AD-A254 646

(2)

OFFICE OF NAVAL RESEARCH

FINAL TECHNICAL REPORT

STATE SPACE METHODS IN MULTIDIMENSIONAL DIGITAL SIGNAL PROCESSING

GRANT N0014-91-J-1151

R & T PROJECT: 4119354---07

DTIC
ELECTE
AUG 3 1 1992
S A D

Principal Investigator - Dr. Winser E. Alexander

This document has been approved for public release and sale; its distribution is unlimited.

Department of Electrical and Computer Engineering
Box 7911
North Carolina State University
Raleigh, NC 27695-7911

92-22642

39 pg

92 8 10 124

415455

Contents

DTIC QUALITY INSPECTED 5

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

Statement A per telecon Neil Gerr
ONR/Code 1111
Arlington, VA 22217-5000

NWW 8/2/92

Abstract

This report summarizes the results of an extensive research program on the real-time implementation of multidimensional (M-D) digital signal processing algorithms. We began our study on the efficient implementation of M-D digital filters. We mapped the M-D digital filter to a state space model because the state space model supports local data communications. We studied various approaches to implementing the state space model for M-D digital signal processing applications. We found that the best approach involves mapping the state space model onto a generalized linear finite state machine which facilitates the hardware implementation. Using this approach, we were able to develop a multiprocessor system architecture which is scalable, which is modular, and which has a high efficiency. Based upon these results, we developed the architecture for an application specific computing system which we call a Block Data Flow Architecture (BDFA). We are currently studying the mapping of several other M-D signal processing algorithms and matrix operations to the BDFA. These studies show that multiprocessor systems using the BDFA can achieve high throughput and high efficiency at a modest cost.

Chapter 1

Introduction.

1.1 Statement of the Problem.

Extensive research and development have been devoted to multidimensional (M-D) digital signal processing [1]. Recently, there has been a dramatic increase in the performance of computer systems. Thus, it has become more practical to implement many of the M-D digital signal processing applications in real-time. Practical applications of M-D digital signal processing include remote sensing, industrial inspection, robot vision, data compression for communications, processing biomedical images for diagnosis, character recognition, recognition of figure prints, weather forecasting, etc. In general, these applications are computationally intensive and require substantial data communications.

In many cases, the reduction of computer hardware cost makes it practical to design special purpose computer systems tailored to the specific requirements of a given class of algorithms. Systems with the computational capability to handle real-time or near real-time M-D digital signal processing are just becoming available as a result of these efforts. However, most M-D digital signal processing tasks are too complicated to implement in real-time using a single processor system. Thus, the development of M-D digital signal processing algorithms specifically designed for multiprocessor systems is an important research area.

In this research program, we have concentrated on the development of algorithms which can be effectively used for high speed, M-D digital signal processing in a multiprocessor or multicomputer environment. The traditional approach to research on the development of efficient algorithms for digital signal processing is to reduce the total number of multiplications (or complex multiplications) required. However, this approach is not valid for algorithms to be implemented on a state-of-the-art multiprocessor system. For example, the transfer of a data word between chips in a multiple chip system (typically on the order of 30

to 100 nanoseconds) can require as much time as required for a multiplication. Thus, data communications requirements should be given at least equal consideration to computational complexity in developing algorithms for multiprocessor systems.

We began our research program on the real-time implementation of two-dimensional (2-D) digital filters. We later generalized our results to include all discrete, linear, shift-invariant (DLSI) M-D systems. A DLSI system is a discrete system for which the system parameters do not vary with changes in the independent variables (time, space, distance, range, etc.). Thus, the coefficients for the finite difference equation representation of a DLSI system are constants. A finite difference equation expresses the result of a computation as a weighted average of current and previous inputs and past outputs. Quite often the independent and dependent variables are parameters such as time, space, range, temperature, etc. Many practical digital signal processing and digital control problems can be represented as DLSI systems. In addition, many shift variant systems can be approximated over small intervals as DLSI systems. Our approach has been to design computationally efficient algorithms which are optimized for implementing M-D DLSI systems in a multiprocessor environment. In this way, our results can be applied to a large variety of problems.

Real-time M-D digital signal processing has a wide range of applications such as radar and sonar signal processing, biomedical diagnosis, photography, broadcast television, computer vision, and seismology. Computational requirements of signal processing tasks such as beam-forming, adaptive filtering, data compression and parameter estimation can be reduced to a common set of matrix operations[2]. Matrix operations also find important applications in many areas such as oceanography, weather prediction, dynamic quantum field theory, aerodynamics, petroleum exploration, astrophysics, fluid mechanics, geophysics and particle physics. These applications require a system with high throughput and high efficiency for real-time implementation.

Normally, signal processing tasks and matrix operations possess a large amount of inherent parallelism. Many parallel algorithms and parallel structures have been developed to exploit this parallelism[3][4]. However, most parallel algorithms have been optimized for implementation on general purpose computers. General purpose computers can not achieve the high system throughput required for real-time processing because of limitations due to system management and control overhead and data communication problems. Data communications requirements are very important in developing multiprocessor implementations of these algorithms.

Most parallel multiprocessor system such as systolic arrays and hypercube multiprocessor systems have a synchronous SIMD structure. A synchronous system achieves its parallelism by synchronous clock-step operations [5]. This implies that all operands have to be ready before any processor can start its designated operation. This strictly synchronous operation imposes a severe timing restriction on the system design and causes implementation difficulties such as the clock skew problem for large scale systems. Thus, the throughput

rate of most multiprocessor systems fails to increase linearly proportional to the increase in the number of processors.

A wavefront array replaces the requirement for correct timing by a requirement for correct sequencing to overcome the globally synchronous timing problem[6]. However, if the handshaking for the wavefront array is done at the word level, then the resources required to implement the handshaking protocol limit the overall system efficiency and throughput. The BDFA is essentially a wavefront array with a block data handshaking protocol. Thus, the BDFA has the asynchronous timing advantages of the wavefront array but it can still have a very high efficiency because of the reduction of overhead due to handshaking for data communications.

Algorithms designed for systolic arrays and wavefront arrays use an algorithm partitioning strategy. In an algorithm partitioning strategy, each processor implements a different part of the algorithm and the total problem is solved using a pipeline. The use of this strategy may lead to unnecessary data movement among processing elements because only the edge processors have access to input or output devices. Processing results go through processor by processor in order to reach the one which can interface to the output device. This unnecessary data movement may increase system management and data communication overhead, and may increase hardware complexity. It also increases the data dependency among the processing elements.

While it is possible to obtain impressive performance with bus-organized multiprocessor systems and multiprocessor array systems for individual algorithms, the performance typically falls off due to data communication problems and/or synchronization problems as the number of processors is increased. In addition, hardware especially designed for a given algorithm either can not be adapted to solve other problems or the performance is drastically reduced on other problems. We have attempted to develop an application specific architecture which can solve a class of problems with high throughput and high efficiency. We expect this approach to result in a cost effective solution to demanding M-D digital signal processing problems.

1.2 Application Specific Computing Systems

Although the primary goal of our research program is the development of algorithms and computational structures for high performance M-D digital signal processing applications, a secondary goal of our research program is the development of application specific computing systems for digital signal processing with emphasis on real-time applications. We are especially interested in computationally intensive M-D digital signal processing applications such as beam-forming, M-D digital filtering, discrete transforms, adaptive filters, etc. Since many of these applications can be formulated as matrix operations, we include matrix operations

in the desired family of algorithms.

We developed the BDFA to have the flexibility to efficiently solve a variety of problems in this class of algorithms while still providing the high throughput for real-time applications. By exploiting the regularity and inherent parallelism in these applications, we found that many M-D signal processing and matrix algorithms can be solved using a data partitioning strategy. In a data partitioning strategy, each processor receives a different portion of the data and attempts to complete all of the necessary computations for its assigned data partition. We eliminate data communications to other processors when possible and minimize it when it is necessary.

We choose the data partitioning strategy for the BDFA to reduce data dependency between processors, to reduce interprocessor communication, and to simplify the interconnection network. In a BDFA, input (output) data can be moved directly into (from) any processor without interfering with any other processors. Thus, in this scheme, the interprocessor communications only involve the passing of intermediate computational results.

The interprocessor communications for the BDFA are in only one direction by design. This permits the use of FIFO buffers for interprocessor data communications. The FIFO buffers also provide asynchronous data communication capability which in turn relaxes the requirement for strict timing between processors. This is an important advantage as we increase the number of processors in a BDFA.

In mapping a given algorithm to a BDFA system, we try to minimize the interprocessor communications and data movements since they affect system throughput, system efficiency and system management overhead. Secondly, the BDFA maintains a direct input data channel to each processor and a direct output data channel from each processor to substantially reduce the required data movements for the processor array.

1.3 Overview of the Report.

This report presents a summary of the results achieved under Office of Naval Research contract N00014-83-K-0138. In chapter 2, we develop the state space representations for the 2-D and the M-D discrete linear shift-invariant (DLSI) systems. We use these state space representations to obtain the computational structure for real-time implementation on M-D DLSI systems. In chapter 3, we present the architectual features of the BDFA. We also present a performance evaluation of the use of the BDFA for the 2-D digital filters. The outstanding performance on this problem has encouraged us to consider the use of the BDFA for other M-D digital signal processing problems[8],[7].

Chapter 2

State Space Representation of M-D Digital Systems.

The state space representation provides the potential for minimizing the data communication requirements for a given algorithm without increasing computational complexity. Other advantages of the state space implementation over direct implementation include decreased sensitivity to parameter variations and improved performance when finite arithmetic is used.

A set of finite difference equations is one of the forms commonly used for representing DLSI systems. We have chosen this mathematical abstraction as a convenient starting point for development of the algorithm decomposition scheme for implementing the M-D DLSI system. The first step in the procedure is the state space representation of the system. Although we show the development of the state space representation from the finite difference equation, we can also obtain a state space representation from a signal flow graph or a block diagram representation. We use the state space representation as an intermediate form for representing the system. In order to clearly explain the concepts involved in this approach, we first discuss the state space implementation of 2-D DLSI systems. We then show that the concepts used in the 2-D case can be extended to the M-D case ($M > 2$).

