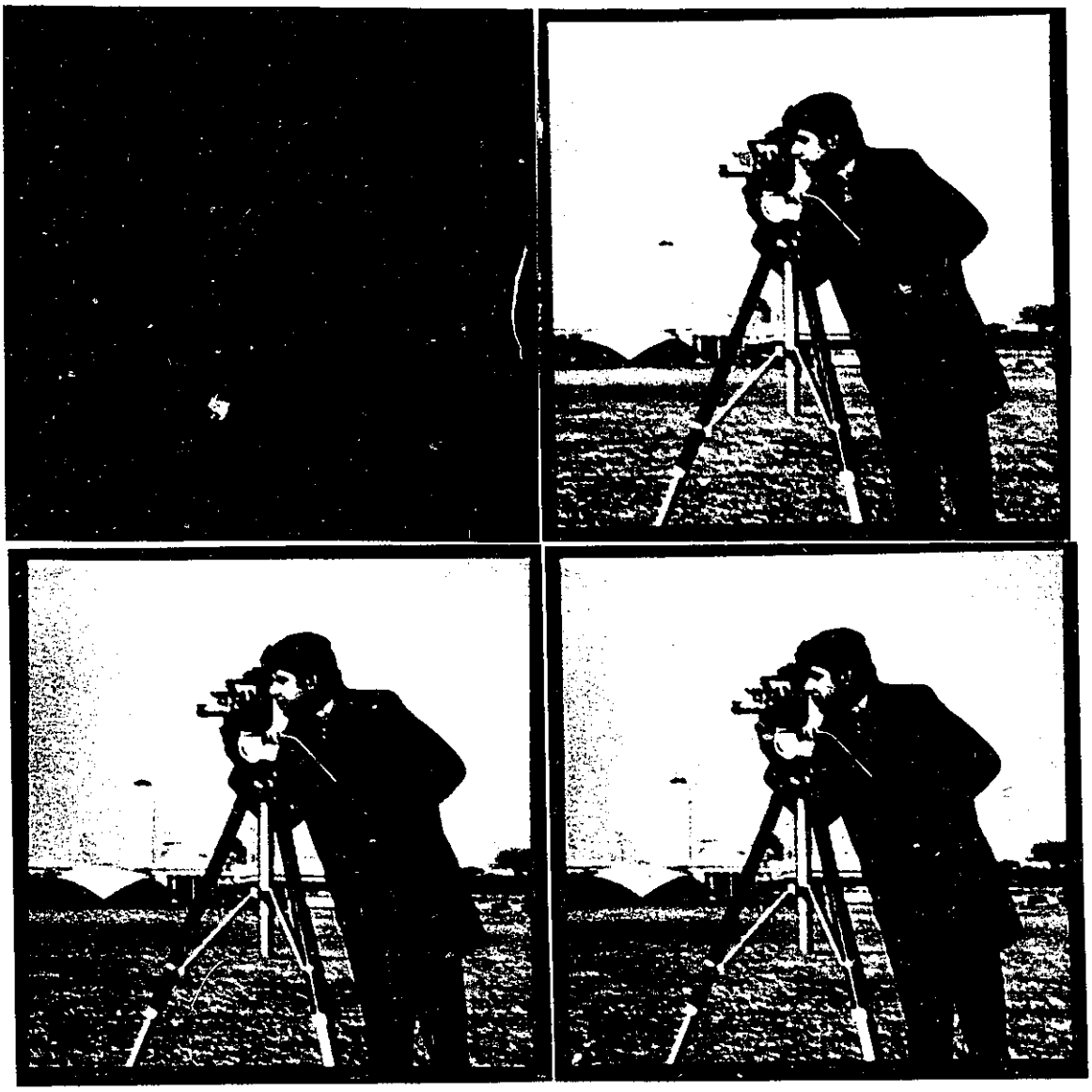


Figure 6.20: The pictorial results for filtering the blurred and noised $256 \times 256$ cameraman image. The PSF is $2 \times 2$. The $BSNR = 20db$. The left-up one is the noisy image, the right-up is the restored image by ROMKF, the left-down is the restored image by SSKFD, and the right-down is the restored image by SSKFL.

| *Images* | *Mean* | *MSE* | *MSD* | |
|---|---|---|---|---|
| Degraded by 2 × 2 PSF | 48.61171 | 229.1342 | 6441.271 | $BSNR = 20db$ |
| Restored by ROMKF | 118.6020 | 1387.524 | 196.7302 | $\eta = 15.2db$ |
| Restored by SSKFD | 117.9062 | 1378.268 | 213.5948 | $\eta = 14.8db$ |
| Restored by SSKFL | 117.9172 | 1378.335 | 214.2590 | $\eta = 14.8db$ |

Table 6.6: Comparative results for blurred and noised 256 × 256 image.

In this experiment, we applied three Kalman filters to the 256 × 256 image again. The original image is blurred by 3 × 3 PSF given in the experiment 4 and BSNR is equal to $20db$. The results are given in Table 6.7 and Figure 6.21.

| *Images* | *Mean* | *MSE* | *MSD* | |
|---|---|---|---|---|
| Degraded by 3 × 3 PSF | 58.28909 | 325.6016 | 4830.787 | $BSNR = 20db$ |
| Restored by ROMKF | 118.0664 | 1381.078 | 93.63222 | $\eta = 17.1db$ |
| Restored by SSKFD | 117.9411 | 1384.042 | 116.3631 | $\eta = 16.2db$ |
| Restored by SSKFL | 118.1005 | 1381.990 | 102.1722 | $\eta = 16.7db$ |

Table 6.7: Comparative results for blurred and noised 256 × 256 image.

