EVALUATION OF A SINGLE VLSI CHIP ALGORITHM FOR TRIANGULATING LARGE BAND FORM MATRICES

By

Wen Chang Hsu

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering and
Systems Science

1982

ABSTRACT

EVALUATION OF A SINGLE VLSI CHIP ALGORITHM
FOR
TRIANGULATING LARGE BAND FORM MATRICES

By

Wen Chang Hsu

One of the efficient procedures to solve a large, sparse system of linear equations, $\underline{A} \cdot \underline{x} = \underline{b}$, as encountered in a variety of common engineering problems, is to go through fast, band matrix triangulation utilizing concurrent Gaussian elimination. This algorithm may be implemented using systolic computing cells configured into a simple, regularly connected processing array. With the most recent advances in microelectronics technology, it will soon be possible to fabricate this computing array on a single chip.

A computing structure, utilizing both parallel and pipeline concepts to triangulate an augmented band form coefficient matrix $\left\{ \underline{A} \;\vdots\; \underline{b} \right\}$ in O(N) time steps, where N is the order of the matrix, is presented. This new design utilizes only $O(B^2)$ processing cells, where B is the matrix bandwidth. This compares favorably to previous algorithms which require $O(N^2)$ cells. Hardware arithmetic algorithms of MAC and DC, the required processing cells of the computing structure, are designed and evaluated.

Three I/O circuits are presented and compared with respect to area-time trade-offs. An optimal input strategy, the data controlled algorithm and optimal output strategy, the SCS algorithm are chosen.

A transistor level circuit simulation, based on parameters of word size, matrix bandwidth and minimum lithographic linewidth, provided parameters of area geometry and delay time estimations for the I/O and computing structures. The ultimate goal of this simulation was to provide two important comparisons, total propagation delay and chip size versus matrix bandwidth. For a given matrix bandwidth, the optimal chip size and throughput speed based on current and projected lithographic linewidths were evaluated. Bottlenecking of the I/O operands was not observed for word lengths $\geq$ 16 bits and small matrix bandwidths.

These results, in turn, lead to an assessment of the feasibility and advantages of this class of special purpose VLSI computing structures.

To my parents

Mr. and Mrs. Chiung-Yung Hsu

ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation to his major advisor, Dr. Michael A. Shanblatt, for his guidance and encouragement in the course of this research.

He also wishes to thank the committee members, Dr. P. D. Fisher, Dr. R. G. Reynolds, Dr. E. D. Goodman and Dr. S. R. Crouch for giving the valuable suggestions and comments in this research.

Finally, the author owes a special thank you to his wife, Huey-Fen, for her confidence and emotional support that only a wife can give.

Work reported here was supported in part by NSF under Grant ECS-8106675.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

High speed VLSI (Very Large Scale Integration) computing arrays, based on concurrent Gaussian elimination, for triangulating the linear equation system $\underline{A} \cdot \underline{x} = \underline{b}$ have become a topic of recent interest. By combining parallel and pipeline concepts, these "systolic arrays" [1] can rapidly execute the numerous inner-product step operations which serial computers must tediously operate upon sequentially. It is projected that these high performance algorithms can be implemented directly using low cost, high speed VLSI circuit technology either on a simple chip, or perhaps in a modular fashion on a few chips. These devices can then be attached to either dedicated or general purpose host computers as plug-in components capable of producing fast solution vectors to large scale systems of equations. This concept is similar to the currently available attached peripheral array processors and to those devices which perform dedicated fast Fourier transforms exclusively in hardware.

The work described herein advances this topic by the design of a computing algorithm to triangulate systems of arbitrary size which have been permuted, a priori, to minimum band form. The algorithm is applicable to any structurally symmetric, diagonally dominant coefficient matrix. This permits use of the algorithm on many naturally occuring band form matrices and, more importantly, on the highly

sparse, diagonally dominant matrices arising from a multitude of engineering and scientific problem formulations. The salient feature of the algorithm is that the key restriction limiting network size is on the reduced matrix bandwidth, not on the original network dimension. Moreover, several good heuristic bandwidth reduction algorithms, based on graph theoretic concepts, already exist and are quite easy to implement [2, 3].

It is assumed that the permutation of $\underline{A}$ to minimum band form involves full row and column pivoting so as to ensure the retention of the necessary property of diagonal dominance.

The processing elements of the computing structure were originally designed to operate with fixed-point binary numbers. But the nature of this structure also allows these PE's to work with the mantissas of floating-point input operands provided that all exponent values have been equalized and stored in the host processor. More details of this idea will be presented in Chapter III.