2.1 State Space Representation of 2-D DLSI Systems

The general-order, causal 2-D finite difference equation with quarter-plane support is given by [1].

$$g(n_1,n_2) = \sum_{j_1=0}^{L_1}\sum_{j_2=0}^{L_2} b(j_1,j_2)f(n_1-j_1,n_2-j_2) - \sum_{\substack{j_1=0\\ j_1+j_2>0}}^{L_1}\sum_{j_2=0}^{L_2} a(j_1,j_2)g(n_1-j_1,n_2-j_2) \quad (2.1)$$

The parameters $a(j_1, j_2)$ and $b(j_1, j_2)$ in the above equation are coefficients which determine the characteristics of the algorithm. Since the coefficients can take on arbitrary values, this equation can represent many 2-D problems including spatial domain filters, image processing, simulation, control systems, etc. The state space approach can be extended to the 2-D DLSI system [9] [10]. For the 1-D case, a simularity transformation can be used to optimize the state-space representation for a given criteria. However, there is a fundamental problem in extending this concept to the 2-D case because an arbitrary bivariate transfer function cannot be factored into distinct poles and zeros and cannot be expanded into partial fractions. Thus, these approaches to developing a parallel or cascade implementation are not extendible to M-D systems due to the lack of a fundamental theory of algebra for M-D systems.

Roesser's state space model for 2-D DLSI systems is perhaps the most widely accepted model [9]. This model provides for the update of the next state for a set of vertical state variables and a set of horizontal state variables as a linear combination of the present vertical and horizontal state variables and the current input. The output is also a linear combination of the present vertical and horizontal state variables and the current input.

$$\begin{aligned} \begin{bmatrix} S_H(n_1+1, n_2) \\ S_V(n_1, n_2+1) \end{bmatrix} &= \begin{bmatrix} \mathbf{A_1} & \mathbf{A_2} \\ \mathbf{A_3} & \mathbf{A_4} \end{bmatrix} \begin{bmatrix} S_H(n_1, n_2) \\ S_V(n_1, n_2) \end{bmatrix} + \begin{bmatrix} \mathbf{B_1} \\ \mathbf{B_2} \end{bmatrix} [f(n_1, n_2)] \\ [g(n_1, n_2)] &= [\mathbf{C_1} \quad \mathbf{C_2}] \begin{bmatrix} S_H(n_1, n_2) \\ S_V(n_1, n_2) \end{bmatrix} + \mathbf{D}\,[f(n_1, n_2)] \end{aligned} \tag{2.2}$$

Roesser's state space model is based upon assigning state variables to the output of the delay elements. We find it more convenient to assign state variables to the input of the delay elements. This makes the state space representation compatible with the evential hardware implementation because a state variable identifies a parameter which must be stored for later use. This alternate choice for the state variables is equivalent to the parameter substitution:

$$\begin{aligned} Q_H(n_1, n_2) &= S_H(n_1+1, n_2) \\ Q_V(n_1, n_2) &= S_V(n_1, n_2+1) \end{aligned} \tag{2.3}$$

With this substitution, the indices for the modified state vector are the same as those for the current input. Thus, the modified model is conceptually simpler because it more closely resembles the finite difference equation model. It also simplifies our later derivations for the block-state model and the development of the initial conditions models.

We can combine the vertical state variables and the horizontal state variables into a state vector for a given location in the M-D array. Thus,

$$Q(n_1, n_2) = \begin{bmatrix} Q_H(n_1, n_2) \\ Q_V(n_1, n_2) \end{bmatrix} \tag{2.4}$$

We can then update this state vector and compute the current output using a linear combination of the most recent vertical state variables, the most recent horizontal state variables

and the current input. The revised model is equivalent to Roesser's original model. However, the notation more accurately reflects the computational model and the resulting architecture presented in this report.

The modified state model for the causal 2-D DLSI system with quarter plane support is given by

$$\begin{bmatrix} Q_H(n_1,n_2) \\ Q_V(n_1,n_2) \end{bmatrix} = \begin{bmatrix} \mathbf{A_1} & \mathbf{A_2} \\ \mathbf{A_3} & \mathbf{A_4} \end{bmatrix} \begin{bmatrix} Q_H(n_1-1,n_2) \\ Q_V(n_1,n_2-1) \end{bmatrix} + \begin{bmatrix} \mathbf{B_1} \\ \mathbf{B_2} \end{bmatrix} [f(n_1,n_2)]$$

$$[g(n_1,n_2)] = [\mathbf{C_1} \quad \mathbf{C_2}] \begin{bmatrix} Q_H(n_1-1,n_2) \\ Q_V(n_1,n_2-1) \end{bmatrix} + \mathbf{D}\,[f(n_1,n_2)] \qquad (2.5)$$

In this equation, $Q_H(n_1,n_2)$ is a column vector whose elements are the current values of the state variables for the horizontal processing direction corresponding to the index n_1. $Q_V(n_1,n_2)$ is a column vector whose elements are the current values of the state variables for the vertical processing direction corresponding to the index n_2. The index (n_1-1,n_2) implies a delay in the horizontal direction and the index (n_1,n_2-1) implies a delay in the vertical direction. $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$, $\mathbf{A_4}$, $\mathbf{B_1}$, $\mathbf{B_2}$, $\mathbf{C_1}$, $\mathbf{C_2}$, and $\mathbf{D}$ are appropriate coefficient matrices such that Eq. 2.1 and Eq. 2.5 are equivalent. Fig. 2.1 gives a block diagram for a linear finite state machine which is equivalent to the state space representation of the 2-D DLSI system given in Eq. 2.5. The linear finite state machine for Eq. 2.2 is identical except for the designation of variables as specified in Eq 2.3.

A state variable represents information that must be stored for later use. Therefore, it is important to select state variables that minimize data communication requirements without increasing computational complexity. In a typical image processing application, a horizontal delay represents a storage of one word while a vertical delay represents a storage of an entire row of data. Therefore we selected a canonical form which minimizes the number of vertical state variables.

The state space representation for a given 2-D DLSI system is not unique. In addition, the problem of defining a representation with the minimum number of states has not been solved [11]. We choose a particular canonical form to facilitate the development of a computational primitive for 2-D DLSI systems. We then assign state variables to the inputs of the delay elements to obtain the state variable representation. The procedures which we use are general and can be applied to obtain a state space representation from any signal flow graph or block diagram.

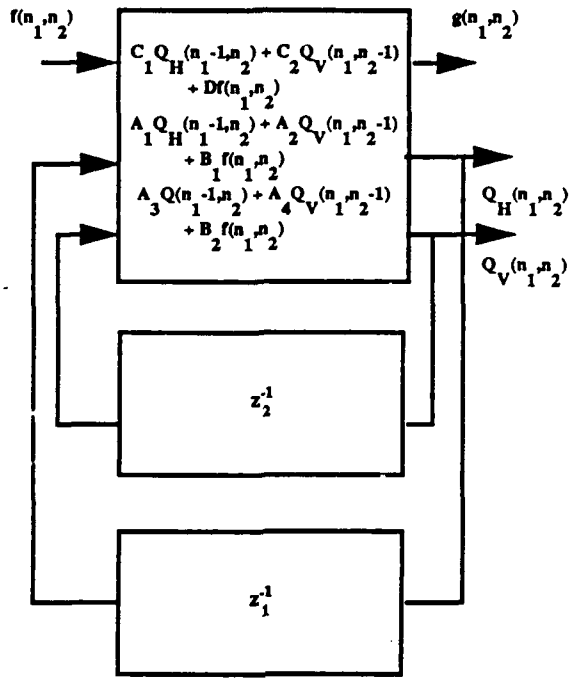

Figure 2.1: Two-dimensional generalized finite state machine.

2.1.1 Deriving the 2-D State Space Equations.

The 2-D transfer function corresponding to Eq. 2.1 is given by

$$H(z_1, z_2) = \frac{\sum_{j_1=0}^{L_1} \sum_{j_2=0}^{L_2} b(j_1, j_2) z_1^{-j_1} z_2^{-j_2}}{1 + \sum_{\substack{j_1=0 \\ j_1+j_2>0}}^{L_1} \sum_{j_2=0}^{L_2} a(j_1, j_2) z_1^{-j_1} z_2^{-j_2}} \tag{2.6}$$

Note that $H(z_1, z_2)$ describes an input/output relationship between the transform of the input sequence, $F(z_1, z_2)$, and the transform of the output sequence, $G(z_1, z_2)$. We can show

this relationship as follows:

$$G(z_1, z_2) = b(0,0)F(z_1, z_2) + \sum_{\substack{j_2=0 \\ j_1+j_2>0}}^{L_2} \sum_{j_1=0}^{L_1} [b(j_1, j_2)F(z_1, z_2) - a(j_1, j_2)G(z_1, z_2)]\, z_1^{-j_1} z_2^{-j_2} \tag{2.7}$$

Fig. 2.2 gives a block diagram representation of a 2-D filter partitioned as specified by Eq. 2.7. Note that the number of vertical delays is the same as the order of the filter in the