These results indicate the following:

- The Kalman filter is not very effective to remove the statistical degradations. However, it is quite effective to improve the image quality which is degraded by blurring or by both blurring and noise.

- In terms of MSE's, the improvement of our state-space Kalman filter with the diagonal scanning is comparable with the line-by-line scanning and also the reduced order model Kalman filter. Visually, the restored images by three methods are the same for each different case.

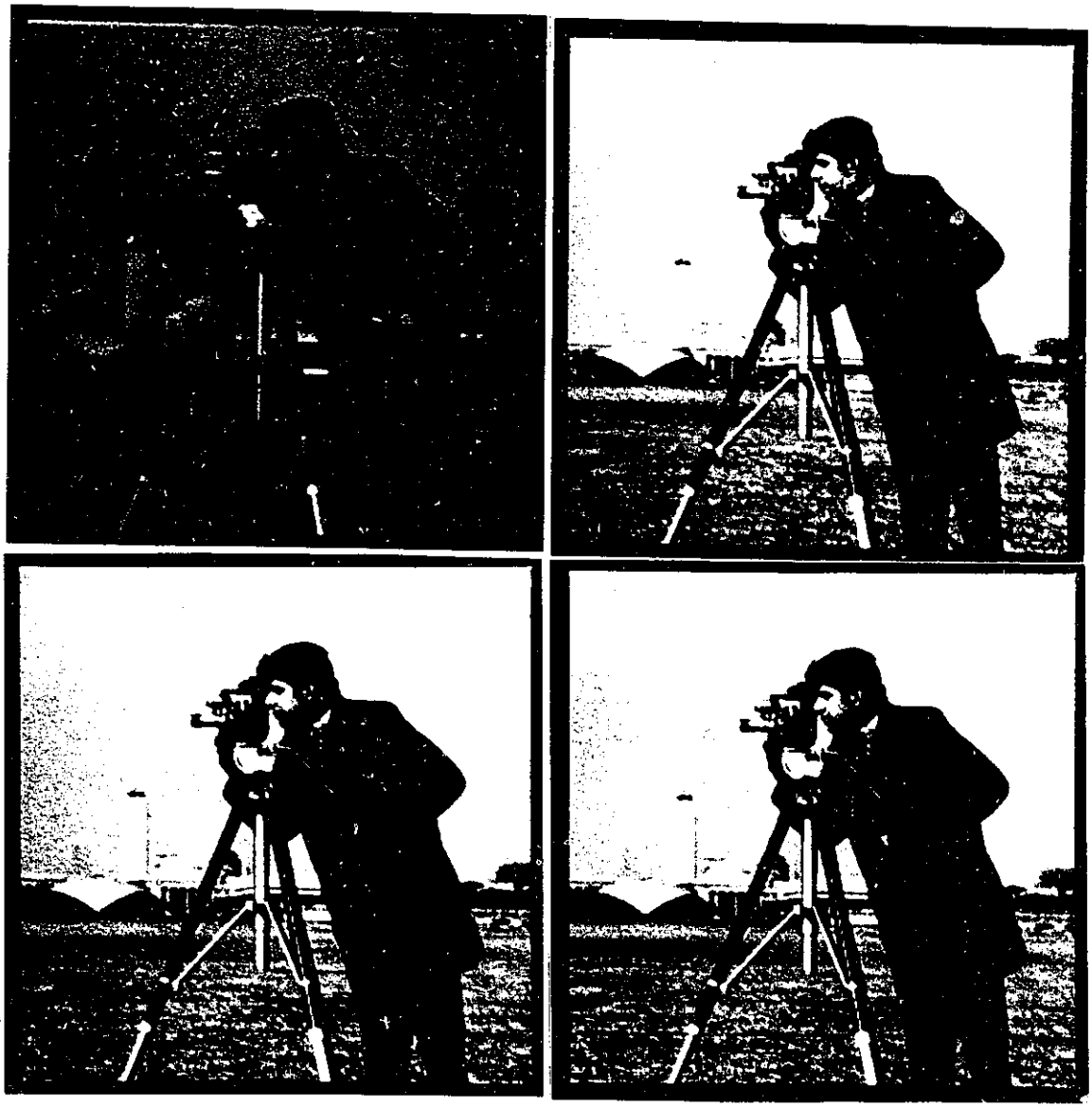


Figure 6.21: The pictorial results for filtering the blurred and noised $256 \times 256$ cameraman image. The PSF is $3 \times 3$. The $BSNR = 20db$. The left-up one is the noisy image, the right-up is the restored image by ROMKF, the left-down is the restored image by SSKFD, and the right-down is the restored image by SSKFL.

- In terms of processing speed, the performances are quite different. By using our proposed architecture, one pixel is estimated at every clock time, which is equal to one multiplication time plus one addition time by SSKFD. If both the image generation model and the blur model are the first order, 24 clock times are required by SSKFL. In the reduced order model Kalman filter algorithm, there are about 540 multiplications for the same order filter. Of course, if an uniprocessor is used, the processing speed is very slow. If we can design a parallel architecture for it, the throughput rate can be increased. However, since it is based on the line-by-line scanning scheme, the speed can be the same order with SSKFL, not the same as SSKFD.