In a single chip system, operand input and output may present a bottleneck due to some very practical problems of packaging considerations. Therefore, high speed serial-to-parallel input and parallel-to-serial output circuits must be included in the design so as to minimize any potential bottleneck and thus retain optimal system throughput. Three different I/O strategies are presented in Chapter IV. Comparisons are made in terms of area and propagation delay requirements.

The final quantifications of area geometry and propagation delay for the total chip, composed of the computing structure with the optimal input/output strategies, are examined via a chip model simulation in terms of matrix bandwidth, word size and lithographic linewidth.

These simulation results are intended to provide a realistic estimation of the feasibility, with respect to problem size and throughput, of this type of systolic array numerical algorithm fabricated on a single VLSI chip. This, in turn, will shed light on how these ideas may benefit the numerous scientific and engineering problems for which enhanced throughput in the rapid solution of large scale simultaneous equations is greatly desired.

1.1 Statement of Problem

VLSI based dedicated computing structures offer great potential for optimizing problem throughput and hardware investment. Yet many aspects of the design and implementation of these ideas still remain incomplete or underdeveloped. Such problems include the inner details of various processing elements, packaging constraints and input/output considerations.

The goal of this work is to develop and assess aspects of a special purpose VLSI computing architecture to triangulate large scale, band form, linear equation systems. Specific areas to be examined are as follows:

1. The design and evaluation of a computing array

structure for this problem type is presented. The global dimensions of this structure are based on a function of coefficient matrix bandwidth and on the assumption of an implementation of the row-ordered permutation of Gaussian elimination resulting in a strict upper triangulation of the coefficient matrix. A coefficient input and output patterns are presented.

2. Individual processing elements are designed and evaluated. This includes exploration of various fixed-point multiplier, adder and divider algorithms. Selection of the best algorithms are based on comparisons of speed, area, modularity and local communication properties.
3. Possible input demultiplexer and output multiplexer circuit designs are evaluated. Parameters which are examined include area geometry, propagation delay and a consideration of the effects of the fundamental limitation of pinouts per chip. I/O designs are also evaluated with respect to potential bottlenecking of operands which can severely degrade the overall system throughput.
4. Realistic estimation of the total propagation delay versus matrix bandwidth and chip size versus matrix bandwidth of the entire chip is made by incorporating the optimal computing structure and I/O circuit designs.

The results of this work, particularly tasks 1 and 4, enable a more comprehensive assessment of VLSI's potential for reducing solution time of large linear equation system problems.

1.2 Approach

In the past few years the advantages of systolic, hex-connected array structures, realizable with VLSI technology, were introduced [12, 13]. These structures have been shown to be applicable to many matrix computation problems including Gaussian elimination [16, 17].

The first step of this work is the development of a computing array to perform a complete factorization (not merely L-U decomposition) of an N x (N + 1) matrix ($\underline{A}$ augmented by $\underline{b}$) in minimum unit time. In other words, the objective is to create a set of possible input patterns from the nonzero entries of a matrix, most suitably in band form, and implement a computing structure in a simple hexagonal array. The plan is that these two structures, the input pattern and the computing array, may be manipulated back and forth until their subsequent output pattern fits the required upper triangular pattern. The output pattern should be that which is most conducive to rapid back-substitution.

The second step entails study of the input/output problems for the main body of the chip - the developed computing array. The number of pinouts per chip of current and projected VLSI technology give primary ideas on how control

signals, fan-in demultiplexer and fan-out multiplexer circuits must be developed. In order to avoid a potentially serious I/O bottleneck problem, the major point of these input and output circuit designs is to provide high speed channels for operand routing. Also, design regularity and low power requirements, necessary for good VLSI implementation, are considered.

The third step develops some of the inner details of the processing element structures. In order to fully assess the area geometry and propagation delay of these processing elements, schematic logic circuit diagrams of various arithmetic algorithms are studied. For example, the method for designing an MAC (Multiply-Add Cell) encompasses the following:

1. Review existing LSI multiplication algorithms and evaluate the optimal hardware implementation in terms of design complexity, local connection properties, modularity, speed and area. Also, study a possible attached high-speed full adder using identical procedures.
2. Develop a network realization of the chosen optimal fixed-point arithmetic algorithm for a complete MAC. This step will lead to quantification of the necessary performance measurements of area geometry and propagation delay.
3. Develop a hand layout of the fundamental building block modules, such as a full-adder, in terms of

scalable minimum lithographic linewidth, λ. Using these design layouts, quantify the A_C (area per cell) in terms of λ, and T_C (propagation delay per cell). T_C will be determined in terms of typical basic inverter discharge time, also a function of λ, and any non-negligible communication path delays.