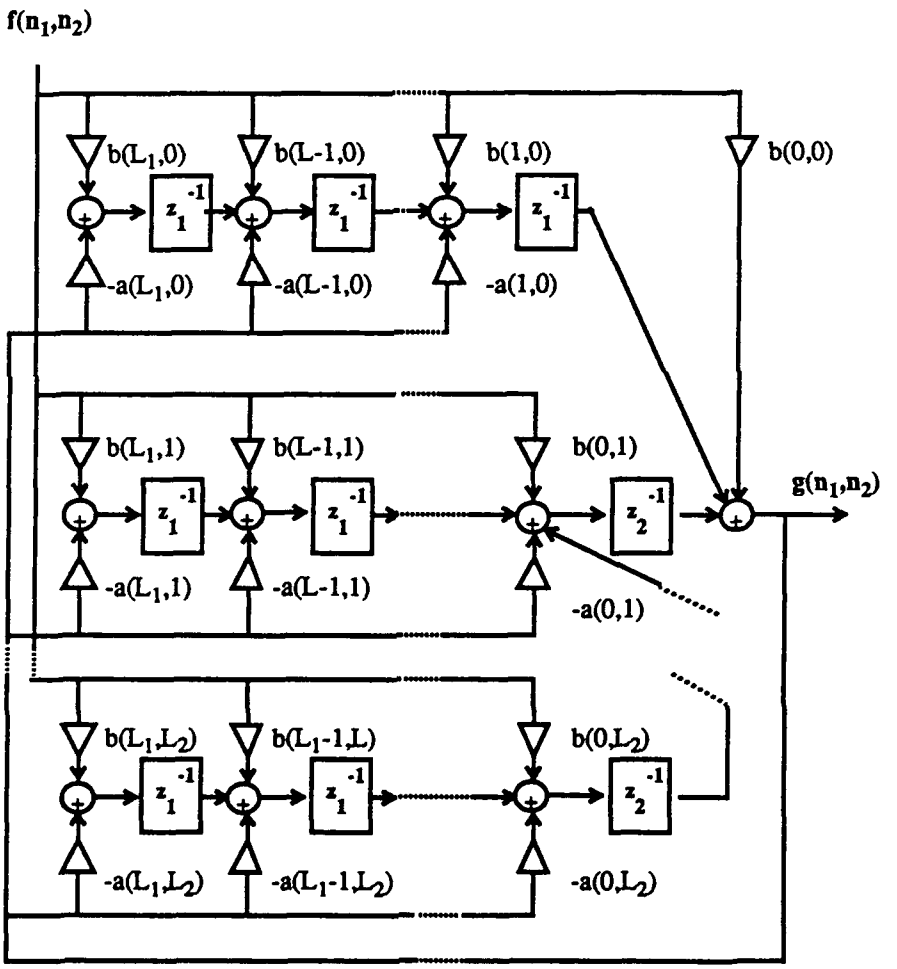

Figure 2.2: Block diagram representation of a 2-D system.

z_2 variable which is the minimum possible number as desired. We can obtain the desired state space representation by assigning a horizontal state variable to the input of each of the

horizontal delay blocks (associated with the z_1 variable) and a vertical state variable to each of the vertical delay blocks (associated with the z_2 variable). We then write the resulting equations in matrix form as given in Eq. 2.5.

Fig. 2.3 gives a section of the block diagram of the 2-D DLSI system having one horizontal delay and one vertical delay. Assigning state variables as described above, the

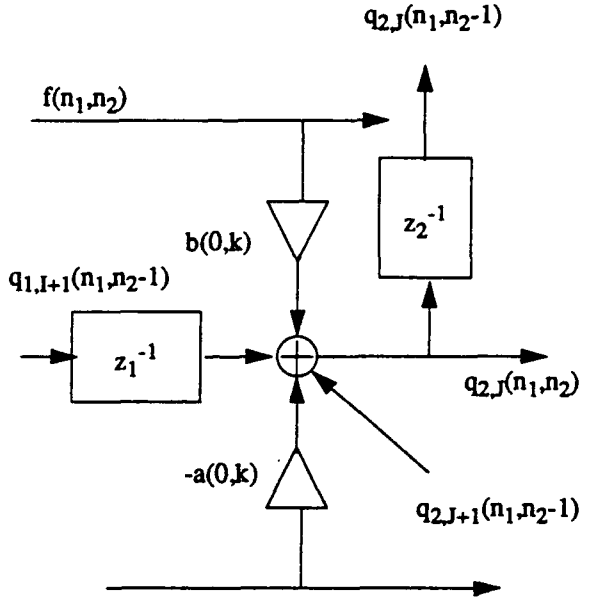

Figure 2.3: Section of the block diagram of a 2-D DLSI system.

typical vertical state equation for the 2-D DLSI system can be represented as

$$\begin{aligned} q_{2,I_2}(n_1,n_2) &= b(0,j_2)f(n_1,n_2) - a(0,j_2)g(n_1,n_2) \\ &+ q_{1,I_1}(n_1-1,n_2) + q_{2,I_2+1}(n_1,n_2-1)\ ;\ 1 \le j_2 \le L_2 \\ I_1 &= L_1 j_2 + 1 \\ I_2 &= j_2 \\ q_{2,L2+1}(n_1,n_2-1) &= 0 \end{aligned} \tag{2.8}$$

In a similar way, the typical horizontal state variable is given by

$$\begin{aligned} q_{1,I_1}(n_1,n_2) &= b(j_1,j_2)f(n_1,n_2) - a(j_1,j_2)g(n_1,n_2) \\ &+ q_{1,I_1+1}(n_1-1,n_2);\ 1 \le j_1 \le L_1-1\ ;\ 0 \le j_2 \le L_2 \\ I_1 &= j_2 L_1 + j_1\ . \end{aligned} \tag{2.9}$$

$$\begin{aligned} q_{1,I_1}(n_1,n_2) &= b(j_1,j_2)f(n_1,n_2) - a(j_1,j_2)g(n_1,n_2)\ ; \\ j_1 &= L_1\ ;\ 0 \le j_2 \le L_2\ . \\ I_1 &= (j_2+1)L_1\ . \end{aligned} \tag{2.10}$$

The output equation is given by

$$g(n_1,n_2) = b(0,0)f(n_1,n_2) + q_{1,1}(n_1-1,n_2) + q_{2,1}(n_1,n_2-1) \tag{2.11}$$

The vertical and horizontal state variables can then be represented by [12]

$$\begin{aligned} q_{i,I_i}(n_1,n_2) &= \widehat{b_{i,I_i}}f(n_1,n_2) - \widehat{a_{i,I_i}}g(n_1,n_2) + q_{i,I_i+1}(n_i-1,n_{\bar{i}}) \\ &+ q_{\bar{i},I_{\bar{i}}}(n_i,n_{\bar{i}}-1) \end{aligned} \tag{2.12}$$

If $i=1$ then $\bar{i}=2$ and vice versa. Note that if $i=1$ in Eq. 2.12, then the corresponding vertical state variable is equal to zero $[q_{\bar{i},I_{\bar{i}}}(n_i,n_{\bar{i}}-1)=0]$.

Eq. 2.12 is a computational primitive for the 2-D DLSI system since the vertical state variables, the horizontal state variables and the output can be mapped into this equation with a suitable interchange of variables. In using Eq. 2.12 as a computational primitive, $q_{i,I_i}(n_1,n_2)$ can represent the current value of the horizontal or vertical state variable or the output as appropriate, $q_{i,I_i-1}(n_i-1,n_{\bar{i}})$ represents a previous value of a horizontal state variable (delayed by one pixel), $q_{\bar{i},I_{\bar{i}}}(n_i,n_{\bar{i}}-1)$ represents a previous value of a vertical state variable (delayed by one row). We can implement this equation in a tree structure using two multipliers and three adders [12] as shown in Fig. 2.4 or in two steps using a multiplier-adder.

2.2 State Space Representation of M-D DLSI Systems.

We now discuss the extension of the 2-D state space implementation presented above to M-D DLSI systems. The general multivariable difference equation for the causal, DLSI system with first section support (the M-D equivalent of quarter plane support) is given by [1]

$$\begin{aligned} g(\mathbf{n}) &= \sum_{j_1=0}^{L_1}\sum_{j_2=0}^{L_2}\cdots\sum_{j_M=0}^{L_M} b(\mathbf{J})f(n_1-j_1,\cdots,n_M-j_M) \\ &- \sum_{\substack{j_1=0\\ j_1+j_2+\cdots+j_M>0}}^{L_1}\sum_{j_2=0}^{L_2}\cdots\sum_{j_M=0}^{L_M} a(\mathbf{J})g(n_1-j_1,\cdots,n_M-j_M) \\ \mathbf{n} &= n_1,n_2,\cdots,n_M\ ;\ \mathbf{J} = j_1,j_2,\cdots,j_M \end{aligned} \tag{2.13}$$

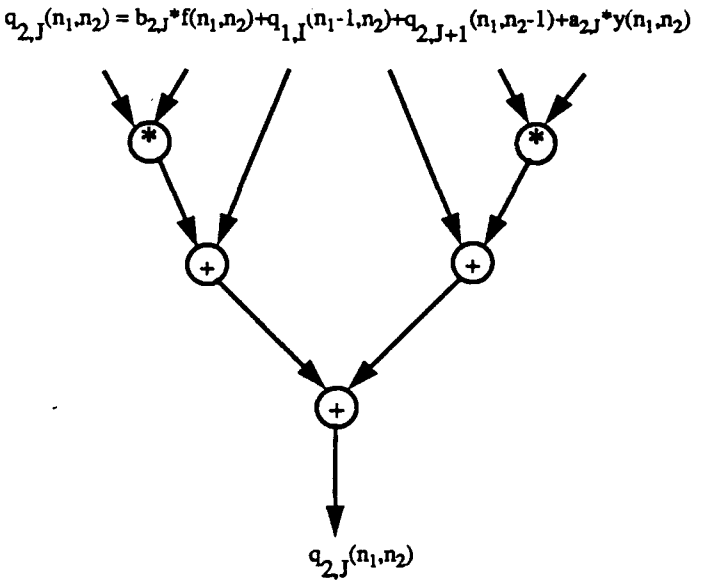

Figure 2.4: Tree structure for the computational primitive..

The input $f(\mathbf{n})$ is assumed to be sampled at uniform intervals in each of the independent variables and $g(\mathbf{n})$ is the corresponding output. The parameters $a(\mathbf{J})$ and $b(\mathbf{J})$ are coefficients which determine the characteristics of the algorithm. Since the coefficients can take on arbitrary values as appropriate, this equation can represent many common M-D problems.