## 6.7 Conclusions

In this chapter, Roesser's 2-D local state space model is used to represent the image process and the blur process. As a result, a simple state-space Kalman filter is derived. This scalar filtering algorithm provides a computationally feasible procedure for the restoration of large images. To implement the Kalman filtering procedure in parallel, a VLSI systolic array structures is presented. For higher speed and higher utilization of this processor, a diagonal scanning method is suggested. The proposed architecture has extensive parallel and pipelining processing capability. The time complexity of processing one strip is $O(MW)$, where $M$ is the length of the image and $W$ is the width of the strip, respectively. If using an ordinary VLSI architecture having similar structure but without using diagonal computational scheme, the time complexity will be $O(MW^2)$. If using an uniprocessor, the time complexity will be $O(MW^4)$. It reduces the time complexity greatly and the utilization of the array processor is 100%. The experimental results show that the proposed algorithm has comparable performance with the ROMKF and SSKFL with less time complexity. It is anticipated that the proposed algorithm and architecture can be extended to

the causal image model and the causal blur with nonsymmetric half-plane support.

# Chapter 7

# Summary and Conclusions

## 7.1 Summary of Results and Contributions

The main purpose of this thesis is to investigate VLSI implementations of recursive state-space filters. After the general mapping methodology for recursive algorithms being discussed, several array architectures for different applications are proposed. The algorithms and architectures developed in this thesis are feasible. Along the way of this implementation investigation, ideas have been developed which can be used in more areas than just state-space filters. In particular, how to design a VLSI algorithm with maximum parallelism, how to improve the utilization efficiency, and how to exploit different level parallelism to speed up the processing will be found useful for design of special-purpose array processors.

In chapter 2, based on the existing results, a modified mapping methodology for mapping recursive algorithms was presented. We avoided the intermediate mapping step so that the DG of an algorithm can be directly mapped onto systolic arrays. To obtain a correct mapping result, the coordinates of the DG were clarified, a new equation for output mapping was given, and a mapping procedure was suggested. Several examples were used to illustrate the capabilities of the suggested approach. In this chapter we next presented the method for optimizing the array design.

The emphasis for this part is on the design of the optimal array for the recursive filtering algorithm. Guidelines for the selection of the schedule vector and the projection vector to obtain an optimal result were given along with the conditions for optimal design (optimality being defined as minimization of computing time or pipeline period). To eliminate the interior inputs and outputs, based on the DG expansion technique, the design method of control signals and processing elements was illustrated by an example. To reduce the number of input/output lines, a data distribution scheme was suggested in the case of state-space filtering. As well, the reduction of the array size was discussed and two methods of partitioning algorithms were shown. Of particular interest was the improvement of the utilization of the arrays. Besides the pipeline interleaving scheme, a multiprojection method to be used for 2-D recursive state-space filtering is emphasized.

It has been found that, for 1D and 2D signals, the state space representation offers a very suitable basis for realization using systolic arrays. The state space matrix representation also allows for the application of state transformation matrices to achieve minimum roundoff noise, low coefficient sensitivity and freedom from overflow oscillations.

Based on the presented approach in chapter 2, VLSI architectures for 1-D and 2-D recursive state-space filtering were proposed in chapter 3 and 4. For 1-D IIR filters, the state-space representation was first used to obtain a simple systolic array, where data broadcast was needed. To speed up the processing, an advanced state update algorithm was developed and a corresponding array was given. By decomposing the algorithm, a cascade form of second-order state-space structures was suggested, which resulted in high speed and high efficiency with less hardware. In the 2-D case, the idea of different level speedup was developed. At the pixel level, also called local speedup, the direct form array and the advanced state update array were used. At the image level, called global speedup, multiprojection method was first used to improve the efficiency of the processors. Then, the spatial concurrency

in 2-D systems was used to find a global speedup architecture. This architecture consists of a number of column array processors, and it works on multiple columns of images in parallel. The throughput rate of this design is adjustable and therefore can be very high.

In chapter 5, a new adaptive algorithm for state-space recursive systems was given. The adaptive state-space filtering algorithm is based on the LMS method. The gradients are derived directly from state equations. In order to reduce the complexity, the parallel form of lower order adaptive sections was suggested. The stability monitoring of the adaptive filter was studied. The sufficient conditions for the stable adaptation were given, and comparisons between state-space filters and other realizations were briefly discussed. The convergence behavior and round-off noise performance were also investigated. The results of computer simulations were quite encouraging. To speed up the filtering and adaptation, a VLSI array architecture was suggested.

Finally, chapter 6 proposed a very fast Kalman filter for image restoration. In particular, a new state-space model Kalman filter algorithm was developed. By using the local state-space structure to represent the image generation model and the degradation model, a simple composite dynamic system representation was obtained. To establish the Kalman filter equations, an appropriate state vector was chosen. For real-time processing, the diagonal scanning scheme was used, and a dedicated VLSI array architecture was proposed. In order to verify the proposed algorithm and implementation, experimental results were given, which show that the state-space Kalman filter can result in real-time image restoration with comparable image restoration quality as other Kalman filter algorithms.

## 7.2 Suggestions For Future Work

VLSI technology is a very fast developing area and many researchers are paying much attention to VLSI from different aspects. The special-purpose array processors have created a new architecture horizon in implementing parallel algorithms directly on hardware. The research work done for this thesis led to a number of unsolved problems and research directions. Some of them are:

The mapping methodology described in chapter 2 uses a projection vector and a schedule vector to map an algorithm onto array architectures. It was demonstrated by examples. (1). The question of whether this mapping method can map any recursive algorithm onto array processors has not been answered theoretically. (2). The architectures for state-space filtering proposed in this thesis were obtained using this method along with some special considerations. Hence, what is the role of the general mapping methodology in the design world of special-purpose array architectures? (3). To eliminate the interior inputs and outputs in mapping the DG of matrix-vector multiplication and addition, the elements of the matrix were assumed to be preloaded into the array processors. If the original DG has interior I/O, how can it be mapped onto an array without interior I/O? (4). If the original DG is spatial-invariant or uniform, after expanding the DG along border lines, the DG is not uniform any more. Therefore, it is important to derive a new uniform recurrence dependency as well as the control signal for processor elements to switch their functions.