The fourth step is the development and evaluation of the required building block modules for several possible I/O circuit strategies, then quantify area geometries and propagation delays of these modules using the methodology of step 3.

The fifth and final step is a Fortran-coded simulation to provide two important comparisons, total propagation delay versus matrix bandwidth and chip size versus matrix bandwidth. These comparisons are determined by varying parameters of minimum lithgraphic linewidth, word size and matrix bandwidth. The results, combined with predictions of future trends in VLSI technology, will help evaluate, and hopefully promote, the feasibility and advantages of special purpose VLSI computing structures as linear equation solvers.

1.3 Contributions

This section provides readers with an additional understanding of the original research which has been accomplished in this dissertation work.

1. The successful design of a new VLSI systolic array algorithm based on concurrent Gaussian elimination is presented. This algorithm is unique in that it triangulates a band form matrix $\underline{A}$ augmented by vector $\underline{b}$ (in other words, $\{\underline{A} \mid \underline{b}\}$). The array requires O(N) unit time and $O(B^2)$ processing elements, (where N and B are the dimension and half bandwidth of $\underline{A}$, respectively). The new algorithm, requiring $O(B^2)$ processing elements represents a dramatic improvement over previously published algorithms of this form which have required $O(N^2)$ PE's. This is a tremendous saving, particularly when $B << N$.
2. The inner details of processing element designs are developed and assessed. This entails evaluating and configuring the required MAC's (Multiply and Add Cell) and DC's (Divider Cell). The best designs were obtained subject to criteria of design regularity, design complexity, chip area and propagation delay requirements.
3. The best input strategy, the data controlled algorithm and the best output strategy, the shift-register control sequence (SCS) algorithm, were developed and evaluated. These strategies helped tremendously in minimizing a potential I/O bottleneck problem which was anticipated in systolic array structures of this type.
4. Determinations of the entire chip area geometry and total propagation delay were obtained using a

comprehensive chip simulation. Results were based on various word size, minimum lithographic linewidth and matrix bandwidth parameters. The simulation results have indicated that this type of special purpose VLSI chip indeed provides rapid triangulation of large scale, band form linear equation systems.

CHAPTER II

BACKGROUND

2.1 Array Processors and the Vector Nature of Gaussian Elimination

Array processors are generally realized as one (vector) or two (matrix) dimensions of processing elements which allow data operands to be processed in parallel. In the literature [37], array processors have been defined as those structures for which the information unit (data string) is an array of one or two dimensions. This would define processors in which both serial and parallel execution would be implemented in a pipelined or even a parallel pipelined fashion. The definition of array processors as used here are those systems where the processing elements have both parallel and pipelined structures. There are two basic classifications of array processors. First, the stand alone type, such as the CRAY-1 or the CDC STAR-100, and second, the scientific processors like the AP-120 B/FPS-164 family or the MAP-300. The latter classification has been referred to as a special algorithm processor [37]. Array architectures are most advantageous for problem types which are vector or array oriented with highly repetitive arithmetic operations. The typical operations include vector addition and subtraction, matrix multiplication and convolution, and many advanced functions like fast Fourier transform. Applications of array processors have been proven to be

cost-effective in areas such satellite image processing [4], power system network computation [5], reservoir simulation [6], and pattern recognition [7].

For a class of large scale, linear equation systems, $\underline{A} \cdot \underline{x} = \underline{b}$, the method of reducing solution time via array processors is to apply a vectorized form of Gaussian elimination optimizing the use of these processors' parallel and pipelined structures in the triangulation procedure of the coefficient matrix.

For example, consider an N x N matrix. A possible FORTRAN coded program segment to perform rwo-order Gaussian elimination is

```
      DO  1   K = 1, N
      DO  1   J = 1, K
      DO  1   I = J + 1, N
      IF (K.EQ.J) GO TO 2
      A(K,I) = A(K,I) - A(K,J) * A(J,I)
      GO TO 1
2     A(K,I) = A(K,I)/A(J,J)
1     CONTINUE
```

The best way to expose the vector nature of this row-ordered elimination is to inspect the inner loop, on index I, of this Fortran program. Neglecting, at first, the conditional branch step for the diagonal divide operation, the program can be rewritten as,

```
      DO  1   K = 1, N
      DO  1   J = 1, K
      DO  1   I = J + 1, N
1     A(K,I) = A(K,I) - A(K,J) * A(J,I)
```

The DO loop on index I, from J+1 to N, is again the inner loop. For each J in the second loop, from 1 to K, the inner loop can be completed in two vector operations. The vectors and scalar operands required at this step are illustrated in Figure 2.1. In this example, K = 4, J = 2 and element A(4,2) is to be eliminated from the remainder of row 4.