The state space representation of the M-D DLSI system is given by [13]

$$\begin{aligned}
\begin{bmatrix} S_1(n_1+1,n_2,\cdots,n_M) \\ S_2(n_1,n_2+1,\cdots,n_M) \\ \vdots \\ S_M(n_1,n_2,\cdots,n_M+1) \end{bmatrix} &= \begin{bmatrix} \mathbf{A_{11}} & \mathbf{A_{12}} & \cdots & \mathbf{A_{1M}} \\ \mathbf{A_{21}} & \mathbf{A_{22}} & \cdots & \mathbf{A_{2M}} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{A_{M1}} & \mathbf{A_{M2}} & \cdots & \mathbf{A_{MM}} \end{bmatrix} \begin{bmatrix} S_1(\mathbf{n}) \\ S_2(\mathbf{n}) \\ \vdots \\ S_M(\mathbf{n}) \end{bmatrix} \\
&+ \begin{bmatrix} \mathbf{B_1} \\ \mathbf{B_2} \\ \vdots \\ \mathbf{B_M} \end{bmatrix} [f(\mathbf{n})] \\
[g(\mathbf{n})] &= [\mathbf{C_1}\ \mathbf{C_2}\ \cdots\ \mathbf{C_M}] \begin{bmatrix} S_1(\mathbf{n}) \\ S_2(\mathbf{n}) \\ \vdots \\ S_M(\mathbf{n}) \end{bmatrix} \\
&+ \mathbf{D}\,[f(\mathbf{n})]
\end{aligned} \tag{2.14}$$

We choose state variables for the M-D case in the same manner as we did with the 2-D case. This is equivalent to the following parameter substitution :

$$Q_i(\cdots, n_i \cdots) \;=\; S_i(\cdots, n_i + 1, \cdots) \;\;;\;\; 1 \;\leq\; i \;\leq\; M \;. \tag{2.15}$$

With this substitution, the indices for the modified state vectors are the same as those for the current input and the state variables are updated as a linear combinations of the delayed state variables and the current input. Thus, the theoretical model more closely matches the computational model and is consistent with the difference equation notation. The resulting state space representation for the M-D DLSI system is given by

$$\begin{aligned}
\begin{bmatrix} Q_1(\mathbf{n}) \\ Q_2(\mathbf{n}) \\ \vdots \\ Q_M(\mathbf{n}) \end{bmatrix} &= \begin{bmatrix} \mathbf{A_{11}} & \mathbf{A_{12}} & \cdots & \mathbf{A_{1M}} \\ \mathbf{A_{21}} & \mathbf{A_{22}} & \cdots & \mathbf{A_{2M}} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{A_{M1}} & \mathbf{A_{M2}} & \cdots & \mathbf{A_{MM}} \end{bmatrix} \begin{bmatrix} Q_1(n_1 - 1, n_2, ..., n_M) \\ Q_2(n_1, n_2 - 1, ..., n_M) \\ \vdots \\ Q_M(n_1, n_2, ..., n_M - 1) \end{bmatrix} \\
&+ \begin{bmatrix} \mathbf{B_1} \\ \mathbf{B_2} \\ \vdots \\ \mathbf{B_M} \end{bmatrix} [f(\mathbf{n})] \\
[g(\mathbf{n})] &= [\mathbf{C_1} \;\; \mathbf{C_2} \;\ldots\; \mathbf{C_M}] \begin{bmatrix} Q_1(n_1 - 1, n_2, ..., n_M) \\ Q_2(n_1, n_2 - 1, ..., n_M) \\ \vdots \\ Q_M(n_1, n_2, ..., n_M - 1) \end{bmatrix} \\
&+ \mathbf{D}\,[f(\mathbf{n})]
\end{aligned} \tag{2.16}$$

We can use the approach we used for the 2-D DLSI system to obtain a state space representation of the M-D DLSI system. First, we obtain a suitable computational graph for the M-D system. Then we assign the input to each delay as a state variable in the corresponding tuple. In the development of the M-D state space implementation that follows, we have chosen a canonical form which minimizes the number of state variables in the *Mth* tuple. This is comparable to choosing a canonical form to minimize the number of veritcal state variables in the 2-D DLSI system.

If we express the state equations and the output equation for the M-D system in matrix form, then we have the M-D state space model as given in Eq. 2.16. For convenience, we define

$$Q(\mathbf{n}) \;=\; \begin{bmatrix} Q_1(\mathbf{n}) \\ Q_2(\mathbf{n}) \\ \vdots \\ Q_M(\mathbf{n}) \end{bmatrix}$$

where k_{i-1} and k_m may either be 0 or 1 depending upon whether the associated delayed state variable appears in the equation for state variable $q_{i,I-1}(\mathbf{n})$ or q_{1,I_m} respectively.

Eq. 2.30 is a computational primitive for the M-D DLSI system since the state variables for each tuple and the output can be mapped into it with a suitable interchange of variables. Also note that if $\widehat{a_{i,I_i}}$ is equal to zero, then Eq. 2.30 becomes a computational primitive for the M-D FIR DLSI system. On the left side of the equal sign, $q_{i,I_i}(\mathbf{n})$ can represent the current value of the state variable in any tuple or the output as appropriate. The state variables on the right side of the equal sign are delayed by one element in the respective tuple. Thus, Eq. 2.30 is a generalization of the 2-D computational primitive as given in Eq. 2.12. Note that only 2 multiplications and a maximum of $M + 1$ additions are required to compute any of the state variables or the output for a M-D system.

2.3 A 2-D Example.

Consider the second order 2-D digital filter with transfer function given by

$$H(z_1, z_2) = \frac{\sum_{j_1=0}^{2}\sum_{j_2=0}^{2} b(j_1, j_2) z_1^{-j_1} z_2^{-j_2}}{1 + \sum_{\substack{j_1=0 \\ j_1+j_2>0}}^{2}\sum_{j_2=0}^{2} a(j_1, j_2) z_1^{-j_1} z_2^{-j_2}} \tag{2.31}$$

Using Eq. 2.30, we can write state equations as follows:

$$\begin{aligned}
y(n_1, n_2) &= q_{1,1}(n_1 - 1, n_2) + q_{2,1}(n_1, n_2 - 1) \\
g(n_1, n_2) &= \widehat{b_{0,0}} f(n_1, n_2) + y(n_1, n_2) \\
q_{1,1}(n_1, n_2) &= \widehat{b_{1,1}} f(n_1, n_2) + \widehat{a_{1,1}} y(n_1, n_2) + q_{1,2}(n_1 - 1, n_2) \\
q_{1,2}(n_1, n_2) &= \widehat{b_{1,2}} f(n_1, n_2) + \widehat{a_{1,2}} y(n_1, n_2) \\
q_{1,3}(n_1, n_2) &= \widehat{b_{1,3}} f(n_1, n_2) + \widehat{a_{1,3}} y(n_1, n_2) + q_{1,4}(n_1 - 1, n_2) \\
q_{1,4}(n_1, n_2) &= \widehat{b_{1,4}} f(n_1, n_2) + \widehat{a_{1,4}} y(n_1, n_2) \\
q_{1,5}(n_1, n_2) &= \widehat{b_{1,5}} f(n_1, n_2) + \widehat{a_{1,5}} y(n_1, n_2) + q_{1,6}(n_1 - 1, n_2) \\
q_{1,6}(n_1, n_2) &= \widehat{b_{1,6}} f(n_1, n_2) + \widehat{a_{1,6}} y(n_1, n_2) \\
q_{2,1}(n_1, n_2) &= \widehat{b_{2,1}} f(n_1, n_2) + \widehat{a_{2,1}} y(n_1, n_2) + q_{1,6}(n_1 - 1, n_2) \\
&+ q_{2,2}(n_1, n_2 - 1) \\
q_{2,1}(n_1, n_2) &= \widehat{b_{2,2}} f(n_1, n_2) + \widehat{a_{2,2}} y(n_1, n_2) + q_{1,3}(n_1 - 1, n_2)
\end{aligned} \tag{2.32}$$

The coefficients are given by

$$\begin{aligned}
\widehat{b_{0,0}} &= b(0,0) \\
\widehat{b_{1,1}} &= b(1,0) - b(0,0)a(1,0) \\
\widehat{a_{1,1}} &= -a(1,0) \\
\widehat{b_{1,2}} &= b(2,0) - b(0,0)a(2,0) \\
\widehat{a_{1,2}} &= -a(2,0) \\
\widehat{b_{1,3}} &= b(1,1) - b(0,0)a(1,1) \\
\widehat{a_{1,3}} &= -a(1,1) \\
\widehat{b_{1,4}} &= b(2,1) - b(0,0)a(2,1) \\
\widehat{a_{1,4}} &= -a(2,1) \\
\widehat{b_{1,5}} &= b(1,2) - b(0,0)a(1,2) \\
\widehat{a_{1,5}} &= -a(1,2) \\
\widehat{b_{1,6}} &= b(2,2) - b(0,0)a(2,2) \\
\widehat{a_{1,6}} &= -a(2,2) \\
\widehat{b_{2,1}} &= b(0,1) - b(0,0)a(0,1) \\
\widehat{a_{2,1}} &= -a(0,1) \\
\widehat{b_{2,2}} &= b(0,2) - b(0,0)a(0,2) \\
\widehat{a_{2,2}} &= -a(0,2)
\end{aligned} \tag{2.33}$$