For the array architectures obtained in chapter 3 and 4, further considerations are required. A further step is necessary to bring the arrays closer to VLSI realization. A review of actual realization constraints should be made, considering both the arithmetic constraints as well as the practical VLSI layout constraints, between which there is a great deal of interaction. The arithmetic elements of the array will be generally of the form of: multipliers and adder/accumulator as well

as delay latches. For some cases also dividers, comparators, etc. are to be considered. To some considerable extent, these elements may be realized in memory form, which leads to the application of table lookup, or stored arithmetic techniques. The realization of arithmetic elements there becomes a question of what technique of element realization is the best, in view of the circuit area requirements (the possible numbers of arithmetic elements per VLSI chip); the interconnections, as well as the inter-chip connections. Other considerations are in the overall circuit reliability and the possibility of replacing faulty elements by substitution with spare elements.

In chapter 5, it was demonstrated that adaptive state-space filters could successfully converge to the desired filter. Adaptive state space filters were compared to direct form adaptive IIR filters and adaptive lattice filters in terms of stability monitoring, convergence rate and roundoff noise using several examples. It would be very useful if a theoretical analysis could be given. Also, the question of ensuring the algorithm converges to the global minimum still needs to be investigated. Overall, this algorithm requires considerably more work to be done. For example, the stability control scheme, the choice of the convergence parameters, the reduction of computation complexity, details of VLSI implementation, and applications.

With regard to the Kalman filter developed in chapter 6, more experimental simulations have to be done. It was found that for some degraded images, Kalman filter algorithms, the one proposed in this thesis and ones proposed by other authors, can not give satisfactory results. Furthermore, in this thesis, the parameters for image generation model and the blur mode were known. In the practical case, the identification of the parameters and the restoration of the images have to be performed at the same time. A fast identification algorithm and architecture needs to be developed. Finally, it is desirable to develop a fast adaptive Kalman filter for image restoration.

# Bibliography


[1] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing.* Prentice-Hall, Inc., 1975.

[2] S. Y. Kung. *VLSI Array Processors.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1988.

[3] K. S. Lin, G. A. Frantz, and Jr. R. Simar. The tms320 family of digital signal processors. *Proceedings of IEEE*, 75(9):1143–1159, September 1987.

[4] T. S. Huang (Ed.). *Two-Dimensional Digital Signal Processing I.* Springer-Verlag, 1981.

[5] A. N. Venetsanopoulos. Real-time image processing. In S. G. Tzafestas, editor, *Multidimensional Systems*, pages 345–399, Marcel Dekker, 1986.

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

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

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

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

[10] S. Y. Kung. On supercomputing with systolic/wavefront array processors. In *Proceedings of the IEEE*, pages 867–884. IEEE, July 1984.

[11] H. T. Kung. Lets design algorithms for VLSI systems. In *Proceedings of the Caltech Conference on VLSI*, January 1979.

[12] H. T. Kung. Why systolic architectures? *IEEE Transaction on Computer*, 37–46, January 1982.

[13] H. T. Kung and C. E. Leiserson. *Systolic Arrays (for VLSI)*, chapter 8. Addison-Wesley, 1980.

[14] H. T. Kung. *Systolic Algorithms*. Academic Press, 1984.

[15] D. W. L. Teh and A. V. Kulkarni. Systolic processing and an implementation for signal and image processing. *IEEE Trans. on Computers*, C-31(10), October 1982.

[16] H. T. Kung. Special purpose devices for signal and image processing: an opportunity in very large scale integration (VLSI). In *Proceedings of SPIE, Real-Time Signal Processing*, SPIE, 1980.

[17] H. T. Kung and S. W. Song. A systolic array chip for the convolution operator in image processing. *MPC 79 Proc., Xerox, PAR/SSL*, 1979.

[18] H. T. Kung and S. W. Song. A systolic 2-d convolution chip. *TR CMU-CS-81-110, Canegie-Mellon University, CS Dept.*, March 1981.

[19] H. T. Kung, L. W. Ruance, and D. W. L. Yeh. A two-level pipelined systolic array for multidimensional convolutions. *Image and Vision Computing*, Feb. 1983.

[20] H. T. Kung and R. L. Picard. *One-Dimensional Systolic Arrays for Multi-Dimensional Convolution and Resampling*. Springer-Verlag, 1984.

[21] G. Bongiovanni. Two VLSI structures for the discrete fourier transform. *IEEE Trans. on Computers*, C-32(8), August 1983.

[22] C. D. Thompson. Fourier transforms in VLSI. *IEEE Trans. on Computers*, C-32(11), November 1983.

[23] E. H. Wold and A. M. Despain. Pipeline and parallel-pipeline fft processors for VLSI implementation. *IEEE Trans. on Computers*, C-33(5), May 1984.

[24] H. T. Kung. Use of VLSI in algebraic computation: some suggestions. In *Proceeding of the 1981 ACM Symposium on Symbolic and Algebraic Computations*, 1981.

[25] K. Hwang and Y H. Cheng. Partitioned matrix algorithms for VLSI arithmetic systems. *IEEE Trans. on Computers*, C-31(12), December 1982.

[26] D. I. Moldovan, C. I. Wu, and J. A. B. Fortes. Mapping an arbitrarily large QR algorithm into a fixed size VLSI array. In *Proc. of 1984 International Conf. on Parallel Processing*, August 1984.

[27] I. V. Ramakrishnan and P. J. Varman. Modular matrix multip'ication on a linear array. *IEEE Trans. on Computers*, C-33(11), November 1984.