The two vector operations required to eliminate A(4,2) are as follows. First, scalar-vector multiplication, that is, scalar element A(4,2) multiplies vector 1. Second, vector-vector subtraction, that is, vector 2 is subtracted from the updated vector 1. These vector functions are commonly available in most array processors' program libraries [8].

To illustrate the advantages of the array processors' melding of parallel and pipelined structures, assume a segmented multiplier where the operation time of the longest segment is T_{op}. Also, assume that both a pipelined and serial processor use the same segmented multiplier, but, the serial processor does not use the pipelined capability. Then, neglecting the set-up time, it takes T_{op} time for each multiply using the pipelined structure and $n \cdot T_{op}$ time, where

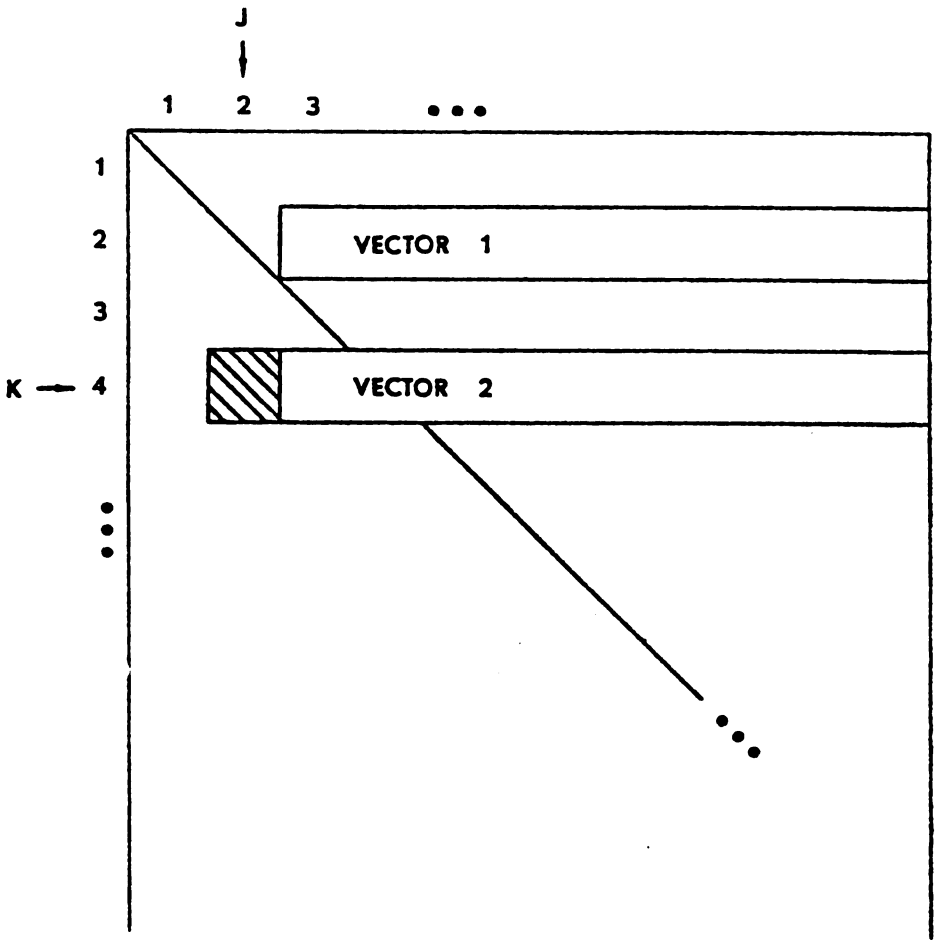

Figure 2.1 Vector nature of a full matrix row-order elimination step [8].

n is the number of segments, for the serial processor to perform its multiply.

If we consider a vector of length N, the serial computer requires $N \cdot n$ T_{op} time to complete the entire vector multiplication; whereas the segmented architecture can perform this multiplication in $N \cdot T_{op}$ time, neglecting set-up time. Under this simple assumption, the pipelined processor is potentially n times faster than the serial version. Therefore, array processors have computational advantages over serial computers on vector oriented problems. One fundamental drawback, however, is the optimal utilization of array processors' capabilities. A programmer must be able to detect optimal parallelism and pipelining of mathematical forms, then write efficient code for the array processor to achieve optimal and cost-effective results.