Thus, we can write

$$\widehat{\mathbf{A_1}} = \begin{bmatrix}
\widehat{a_{1,1}} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\widehat{a_{1,2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\widehat{a_{1,3}} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\widehat{a_{1,4}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\widehat{a_{1,5}} & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\widehat{a_{1,6}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\widehat{a_{2,1}} & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\widehat{a_{2,2}} & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix} \tag{2.34}$$

$$\widehat{\mathbf{A_2}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & \widehat{a_{1,1}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \widehat{a_{1,2}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \widehat{a_{1,3}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \widehat{a_{1,4}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \widehat{a_{1,5}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \widehat{a_{1,6}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \widehat{a_{2,1}} & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & \widehat{a_{2,2}} & 0
\end{bmatrix} \tag{2.35}$$

$$\hat{\mathbf{B}} = \begin{bmatrix} \widehat{b_{1,1}} \\ \widehat{b_{1,2}} \\ \widehat{b_{1,3}} \\ \widehat{b_{1,4}} \\ \widehat{b_{1,5}} \\ \widehat{b_{1,6}} \\ \widehat{b_{2,1}} \\ \widehat{b_{2,2}} \end{bmatrix} \tag{2.36}$$

$$\widehat{\mathbf{C_1}} = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] \tag{2.37}$$

$$\widehat{\mathbf{C_2}} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0] \tag{2.38}$$

$$\mathbf{D} = [b(0,0)] \tag{2.39}$$

2.4 Computation of Initial Conditions.

Many practical digital systems require the use of appropriate boundary values or initial conditions. The classical approach of assigning a value of zero to boundary values often leads to undesirable transients during initialization. Our approach to the initial condition problem involves the state space model and the estimation of the initial state using the constraint that the state does not change upon applying the initial inputs on the boundary. Using this constraint, the initial state can be determined from the relationship

$$Q(\mathbf{O}) = Q(-1,0,\ldots,0) = Q(0,-1,0,\ldots,0) \ldots \tag{2.40}$$

It follows from the use of Eq. 2.22 that

$$\begin{aligned} Q(\mathbf{O}) &= \sum_{i=1}^{M} \widehat{\mathbf{A}_i}\mathbf{Q}(\mathbf{O}) + \mathbf{B}f(\mathbf{O}) \\ g(\mathbf{O}) &= \sum_{i=1}^{M} \widehat{\mathbf{C}_i}\mathbf{Q}(\mathbf{O}) + \mathbf{D}f(\mathbf{O}) \end{aligned} \tag{2.41}$$

Since

$$\sum_{i=1}^{M} \widehat{\mathbf{A}_i} = \mathbf{A}\,, \tag{2.42}$$

we can write

$$\begin{aligned} Q(\mathbf{O}) &= [I - \mathbf{A}]^{-1}\mathbf{B}f(\mathbf{O}) \\ g(\mathbf{O}) &= \left[\mathbf{C}[I - \mathbf{A}]^{-1}\mathbf{B} + \mathbf{D}\right] f(\mathbf{O}) \end{aligned} \tag{2.43}$$

Thus, we can compute the initial state vector and the initial output by using only the initial input.

Using the constraint above on the initial conditions, we can compute the state and the output along any boundary. Consider the use of this constraint along the boundary where the index for tuple k has a value of zero. For this case, we have

$$Q(n_1, n_2, \ldots, n_k, \ldots, n_M) \;=\; Q(n_1, n_2, \ldots, n_k - 1, \ldots, n_M)\;;\;\; n_k \;=\; 0 \tag{2.44}$$

The state equation for this boundary is given by

$$Q(\mathrm{n}) \;=\; \widehat{\mathbf{A_k}} Q(\mathrm{n}) \;+\; \sum_{\substack{i=1 \\ i \neq k}}^{M} \widehat{\mathbf{A_i}}\; \mathbf{Q}(\mathrm{n} - r_i^{-1}) \;+\; \mathbf{B} f(\mathrm{n})\;;\;\; n_k \;=\; 0 \tag{2.45}$$

Thus,

$$Q(\mathrm{n}) \;=\; \left[I \;-\; \widehat{\mathbf{A_k}}\right]^{-1} \sum_{\substack{i=1 \\ i \neq k}}^{M} \widehat{\mathbf{A_i}}\; \mathbf{Q}(\mathrm{n} - r_i^{-1}) \;+\; \left[I \;-\; \widehat{\mathbf{A_k}}\right]^{-1} \mathbf{B} f(\mathrm{n})\;;\;\; n_k \;=\; 0 \tag{2.46}$$

The corresponding output equation is given by

$$\begin{aligned} g(\mathrm{n}) \;\;=\;\; & \sum_{\substack{i=1 \\ i \neq k}}^{M} \widehat{\mathbf{C_i}}\; \mathbf{Q}(\mathrm{n} - r_i^{-1}) \;+\; \mathbf{C_k} \left[I \;-\; \widehat{\mathbf{A_k}}\right]^{-1} \sum_{\substack{i=1 \\ i \neq k}}^{M} \widehat{\mathbf{A_i}}\; \mathbf{Q}(\mathrm{n} - r_i^{-1}) \\ & +\;\; \mathbf{C_k} \left[I \;-\; \widehat{\mathbf{A_k}}\right]^{-1} \mathbf{B} f(\mathrm{n}) \;+\; \mathbf{D} f(\mathrm{n}) \end{aligned} \tag{2.47}$$

2.4.1 Initial Conditions Example.

We now show the computation of initial conditions for a second order 2-D IIR filter as an example. We derived the coefficient matrices for this case in a previous example. The state space representation for this filter is given by

$$\begin{aligned} Q(n_1, n_2) \;&=\; \widehat{\mathbf{A_1}} Q(n_1 - 1, n_2) \;+\; \widehat{\mathbf{A_2}} Q(n_2, n_2 - 1) \;+\; \hat{\mathbf{B}} f(n1, n2) \\ g(n_1, n_2) \;&=\; \widehat{\mathbf{C_1}} Q(n_1 - 1, n_2) \;+\; \widehat{\mathbf{C_2}} Q(n_2, n_2 - 1) \;+\; \mathbf{D} f(n1, n2) \end{aligned} \tag{2.48}$$

Let the numerator polynomial for the 2-D transfer function $H(\mathbf{z})$ be given by

$$\begin{aligned} N(z_1, z_2) \;\;=\;\; & 0.0427 \;+\; 0.0853 z_1^{-1} \;+\; 0.0427 z_1^{-2} \\ & +\;\; 0.0853 z_2^{-1} \;+\; 0.1707 z_1^{-1} z_2^{-1} \;+\; 0.0853 z_1^{-2} z_2^{-1} \\ & +\;\; 0.0427 z_2^{-2} \;+\; 0.0853 z_1^{-1} z_2^{-2} \;+\; 0.0427 z_1^{-2} z_2^{-2} \end{aligned} \tag{2.49}$$

Let the denominator polynomial for $H(\mathbf{z})$ be given by

$$\begin{aligned} D(z_1, z_2) &= 1.0 - 0.3695 z_1^{-1} + 0.1958 z_2^{-2} \\ &- 0.3695 z_2^{-1} + 0.1366 z_1^{-1} z_2^{-1} - 0.0724 z_1^{-2} z_2^{-1} \\ &+ 0.1958 z_2^{-2} - 0.0724 z_1^{-1} z_2^{-2} + 0.0383 z_1^{-2} z_2^{-2} \end{aligned} \tag{2.50}$$

The coefficient matrices corresponding to Eq. 2.22 are given by

$$\widehat{\mathbf{A}} = \begin{bmatrix} 0.3695 & 1 & 0 & 0 & 0 & 0 & 0.3695 & 0 \\ -0.1958 & 0 & 0 & 0 & 0 & 0 & -0.1958 & 0 \\ -0.1366 & 0 & 0 & 1 & 0 & 0 & -0.1366 & 0 \\ 0.0724 & 0 & 0 & 0 & 0 & 0 & 0.0724 & 0 \\ 0.0724 & 0 & 0 & 0 & 0 & 1 & 0.0724 & 0 \\ -0.0383 & 0 & 0 & 0 & 0 & 0 & -0.0383 & 0 \\ 0.3695 & 0 & 1 & 0 & 0 & 0 & 0.3695 & 1 \\ -0.1958 & 0 & 0 & 0 & 1 & 0 & -0.1958 & 0 \end{bmatrix} \tag{2.51}$$

$$\widehat{\mathbf{A_1}} = \begin{bmatrix} 0.3695 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.1958 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.1366 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0.0724 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0724 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -0.0383 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.3695 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -0.1958 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.52}$$

$$\widehat{\mathbf{A_2}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0.3695 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.1958 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.1366 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0724 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0724 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.0383 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3695 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.1958 & 0 \end{bmatrix} \tag{2.53}$$

The initial state vector at $n_1 = n_2 = 0$ is given by

$$Q(0,0) = [\mathbf{I} - \mathbf{A}]^{-1}\mathbf{B}\, f(0,0) \tag{2.54}$$

The corresponding initial output is given by

$$g(0,0) = \left[[\widehat{\mathbf{C_1}} + \widehat{\mathbf{C_2}}][\mathbf{I} - \mathbf{A}]^{-1}\mathbf{B} + \mathbf{D}\right] f(0,0) \tag{2.55}$$

Thus,

$$g(0,0) \;=\; f(0,0) \tag{2.56}$$

where

$$\left[[\widehat{\mathbf{C_1}} \;+\; \widehat{\mathbf{C_2}}][\mathbf{I} \;-\; \mathbf{A}]^{-1}\mathbf{B} \;+\; \mathbf{D}\right] \;=\; 1.0 \tag{2.57}$$

For the first row, we have

$$Q(n_1,-1) \;=\; Q(n_1,0) \tag{2.58}$$

Using this assumption, we obtain

$$Q(n_1,0) \;=\; [\mathbf{I} \;-\; \widehat{\mathbf{A_2}}]^{-1}\,\widehat{\mathbf{A_1}}Q(n_1-1,0) \;+\; [\mathbf{I} \;-\; \widehat{\mathbf{A_2}}]^{-1}\,\mathbf{B}\;f(n_1,0) \tag{2.59}$$