[28] C. S. Yeh, I. S. Reed, and T. K. Truong. Systolic multipliers for finite fields GF($2^m$). *IEEE Trans. on Computers*, C-33(4), April 1984.

[29] B. B. Zhou and R. P. Brent. A high throughput systolic implementation of the second order recursive filter. In *Proc. of ICASSP'88*, pages 2053–2056, IEEE, 1988.

[30] J. A. B. Fortes and B. W. Wah. Systolic arrays: a survey of seven projects. *IEEE Computer*, July 1987.

[31] P. Quinton. The systematic design of systolic arrays. In Y. Robert F. F. Soulie and M. Tchuente, editors. *Automata Networks*, chapter 9, 1988.

[32] L. Johnson and et al. *A Mathematical Approach to Modeling the Flow of Data and Control in Computational Networks*. Computer Science Press, 1981.

[33] D.I. Moldovan and A. Varma. Design of algorithmically specialized VLSI devices. In *Proc. of 1983 International Conf. on Computer Design*, 1983.

[34] D. Cohen. Mathematical approach to iterative computation networks. In *Proceedings of 4th Symp. on Computer Arithmetic*, pages 226–238, 1978.

[35] U. Weiser and A. Davis. *A wavefront notation tool for VLSI design.* Computer Science Press, 1981.

[36] C. E. Leiserson and J. B. Saxe. Optimizing synchronous circuits. In *Proceedings of 22nd IEEE Symp. on Foundations of Computer Science*, pages 23–36, 1981.

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

[38] W. L. Miranker and A. Winkler. Spacetime representations of computational structures. *Computing*, 32, May 1984.

[39] H. V. Jagadish, S. K. Rao, and T. Kailath. Array architectures for iterative algorithm. *Proc. of IEEE*, 75(9):1304–1321, September 1987.

[40] S. K. Rao and T. Kailath. Regular iterative algorithms and their implementation on processor arrays. *Proc. of IEEE*, 76(3):259–269, March 1988.

[41] J. A. B. Fortes, K. S. Fu, and B. W. Wah. Systematic approaches to the design of algorithmically specified systolic arrays. In *Proc. of International Conf. on Acoustics, Speech, and Signal Processing*, IEEE, 1985.

[42] J. A. B. Fortes. *Algorithm transformmations for parallel processing and VLSI architecture design.* PhD thesis. University of Southern California, Los Angeles, Ca, 1983.

[43] C. V. Ramamoorthy and Y. W. Ma. *Large scale computer systems.* Van Nostrand Reinhold Company, 1983.

[44] W. Steenaart, J. Y. Zhang, and Y. Tunca. Mapping recursive algorithms onto array architectures. In E. De Prettere, editor, *Algorithms and Parallel VLSI Architectures*, 1991.

[45] W. Steenaart and J. Y. Zhang. Mapping recursive algorithms onto systolic arrays. In *Proceedings of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, May 1991.

[46] R. W. Priester, H. J. Whitehouse, K. Bromley, and J. B. Clay. Problem adaptation to systolic arrays. In *Proceedings of SPIE Vol. 298 Real-Time Signal Processing IV*, pages 33–39, 1981.

[47] C. T. Mullis and R. A. Roberts. Synthesis of minimum roundoff noise fixed point digital filters. *IEEE Transaction on Circuits and Systems*, CAS-23(9):551–562, September 1976.

[48] C. T. Mullis and R. A. Roberts. Filter structures which minimize roundoff noise in fixed point digital filters. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 505–508, IEEE, April 1976.

[49] C. T. Mullis and R. A. Roberts. Roundoff noise in digital filters: frequency transformation and invariants. *IEEE Transaction on Acoustics, Speech and Signal Processing*, ASSP-24:538–550, December 1976.

[50] S. Y. Hwang. Dynamic range constraint in state-space digital filtering. *IEEE Transaction on Acoustics, Speech, and Signal Processing*, ASSP-23:591–593, June 1975.

[51] S. Y. Hwang. Roundoff noise in state-space digital filtering: a general analysis. *IEEE Transaction on Acoustics, Speech, and Signal Processing*, ASSP-24(1):256–262, June 1976.

[52] S. Y. Hwang. Roundoff noise minimization in state-space digital filtering. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 498–500, IEEE, April 1976.

[53] L. B. Jackson, A. G. Lindgren, and Y. Kim. Synthesis of state-space digital filters with low roundoff noise and coefficient sensitivity. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 41–44, IEEE, April 1977.

[54] L. B. Jackson, A. G. Lindgren, and Y. Kim. Optimal synthesis of second-order state-space structures for digital filters. *IEEE Transaction on Circuits and Systems*, CAS-26(3):149–153, March 1979.

[55] W. Martin Snelgrove and A. S. Sedra. Synthesis and analysis of state-space active filters using intermediate transfer functions. *IEEE Transaction on Circuits and Systems*, CAS-33(3):287–301, March 1986.

[56] B. W. Bomar. Computationally efficient low roundoff noise second-order state-space structures. *IEEE Transaction on Circuits and Systems*, CAS-33(1):35–41, January 1986.

[57] C. W. Barnes and A. T. Fam. Minimum norm recursive digital filters that are free of overflow limit cycles. *IEEE Transaction on Circuits and Systems*, CAS-24(10):569–574, October 1977.

[58] C. W. Barnes. Roundoff noise and overflow in normal digital filters. *IEEE Transaction on Circuits and Systems*, CAS-26(3):154–159, March 1979.

[59] L. B. Jackson. Limit cycles in state-space structures for digital filters. *IEEE Transaction on Circuits and Systems*, CAS-26(1):67–68, January 1979.