The corresponding output equation is given by

$$g(n_1,n_2) \;=\; [\widehat{\mathbf{C_1}} \;+\; \widehat{\mathbf{C_2}}[\mathbf{I} \;-\; \widehat{\mathbf{A_2}}]^{-1}]Q(n_1-1,0) \;+\; [\widehat{\mathbf{C_2}}[\mathbf{I} \;-\; \widehat{\mathbf{A_2}}]^{-1}\mathbf{B} \;+\; \mathbf{D}]f(n_1,0) \tag{2.60}$$

Let

$$\mathbf{A_h} \;=\; [\mathbf{I} \;-\; \widehat{\mathbf{A_2}}]^{-1}\,\widehat{\mathbf{A_1}}\;, \tag{2.61}$$

$$\mathbf{B_h} \;=\; [\mathbf{I} \;-\; \widehat{\mathbf{A_2}}]^{-1}\,\mathbf{B}\;, \tag{2.62}$$

$$\mathbf{C_h} \;=\; [\widehat{\mathbf{C_1}} \;+\; \widehat{\mathbf{C_2}}[\mathbf{I} \;-\; \widehat{\mathbf{A_2}}]^{-1}\;, \tag{2.63}$$

and

$$\mathbf{D_h} \;=\; [\widehat{\mathbf{C_2}}[\mathbf{I} \;-\; \widehat{\mathbf{A_2}}]^{-1}\mathbf{B} \;+\; \mathbf{D}] \tag{2.64}$$

Then, the state space representation for the first row can be written as

$$\begin{aligned} Q(n_1,0) \;&=\; \mathbf{A_h}Q(n_1-1,0) \;+\; \mathbf{B_h}f(n_1,0) \\ g(n_1,0) \;&=\; \mathbf{C_h}Q(n_1-1,0) \;+\; \mathbf{D_h}f(n_1,0) \end{aligned} \tag{2.65}$$

For our example, we have

$$\mathbf{A_h} \;=\; \begin{bmatrix} 0.44721360 & 1 & 0.44721360 & 0 & 0.44721360 & 0 & 0 & 0 \\ -0.23698230 & 0 & -0.23698230 & 0 & -0.23698230 & 0 & 0 & 0 \\ -0.16525767 & 0 & -0.16525767 & 1 & -0.16525767 & 0 & 0 & 0 \\ 0.08757145 & 0 & 0.08757145 & 0 & 0.08757145 & 0 & 0 & 0 \\ 0.08757145 & 0 & 0.08757145 & 0 & 0.08757145 & 1 & 0 & 0 \\ -0.0464048^{c} & 0 & -0.04640486 & 0 & -0.04640486 & 0 & 0 & 0 \\ 0.02102313 & 0 & 1.21023130 & 0 & 1.21023130 & 0 & 0 & 0 \\ -0.23698230 & 0 & -0.23698230 & 0 & 0.76301770 & 0 & 0 & 0 \end{bmatrix} \tag{2.66}$$

$$\mathbf{B_h} = \begin{bmatrix} 0.16167809 \\ 0.00222197 \\ 0.14248059 \\ 0.10029146 \\ 0.10029146 \\ 0.00347517 \\ 0.16390006 \\ 0.00222197 \end{bmatrix} \tag{2.67}$$

$$\mathbf{C_h} = [1.2102313 \quad 0 \quad 1.2102313 \quad 0 \quad 1.2102313 \quad 0 \quad 0 \quad 0] \tag{2.68}$$

For the first column, we have

$$Q(-1, n_2) = Q(0, n_2) \tag{2.69}$$

Using this assumption, we obtain

$$Q(0, n_2) = [\mathbf{I} - \widehat{\mathbf{A_1}}]^{-1} \widehat{\mathbf{A_2}} Q(0, n_2 - 1) + [\mathbf{I} - \widehat{\mathbf{A_1}}]^{-1} \mathbf{B} f(0, n_2) \tag{2.70}$$

The corresponding output equation is given by

$$g(0, n_2) = [\widehat{\mathbf{C_2}} + \widehat{\mathbf{C_1}}[\mathbf{I} - \widehat{\mathbf{A_1}}]^{-1}] Q(0, n_2 - 1) + [\widehat{\mathbf{C_1}}[\mathbf{I} - \widehat{\mathbf{A_1}}]^{-1}\mathbf{B} + \mathbf{D}] f(0, n_2) \tag{2.71}$$

Let

$$\mathbf{A_v} = [\mathbf{I} - \widehat{\mathbf{A_1}}]^{-1} \widehat{\mathbf{A_2}}\,, \tag{2.72}$$

$$\mathbf{B_v} = [\mathbf{I} - \widehat{\mathbf{A_1}}]^{-1} \mathbf{B}\,, \tag{2.73}$$

$$\mathbf{C_v} = [\widehat{\mathbf{C_2}} + \widehat{\mathbf{C_1}}[\mathbf{I} - \widehat{\mathbf{A_1}}]^{-1}\,, \tag{2.74}$$

and

$$\mathbf{D_v} = [\widehat{\mathbf{C_1}}[\mathbf{I} - \widehat{\mathbf{A_1}}]^{-1}\mathbf{B} + \mathbf{D}] \tag{2.75}$$

Then, the state space representation for the first column can be written as

$$\begin{aligned} Q(0, n_2) &= \mathbf{A_v} Q(0, n_2 - 1) + \mathbf{B_v} f(0, n_2) \\ g(0, n_2) &= \mathbf{C_v} Q(0, n_2 - 1) + \mathbf{D_v} f(0, n_2) \end{aligned} \tag{2.76}$$

For our example, we have

$$\mathbf{A_v} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0.21023129 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.23698230 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.07768622 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.08757145 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.04116659 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.04640486 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.36952738 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.19581571 & 0 \end{bmatrix} \tag{2.77}$$

$$\mathbf{B_v} = \begin{bmatrix} 0.16390006 \\ 0.00222197 \\ 0.24277204 \\ 0.10029146 \\ 0.13504272 \\ 0.03475127 \\ 0.40445013 \\ 0.13726469 \end{bmatrix} \tag{2.78}$$

$$\mathbf{C_v} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1.3695274 \quad 1] \tag{2.79}$$

Chapter 3

The Block Date Flow Architecture.

3.1 Introduction.

Two-dimensional digital filtering is one of the important applications of 2-D DLSI systems. We need a multiprocessor system to implement 2-D digital filters in real-time at rates appropriate for image display. We began our research on the real-time implementation of 2-D digital filters by exporing the design of a special purpose multiprocessor system for this purpose. We later explored the potential for increasing the programmability of our design to solve other problems. Thus, we derived the BDFA. We are currently exploring the mapping of other problems to the BDFA.

3.2 The BDFA Configuration

A BDFA system consists of three modules: an Input Control Module (ICM), a Processor Array (PA), and an Output Control Module (OCM) as shown in Figure 3.1.

3.2.1 Input Control Module

The ICM serves as a buffer between the host system (or an input/output device) and the processor array. It includes two FIFO buffers and it converts the input data stream into blocks of data. It maintains a direct input channel to each processor. Designated data blocks are sent to each processor through these channels without any interference from other processors. A control logic submodule provides each processor with control for data management and communication services. The block diagram of the ICM is shown in Figure 3.2.

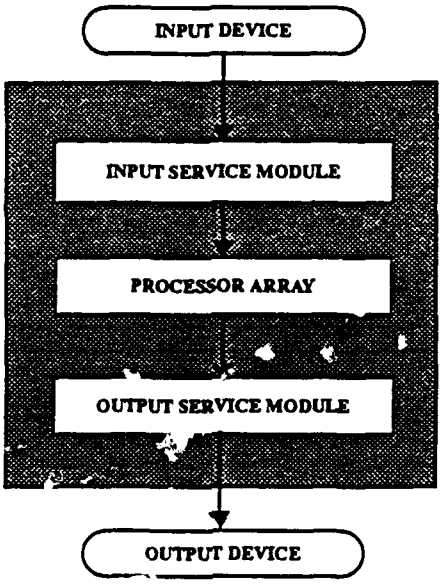

Figure 3.1: Block Diagram of the BDFA

3.2.2 The Processor Array

The PA contains enough processors to provide the computational power required for real-time signal processing and fast matrix operations. Since we limit the interprocessor communications to being local and in one direction, we can simply connect all the processors together to form a linear array. Each processor has a separate input channel and a separate output channel. The processors are divided into two processor groups: an odd number processor group and an even number processor group. Each processor group is directly connected to an input FIFO buffer and an output FIFO buffer. Therefore, each processor always uses the same input and output FIFO. Finally, FIFO buffers are used for interprocessor communications to minimize overhead due to addressing and routing. The block diagram of the processor array is shown in Figure 3.3.

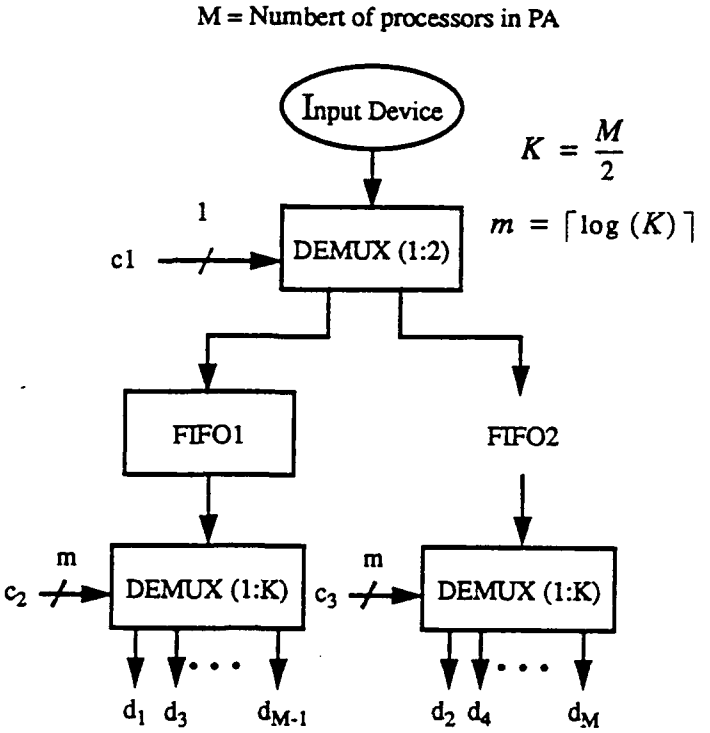

Figure 3.2: Input Control Module

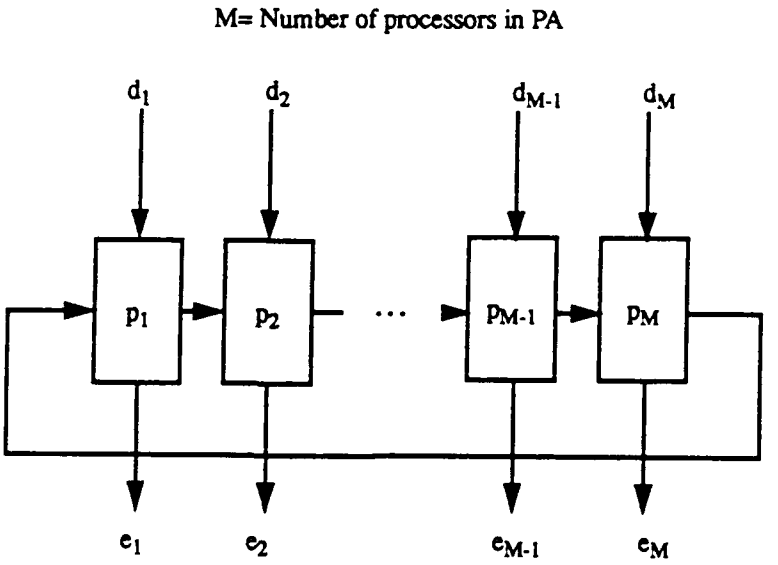

Figure 3.3: Processor Array

3.2.3 The Output Control Module

The OCM consists of a control logic submodule, a submodule for post-processing, and two output FIFO buffers. It collects processing results from each processing element and converts the blocks of data into a synchronized output data stream. It provides each processor with data management and communication services. The post-processing submodule also may implement different dynamic scaling algorithms for signal processing. It collects overflow information from each processor and adjusts the system gain based on this information. For example, the system gain factor may be fed back to the processor array at the end of each frame. A scale memory also can be used as a "look-up table" to scale the output for a particular output device. The post-processing submodule is very flexible and can contain different function modules for specific applications. The block diagram of an OCM is shown in Figure 3.4.

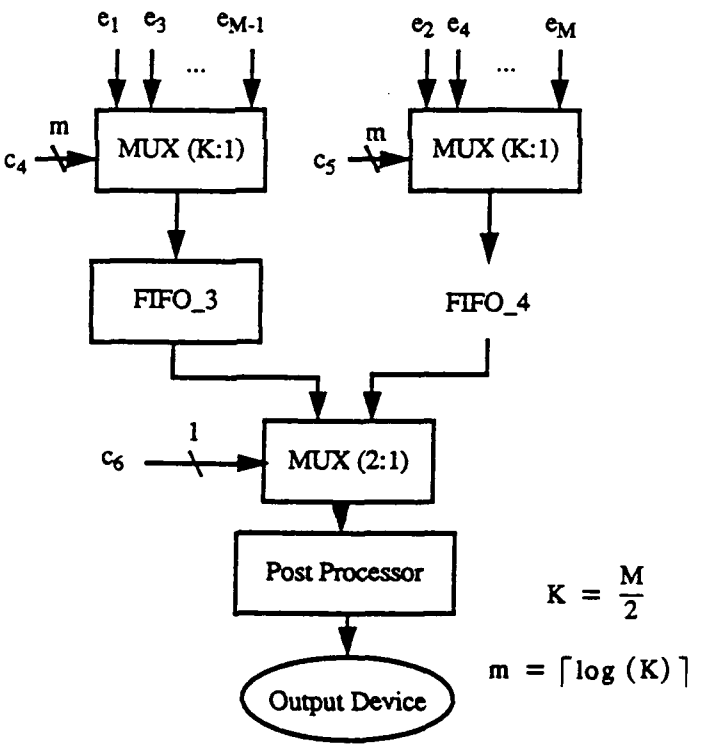

Figure 3.4: Output Control Module

3.3 Architectural Features of a BDFA

The architectural features of a BDFA are:

- block data processing and the block data flow paradigm,
- globally asynchronous and locally synchronous data transmission protocol,
- linear array topology and "skew" operations among processors,
- local data transmission in only one direction, and
- overlap of data movement and interprocessor communication with data computations.

Large scale tasks can be divided into smaller tasks using either an algorithm partitioning strategy or a data partitioning strategy. With an algorithm partitioning strategy, a complex algorithm is decomposed into a sequence of simple operations. Each simple operation or group of operations is assigned to a different processor. With the data partitioning strategy the whole image or matrix is divided into data blocks and each data block is assigned to a different processor. Each processor is capable of performing all the required functions for the assigned data blocks. The algorithm-partitioning strategy can simplify the structure of each processing element. However, the processors in an algorithm partitioned system cannot operate independently and they are subject to timing, sequencing and data dependency restrictions.

In the BDFA, we adopted the data partitioning strategy at the high level to build an alternative structure with independent processors. The more independent the processors are, the less time required to implement data communications protocols or to synchronize data movements. A structure with independent processors is also more flexible in coping with a variety of algorithms with different operational requirements.

The second advantage of the data partitioning strategy is the reduction of unnecessary data movement between the processors. Input data goes directly to the processor that will use it. The interprocessor communications are limited to passing necessary intermediate computational results. Output results go directly from each processor to an output device without any interference to or from other processors. Additionally, block data processing provides the opportunity for intermediate computational results to be used locally. A large reduction in interprocessor data communications can have a tremendous impact on the hardware implementation.

3.3.1 Block Data Flow

The BDFA implements the block data flow paradigm to achieve maximum parallelism at the processor level. Incontestably, we need many processing elements working together to increase computational power. The use of the Von Neumann computation model restricts the full utilization of all of the processing elements. The management of and the contention for the globally addressable memory necessary with the Von Neumann structure also limits effective parallelism.

The data flow model is different from the Von Neumann computation model. Data flow processors are stored-program computers. If sufficient resources are provided, the system can exploit all concurrency present in the program. This approach can be naturally extended to an arbitrary number of processors[14]. This concept also reduces the data dependency, control dependency, and resource dependency among processors. However, it is difficult to manage the data flow model for a multiprocessor system[15]. We implemented the data flow paradigm at the processor array level with a large data-block-grain and limited our array to being linear. With these restrictions, we have successfully implemented the data flow paradigm for the BDFA.

When a processor in a BDFA system has received its assigned data block and all of its intermediate data, then the processor is able to perform its designated functions independently. When data blocks and the necessary intermediate data are available for all processors, then all processors are able to perform their designated functions on their own data blocks independently.

The use of the block data flow paradigm also helps us to reduce data storage requirements. In a BDFA system, the input data blocks flow into the system and the output data blocks flow out of the system. There is no need to store the whole frame of the image or all the entries of a matrix into a BDFA system.

3.3.2 Data Transmission Protocol

Data transmission protocols may be categorized into synchronous data transmission protocols and asynchronous data transmission protocols. The synchronous data transmission protocol is fast and simple and there is no handshaking overhead. However, the synchronous data transmission protocol places a timing restriction on the system design. In particular, this can be a problem for large-scale systems. The asynchronous data transmission protocol does not have this timing restriction. However, there is a considerable amount of overhead associated with the asynchronous data transmission protocol. The BDFA system uses a globally asynchronous data transmission protocol with a large data grain and a locally synchronous data transmission protocol with small data elements to:

- avoid the globally synchronous transmission problem,
- minimize overhead due to asynchronous handshaking signals,
- reduce communications control hardware, and
- minimize data communications overhead.

3.3.3 Linear Array Topology and Skew-Operations

Since the BDFA uses the data partitioning strategy, the interprocessor communications have been limited to only passing intermediate computational results. We restrict the interprocessor communications to be local and only in one-direction to make the implementation of the data flow paradigm feasible. This means we can simply connect the processors together to form a linear array. The linear array topology is simple and has efficient channel utilization[16]. Furthermore, the linear array topology allows us to skew the operations among the processors[17]. Allowing the operations to be skewed among processors plays an important role in balancing the system input/output bandwidth and the computational intensity. It also helps to reduce the storage requirements of the interprocessor communication buffers.

3.3.4 Data Communications

A BDFA system overlaps the input/output data movements and the interprocessor communication with data computations. This is possible because the ICM takes care of routing the input data block to the appropriate processor as soon as it is available and the OCM always provides an output FIFO whenever a processor needs to output a block of processing results. Therefore, the processors are able to devote almost 100% of their time to computations. Consequently, the system achieves high throughput and high efficiency.

In addition, a BDFA system has all of the advantageous features of a systolic array or wavefront array. This includes such features as modularity, regularity, local interconnection, highly pipelined multiprocessing, highly parallel processing at the array level, and a balance of external I/O and computational intensity. The BDFA system is also able to use a systolic array or a wavefront array for its processing elements.

3.3.5 The BDFA Mapping Criteria

We established three criteria for mapping algorithms to a BDFA. These three criteria are:

- the algorithm must be data partitionable,
- data communications must be local and in only one direction, and
- the computational load must be balanced among the processors.

The requirement for the algorithm to be data partitionable lays the foundation for block data processing. The requirement for local uni-directional interprocessor communication makes it easy to implement the block data flow paradigm. Basically, any algorithm which conforms to these two criteria can be implemented on a BDFA. The third criterion, the computational load balance, sometimes is hard to achieve because of the variety of computational requirements of different algorithms. However this criterion only affects the system's hardware efficiency. In some applications, the hardware efficiency is not very critical and the system throughput is of most concern. Thus, if an algorithm meets the first two BDFA mapping criteria but not the third one, it still can be implemented on a BDFA with high throughput. In addition, these criteria are not very restrictive and many algorithms can meet this criteria or can be adapted to this to meet this criteria.