[60] L. B. Jackson. *Digital Filters and Signal Processing*, chapter 11, pages 181–234. Kluwer Academic, 1986.

[61] R. A. Roberts and C. T. Mullis. *Digital Signal Processing*, chapter 8, pages 287–323. Addison-Wesley, 1987.

[62] R. A. Roberts and C. T. Mullis. *Digital Signal Processing*, chapter 9, pages 343–416. Addison-Wesley, 1987.

[63] S. Y. Kung. VLSI signal processing: from transversal filtering to concurrent array processing. In S. Y. Kung, H. J. Whitehouse, and T. Kailath, editors, *VLSI and Modern Signal Processing*, pages 127–152, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

[64] B. B. Zhou and R. P. Brent. A high throughput systolic implementation of the second order recursive filter. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2053–2056, IEEE, 1988.

[65] S. K. Rao and T. Kailath. VLSI arrays for digital signal processing: part i– a model identification approach to digital filter realizations. *IEEE Transaction on Circuits and Systems*, CAS-32(11):1105–1117, November 1985.

[66] D. Dubois and W. Steenaart. High speed stored product recursive digital filters. *IEEE Transaction on Circuits and Systems*, CAS-29(6):390–393, June 1982.

[67] S. Sridharan. Implementation of state-space digital filter structures using block floating point arithmetic. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 908–911, IEEE, 1987.

[68] H. H. Lu, E. A. Lee, and D. G. Messerschmitt. Fast recursive filtering with multiple slow processing elements. *IEEE Transaction on Circuits and Systems*, CAS-32(11):1119–1129. November 1985.

[69] T. Aboulnasr and W. Steenaart. Real-time array processor for 2-d spatial filtering. In *Proceedings of EUSIPCO'86*, pages 687–690, 1986.

[70] R. B. Urquhart and D. Wood. Systolic matrix and vector multiplications methods for digital signal processing. In *Proceedings of the IEEE*, IEEE, October 1984.

[71] J. Y. Zhang, W. Steenaart, and T. Aboulnasr. Systolic array implementation for IIR filter. In G. Glasford and K. Jabbour, editors, *Proceedings of 30th Midwest Symposium on Circuits Systems*, pages 1336–1339, August 1987.

[72] J. Y. Zhang and W. Steenaart. VLSI realizable high performance structures for real-time state-space filtering. *IEEE Transaction on Circuits and Systems*, CAS-36(4), April 1989.

[73] J. Y. Zhang and W. Steenaart. VLSI high performance structures for real-time state-space recursive filters. In *Proceedings of Int. Conf. on Circuits and Systems in China*, pages 886–889, July 1989.

[74] T. Aboulnasr and W. Steenaart. Real-time array processor for 2-d spatial filtering. *IEEE Transaction on Circuits and Systems*, CAS-35(4):451–455, April 1988.

[75] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* Prentice-Hall, Inc., 1989.

[76] M. Kawamata and T. Higuchi. Synthesis of 2-d separable denominator digital filters with minimum roundoff noise and no overflow oscillations. *IEEE Trans. on Circuits and Systems*, CAS-33(4):365–372, April 1986.

[77] M. Kawamata and T. Higuchi. Synthesis of 2-d separable denominator digital filters with minimum roundoff noise and no overflow oscillations. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 1087–1091, IEEE, Kyoto, Japan, June 1985.

[78] T. Lin, M. Kawamata, and T. Higuchi. A unified study on the roundoff noise in 2-d state space digital filters. *IEEE Trans. on Circuits and Systems*, CAS-33(7):724–730, July 1986.

[79] W. S. Lu and A. Antoniou. Synthesis of 2-d state-space fixed-point digital-filter structures with minimum roundoff noise. *IEEE Transaction on Circuits and Systems*, CAS-33(10):965–973, October 1986.

[80] T. Lin, M. Kawamata, and T. Higuchi. Design of 2-d digital filters with an arbitrary response and no overflow oscillations based on a new stability condition. *IEEE Trans. on Circuits and Systems*, CAS-34(2):113–126, Feb. 1987.

[81] N. G. El-Agizi and M. M. Fahmy. Two-dimensional digital filters with no overflow oscillations. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, ASSP-27(5):465–469, October 1979.

[82] A. N. Venetsanopoulos, K. M. Ty, and A. C. P. Loui. High-speed architectures for digital image processing. *IEEE Transaction on Circuits and Systems*, CAS-34(8):887–896, August 1987.

[83] K. M. Ty and A. N. Venetsanopoulos. A fast filter for real-time image processing. *IEEE Transaction on Circuits and Systems*, CAS-33(10):948–957, October 1986.

[84] R. A. Cohen, M. S. Woods, and J. F. McDonald. A video rate architecture for a recursive two-dimensional filter. In *Proceedings of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pages 1973–1976, 1987.

[85] K. K. Parhi and D. G. Messerschmitt. Concurrent architectures for two-dimensional recursive digital filtering. *IEEE Trans. on Circuits and Systems*, CAS-36(6):813–829, June 1989.

[86] K. K. Parhi and D. G. Messerschmitt. Two-dimensional recursive digital filtering: pipelining: one- and two-dimensional block processing. In *Proceedings of IEEE Int. Symp. on Circuits and Systems*, pages 1521–1524, May 1988.

[87] S. Attasi. Systèmes lineaires homogènes á duex indices. *Rapport Laboria*, September 1973.

[88] S. Attasi. Modèlisation et traitement des suites á duex indices. *Rapport Laboria*, September 1975.