We have been able to map the following algorithms onto a BDFA:

- 2-D digital IIR filter[8],
- 2-D digital FIR filter[8],
- orthogonal transformation of a dense matrix using Givens rotations[7],
- updating and down-dating for the least square problem based on an inverse QR decomposition[7],
- lower-upper (LU) decomposition of a dense matrix[7], and
- 2-D discrete cosine transform[8].

3.3.6 Performance Evaluation

The BDFA was developed as a part of our efforts to implement 2-D IIR filters in real-time [12],[8],[18]. As a part of this effort, we designed a special purpose node processor [18] and we developed a multiprocessor system which uses this processor to implement 2-D IIR filters in real-time [7]. We refer to the special purpose node processor as a 2-D DSP. In this section, we summarize our simulation results on the performance evaluation of this multiprocessor system as an example of the expected performance of a BDFA system.

Table 3.1: Performance of Systems with a Different Number of Processors

N	initialization	latency	wait	throughput
10	3492	1418	4	0.9696
6	2129	1418	505	0.5940
2	765	1418	1028	0.1988

Table 3.1 and table 3.2 give the functional level simulation results for the 2-D IIR digital filter BDFA system. In these tables, "initialization" refers to the number of cycles required to initialize the system and load filter coefficients, "latency" refers to the interval in cycles between the time a processor receives its assigned data block and the time it begins transferring its output to the OCM, and "wait" refers to the number of cycles between output data blocks. We consider the system to be performing in real-time when there is always a processor ready to receive an input block when it is ready.

Table 3.1 shows the performance of a second order system with a different number of 2-D DSPs. The size of the sample image is 128 × 128 pixels. This table reveals that the BDFA system can perform 2-D IIR digital filtering in real-time and that it essentially achieves a linear speed-up rate. The relative system throughput (the ratio of output pixels over system cycles needed) for a ten-processor system is very close to 1 (0.9696). The relative system throughput of a six-processor system is very close to 0.6 (0.5940). The relative system throughput of a two-processor system is very close to 0.2 (0.1988). The system throughput of the ten-processor system is about 5 times as high as the system throughput of the two-processor system and 1.6 times as high as the throughput for the six-processor system. This means the system throughput is proportionally increased with the increase of the number of processors until real-time processing is achieved. Thus, the BDFA system almost achieves a linear speed-up rate.

Table 3.2: Ten-Processor System's Performance on Images with Different Sizes

size	initialization	latency	wait	throughput
512 × 512	3492	5642	4	0.9922
256 × 256	3492	2816	4	0.9846
128 × 128	3492	1418	4	0.9696
64 × 64	3492	714	4	0.9411
16 × 16	3492	186	4	0.8000

Table 3.2 shows a ten-processor system processing images with different sizes. The system achieves its maximum throughput when it processes the image with the largest possible data block size. The system processes all the images in real-time. This indicates the number of processors needed for real-time processing is independent of the processed image

size.

Commercial DSPs, such as the Motorola DSP56000, and general-purpose processors, such as the Intel 80486, can be used as processing elements in a BDFA system. The system's throughput will increase proportionally with the number of processors in the system due to the characteristics of the BDFA. Table 3.3 shows the number of cycles needed to compute the output of a pixel element using different processors in a BDFA system.

Table 3.3: The Number of Cycles for Different Processors

order	2-D DSP	DSP56000	80486
2	10	36	273
4	26	100	785
8	82	324	2577

3.4 Conclusions

High system throughput and high system efficiency are the key requirements for many real-time signal processing and fast matrix operation applications. The BDFA provides an alternative multiprocessor system architecture for high throughput and high efficiency.

Chapter 4

Appendix A - List of Publications.

This appendix gives a list of publications during the contract period of 1 November 1990 to 31 March 1992.

4.1 Publications Currently Being Reviewed.

1. Jae-Gil Jeong and Winser E. Alexander, "Implementation of 2-D digital filters with block mode processing", submitted to *IEEE Transactions on Circuits and Systems for Video Technology.*

2. William Edmonson and Winser E. Alexander, "Boundary value transient suppression for N-D digital systems", submitted to *IEEE Transactions on Signal Processing.*

4.2 Publications in Refereed Journals and Conference Proceedings.

1. Hongyu Xu and Winser E. Alexander, "Parallel QR Factorization on a Block Data Flow Architecture", *Proceedings of the Twenty Fourth Southeastern Symposium on System Theory*, pp. 332-336, 1992.

2. Kwang-Hoon Sohn, Winser E. Alexander and Jung H. Kim, "Constrained Regularization Approach to Curvature Estimation", *Proceedings of the Twenty Fourth Southeastern Symposium on System Theory*, pp. 235-239, 1992.

3. Alvernon Walker, Parag K. Lala and Winser E. Alexander, "Fault diagnosis in analog circuits using element modulation", *IEEE Design and Test of Computers*, pp 19-29,

March, 1992.

4. Alvernon Walker, Parag K. Lala and Winser E. Alexander, "Analogue fault diagnosis of active networks via bias modulation", *Electronics Letters*, vol. 27, no. 24, pp. 2279-2281, 1991.

5. Kwanghoon Sohn, Winser E. Alexander, Jung H. Kim, Yonghoon Kim, Sung H. Yoon, Eui H. Park and C. A. Ntuen, "Optimal boundary smoothing for curvature estimation", *Proceedings of the 25th Asilomar Conference on Signals, Systems and Computers*, November, 1991.

6. Alvernon Walker, Parag K. Lala and Winser E. Alexander, "A Technique for analog fault diagnosis using element modulation" *Proceedings of the 25th Asilomar Conference on Signals, Systems and Computers*, November, 1991.

7. Sung H. Yoon, Jung H. Kim, Eui H. Park, Celestine A. Ntuen, Kwang H. Sohn, Arne A. Nilsson, Young H. Kim and Winser E. Alexander, "Significant point detection and boundary representation with lines and circular arcs", *Proceedings of The Fourth International Conference on Industrial & Engineering Applications*, June, 1991.

8. Hatice Ozurk and Winser E. Alexander, "Sampling error analysis for perfect reconstruction transmultiplexers", *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, May, 1991.

9. Jae Gil Jeong and Winser E. Alexander, "The efficient real-time spatial domain 2-D IIR and FIR filter implementation", *Proceedings of the 1991 Southeastern Symposium on System Theory*, pp. 394-398, 1991.

10. Jae Gil Jeong and Winser E. Alexander, "A real-time implementation of the efficient 2-D discrete cosine transform", *Proceedings of Southeastcon*, April, 1991.

11. Kwanghoon Sohn, Winser E. Alexander, Arne A. Nilsson, Jung H. Kim, Eui H. Park and Sung H. Yoon, "Boundary representation with lines and circular arcs using boundary split-and-merge method", *Proceedings of Southeastcon*, April, 1991.

4.3 Patents During the Contract Period.

1. Winser E. Alexander, Hongyu Xu and Jae-Gil Jeong, *A Block Data Flow Architecture for Digital Signal Processing*, Patent application filed on February 18, 1992.

Bibliography

[1] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984.

[2] J. Speiser and H. Whitehouse, "Architectures for real time matrix operations," in *Proceedings of 1980 Government Microcircuit Application Conference*, pp. 19–21, 1980.

[3] D. R. Kincard, D. M. Young, and T. C. Oppe, *ITPACK User's Guide.* Houston, Texas: Center for Numerical Analysis, University of Texas at Austin, 1989.

[4] SIAM, Philadelphia, *LINPACK User's Guide*, 1989.

[5] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing.* New York: McGraw Hill, 1978.

[6] H. T. Kung, "Why systolic architectures?," *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[7] H. Xu, *BDFA - A Block Data Flow Architecture for Real-Time Signal Processing and Matrix Operations.* PhD thesis, North Carolina State University, 1991.

[8] J. G. Jeong, *High Performance Multiprocessor Architecture for Digital Signal Processing.* PhD thesis, North Carolina State University, 1991.

[9] R. Roesser, "A discrete state space model for linear image processing," *IEEE Trans. on Automatic Control*, vol. AC-20, pp. 1–10, 1975.

[10] E. Fornasini and G. Marchesini, "State space realization theory for two dimensional filters," *IEEE Trans. on Automatic Control*, vol. AC-21, pp. 484–492, 1976.

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

[12] J. H. Kim and W. E. Alexander, "A multiprocessor architecture for two-dimensional digital filters," *IEEE Trans.Comput.*, vol. C-36, pp. 876–884, 1987.

[13] D. D. Givone and R. P. Roesser, "Multidimensional linear iterative circuits - general properties," *IEEE Trans. Comput.*, vol. C-27, no. 10, pp. 1067–1073, 1972.

[14] T. Agrewala and Arvind, "Data flow systems," *IEEE Computer*, vol. C-36, pp. 10–13, 1982.

[15] D. D. Gajski, D. A. Padua, D. J. Kuck, and R. H. Kuhn, "A second opinion on data flow machines and languages," *Computer*, vol. 15, pp. 58–69, 1982.

[16] L. N. Bhuyan, D. Ghosal, and Q. Yang, "Approximate analysis of single and multiple ring networks," *IEEE Trans.Comput.*, vol. C-38, pp. 1027–1031, 1989.

[17] J. Decaluwe and J. M. Rabacy, "Interprocessor communication in synchronous multiprocessor digital signal processing chips," *IEEE Transactions on Acoustic, Speech, and Signal Processing*, vol. Vol-38, pp. 1816–1827, December 1989.

[18] S. M. Park, W. E. Alexander, J. H. Kim, W. E. Batchelor, and W. T. Krakow, "A novel VLSI architecture for the real-time implementation of 2-D signal processing systems," *Proceedings of the IEEE Int. Conf. Comp. Design: VLSI in Computers and Processors*, October 1988.