[89] E. Fornasini and G. Marchesini. Algebraic realization theory of two dimensional filters. In A. Ruberti and R. Mohlers, editors, *Variable Structure Systems*, Springer Verlag, 1975.

[90] E. Fornasini and G. Marchesini. State-space realization theory of two-dimensional filters. *IEEE Transaction on Automation and Control*, AC-21:484–492, August 1976.

[91] D. D. Givone and R. P. Roesser. Minimization of multidimensional linear iterative circuits. *IEEE Transaction on Computer*, C-22:673–678, July 1973.

[92] R. P. Roesser. A discrete state-space model for linear image processing. *IEEE Transaction on Automatic Control*, AC-20:1–10, February 1975.

[93] S. K. Mitra, A. D. Sagar, and N. A. Pendergrass. Realizations of two-dimensional recursive digital filters. *IEEE Trans. on Circuits and Systems*, CAS-22:177–184, March 1975.

[94] G. A. Lampropoulos and M. M. Fahmy. A new realization for 2-d digital filters. *IEEE Transaction on Acoustics, Speech, and Signal Processing*, ASSP-35(4):533–542, April 1987.

[95] M. Bisiacco. State and output feedback stabilizability of 2-d systems. *IEEE*, 1246–1248, 1985.

[96] J. E. Kurek. The general state-space model for a two-dimensional linear digital system. *IEEE Trans. on Automatic control*, AC-30(6):600–602, June 1985.

[97] S. Y. Kung and el al. New results in 2-d systems theory, part II: 2-d state-space models – realization and the notions of controllability, observability, and minimality. In *Proceedings of the IEEE*, pages 945–961, 1977.

[98] T. Aboulnasr. *New Design and Implementation Techniques for Two-Dimensional Digital Filters*. PhD thesis, Queen's University, Kingston, Ontario, Canada, 1984.

[99] J. Y. Zhang and W. Steenaart. VLSI architectures for high speed two-dimensional state-space filtering. In *Proceedings of 31th Midwest Symposium on Circuits Systems*, August 1988.

[100] J. Y. Zhang and W. Steenaart. VLSI architectures for high speed two-dimensional state-space recursive filtering. In T. R. Hsing, editor, *Proceedings of Visual Communications and Image Processing'88*, pages 354–361, SPIE, November 1988.

[101] J. Y. Zhang and W. Steenaart. VLSI implementations of high speed two-dimensional state-space recursive filtering. In *Proceedings of the Int. Symp. on Circuits and Systems*, pages 1099–1102, May 1989.

[102] J. Y. Zhang and W. Steenaart. VLSI architectures for high speed 2-d state-space filtering. In *Symposium Digest of Eighth Kobe Int. Symp. on Electronics and Information Sciences*, pages 8.1–8.10, July 1989.

[103] J. Y. Zhang and W. Steenaart. High speed architectures for two-dimensional state-space filtering. *IEEE Trans. on Circuits and Systems*, CAS-37(6):831–836, June 1990.

[104] L. H. Sibul, editor. *Adaptive Signal Processing.* IEEE Press, 1987.

[105] B. Widrow and S. D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985.

[106] B. Widrow and et al. Adaptive noise cancelling: Principles and applications. *Proceedings of IEEE*, 63(12):1692–1716, December 1975.

[107] C. Richard Johnson Jr. Adaptive IIR filtering: current results and open issues. *IEEE Transaction on Information Theory*, IT-30(2):237–250, March 1984.

[108] J. R. Treichler. *Adaptive algorithms for infinite impulse response filters*, chapter 4, pages 60–90. Prentice-Hall, Englewood Cliffs, N.J., 1985.

[109] M. Nayeri and W. K. Jenkins. Alternative realization to adaptive IIR filters and properties of their performance surfaces. *IEEE Trans. on Circuits and Systems*, 36(4):485–496, April 1989.

[110] S. A. White. An adaptive recursive digital filter. In *Proc. 9th Asilomar Conf. Circuits Syst. Comput.*, pages 21–25, 1975.

[111] S. D. Stearns, G. R. Elliott, and N. Ahmed. On adaptive recursive filtering. In *Proc. 10th Asilomar Conf. Circuits Syst. Comput.*, pages 5–10, 1976.

[112] P. L. Feintuch. An adaptive recursive LMS filter. *Proceedings of IEEE*, 64:1622–1624, November 1976.

[113] M. G. Larimore, J. R. Treichler, and C. R. Johnson. SHARF: an algorithm for adapting IIR digital filters. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-28:428–440, August 1980.

[114] T. C. Hsia. A simplified adaptive recursive filter design. *Proceedings of IEEE*, 69:1153–1155, September 1981.

[115] H. Fan and W. K. Jenkins. A new adaptive IIR filter. *IEEE Trans. Circuits and Systems*, CAS-33:939–947, October 1986.

[116] J. Makhoul. Stable and efficient lattice methods for linear prediction. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-25:423–428, October 1977.

[117] D. Parikh, N. Ahmed, and S. D. Stearns. An adaptive lattice algorithm for recursive filters. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-28:110–112, Feb. 1980.

[118] I. L. Ayala. On a new adaptive lattice algorithm for recursive filters. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-30:316–319, April 1982.

[119] B. Friedlander. Lattice filters for adaptive processing. *Proc. of IEEE*, 70(8):829–867, August 1982.

[120] M. Nayeri and W. K. Jenkins. Analysis of alternate realizations of adaptive IIR filters. In *Proceedings of Int. Symp. Circuits ans Systems*, pages 2157–2160, 1988.

[121] D. A. Johns, W. M. Snelgrove, and A. S. Sedra. State-space adaptive recursive filters. In *Proc. of Int. Symp. Circuits and Systems*, pages 2153–2156, 1988.

[122] D. A. Johns, W. M. Snelgrove, and A. S. Sedra. Adaptive recursive state-space filters using a gradient-based algorithm. *IEEE Trans. Circuits and Systems*, 37(6), June 1990.

[123] J. Y. Zhang and W. Steenaart. Realization and implementation of adaptive state-space recursive filters. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, May 1989.

[124] J. Y. Zhang and W. Steenaart. Adaptive recursive state-space filtering. In *Proceedings of 33rd Midwest Symp. on Circuits and Systems*, August 1990.

[125] J. J. Shynk. Adaptive IIR filtering. *IEEE ASSP MAGAZINE*, 4–21, April 1989.

[126] W. Vetter. Matrix calculus operations and Taylor expansions. *SIAM Rev.*, 15(2):352–369, April 1978.

[127] S. A. H. Aly and M. M. Fahmy. Spatial-domain design of two-dimensional recursive digital filters. *IEEE Trans. Circuits and Systems*, CAS-27(10):892–901, October 1980.

[128] J. R. Treichler, C. R. Johnson Jr., and M. G. Larimore. *Theory and Design of Adaptive Filters*. John Wiley and Sons Inc., 1987.

[129] R. Chellappa and A. A. Sawchuk. *Digital Image Processing*. Volume 1, IEEE Computer Society Press, 1985.

[130] R. C. Gonzalez and P. Wintz. *Digital Image Processing*. Addison-Wesley Publishing Company, 1987.

[131] S. Haykin. *Adaptive Filter Theory.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986.

[132] H. G. Yeh. Systolic implementation on Kalman filters. *IEEE Transaction on Acoustics, Speech, and Signal Processing*, ASSP-36(9):1514–1517, September 1988.

[133] A. O. Aboutalib and L. M. Silverman. Restoration of motion degraded images. *IEEE Transaction on Circuits and Systems*, CAS-22(3):278–286, 1975.

[134] A. O. Aboutalib, M. S. Murphy, and L. M. Silverman. Digital restoration of images degraded by general motion blurs. *IEEE Transaction on Automat. Control*, AC-22:294–302, 1977.

[135] N. E. Nahi and T. Assefi. Bayesian recursive image estimation. *IEEE Transaction on Computer*, C-21:734–738, July 1972.

[136] J. W. Woods and C. H. Rademan. Kalman filtering in two dimensions. *IEEE Transaction on Information Theory*, IT-23:473–482, July 1977.

[137] J. W. Woods and V. K. Ingle. Kalman filtering in two dimensions: further results. *IEEE Transaction on Acoustics, Speech, Signal Processing*, ASSP-29:188–197, April 1981.

[138] M. S. Murphy and L. M. Silverman. Image model representation and line-by-line recursive restoration. *IEEE Transaction on Automat. Control*, AC-23:809–816, 1978.

[139] D. L. Angwin and H. Kaufman. Image restoration using reduced order models. *Signal Processing*, 16:21–28, 1989.

[140] B. R. Suresh and B. A. Shenoi. New results in two-dimensional Kalman filtering with applications to image restoration. *IEEE Transaction on Circuits and Systems*, CAS-28(4):307–319, April 1981.

[141] Z. Wu. Multidimensional state-space model Kalman filtering with application to image restoration. *IEEE Transaction on Acoustics, Speech, Signal Processing*, ASSP-33(6):1576–1592, December 1985.

[142] M. R. Azimi-Sadjadi and P. W. Wong. Two-dimensional block Kalman filtering for image restoration. *IEEE Transaction on Acoustics, Speech, Signal Processing*, ASSP-35(12):1736–1749, December 1987.

[143] A. Andrew. Parallel processing of the Kalman filter. In *Proc. Int. Conf. Parallel Processing*, pages 216–220, 1981.

[144] J. M. Jover and T. Kailath. A parallel architecture for Kalman filter measurement update and parameter estimation. *Automatic*, 22:43–57, 1986.

[145] T. Y. Sung and Y. H. Hu. Parallel VLSI implementation of the Kalman filter. *IEEE Transaction on Aerospace and Electronics Systems*, AES-23(2):215–224, March 1987.

[146] M. J. Chen and K. Yao. On realization of least-squares estimation and Kalman filtering by systolic arrays. In *Proc. Int. Workshop on Systolic Arrays*, July 1986.

[147] J. Y. Zhang and W. Steenaart. High speed Kalman filtering for image restoration. In *Proceedings of SPIE Vol. 1199 Visual Communications and Image Processing '89*, pages 125–135. November 1989.

[148] J. Y. Zhang and W. Steenaart. A very fast Kalman filter for image restoration. In *Proceedings of IEEE Int. Symp. on Circuits and Systems*, pages 250–253, May 1990.

[149] A. K. Jain. *Fundamentals of Digital Image Processing.* Prentice-Hall, 1989.

[150] A. Rosenfeld and A. C. Kak. *Digital Picture Processing.* Volume 2, Academic Press, 1982.

[151] H. Kaufman, J. W. woods, S. Dravida, and M. Tekalp. Estimation and identification of two-dimensional images. *IEEE Transactions on Automatic Control*, AC-28(7):745–756, 1983.

[152] J. W. Woods. Markov image modeling. *IEEE Transactions on Automatic Control*, AC-23(10):846–850, 1978.

[153] D. E. Dudgeon and R. M. Mersereau. *Multidimensional Digital Signal Processing*, chapter 5, pages 218–252. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

[154] S. M. Bozic. *Digital and Kalman Filtering.* Edward Arnold (publishers) Ltd., London, UK, 1979.