2.2 VLSI Architectures and Systolic Algorithms

Progress in microelectronics has been extremely rapid in the past few years. The ability to fabricate 10^7 to 10^8 transistors on a single chip will be possible by the late 1980's [9, 10]. Array processors, as mentioned, are in many ways still general purpose machines (i.e., programable). With VLSI technology, the design of truly special purpose machines will become a reality.

VLSI implementations will improve cost, speed and size of electronic components. Current interest lies in designing high-performance parallel algorithms that can be

implemented directly with low cost, high speed VLSI hardware. These devices are designed to be attached to a general purpose host computer as a plug-in component, capable of performing high speed, cost-effective computation.

A direct VLSI implementation of any computational algorithm should have the following properties [11]:

A. Regularity -

The algorithm must be implemented by only a few types of processing elements so as to reduce the design complexity and design cost.

B. Local Communication -

The processing elements must connect only to their nearest neighbors; global communications are to be avoided at all cost.

C. Parallelism and Pipelining -

The algorithm should ideally use both parallel and pipelined processing concepts.

A "systolic algorithm", which employs both parallel and pipelined processing concepts, is one which can be implemented by a network of simple, regular processors that compute and transmit data synchronously [1]. These processors, with planar external connections, can be configured into various geometric structures utilizing different frameworks [12]. Kung first presented a special purpose array structure built of standard inner product step processors and one division function processor to carry out the L-U decomposition algorithm on a full matrix of arbitrary size

[13]. This algorithm is well suited for a VLSI chip implementation because of its regular processing cell structure, local communication paths and efficient parallelism and pipelining.

Kung provided researchers with an innovative idea and encouraged others to develop different arithmetic algorithms using VLSI systolic function elements [12]. In his work, however, he neglects mention of any practical details, such as I/O considerations and processor cell designs. These issues will be considered in later chapters of this thesis.

Recently, several researchers have advanced the notion that these systolic array architectures can be configured to other highly concurrent numerical algorithms such as matrix multiplication, matrix inversion and recursive digital filtering [1, 9, 13, 14, 15]. Especially, Hwang and Cheng have designed a VLSI architecture to perform L-U decomposition of an entire linear system of equations $\left\{ \underline{A} \mid \underline{b} \right\}$, advancing Kung's design of merely triangulating $\underline{A}$. The upper triangular portion of the whole system's L-U decomposition is analogous to the forward elimination algorithm used in dense matrix linear equation systems [16, 17]. The required time slots and processing elements of this previously published algorithm, when used for triangulating band form linear equation problems, are $O(N)$ and $O(N^2)$, respectively.

2.3 Design Methodology of VLSI Systems

A systematic method is essential in designing this kind of complex system. The design problems can best be tackled when decomposed into several subproblems with simple and clear interrelationships.

VLSI chip design may involve the layout of as many as 2×10^5 gates on a chip [18]. Due to this complexity it is crucial to utilize a supporting system design methodology. Three basic design concepts - hierarchy, regularity and testability must be followed to ease VLSI design complexity. A brief description of each term is given below:

1. Design hierarchy -

   VLSI system designers are forced to use both top-down analysis and bottom-up synthesis procedures [11]. For practical VLSI system design, it is almost impossible to consider global data and control flow, circuit design and transistor characteristics all at once. Consequently, the top-down design levels are partitioned into five levels as follows:

   A. Algorithm level -

      The initial step involves exporing possible parallelism and pipelining of the given problem. Then, the algorithms' performance, i.e. chip area and time parameters, may be estimated as functions of problem parameters; for example, matrix dimension.

      Finally, the optimal algorithm is evaluated and

chosen in terms of area-time trade-off, design complexity and practical design constraints.

B. Block level -

At this step, implementation of the algorithm via functional blocks linked by data and control flow begins. This level of design combines circuit functions and relative circuit positions as well as signal inputs and outputs of different blocks.

C. Gate level -

Configuration of the function blocks with logic gates (NAND, NOR, INVERTER, etc.) and memory elements (flip-flops, delay lines, etc.) is performed at this stage. Additionally at this point, a fault diagnosis or fault testing procedure is used to detect the proper function of the logic circuits. These results may then be used to compare with future field testing results. In other words, a set of benchmark parameters are formulated.

D. Transistor level -

Next, gate level circuits are transferred to the primary transistor level circuits. The area geometry ration of pull-up transistor to pull-down transistor of the logic gates, according to various gate categories and gate couplings, are calculated.

E. Physical layout level -

The complexity of a VLSI system obviously makes hand layout too cumbersome to even consider.

Modern CAD (Computer-Aided Design) layout tools, such as an interactive layout system, enable the designer to create and edit element layout patterns directly on a CRT screen. The final pattern, called a design file, can be represented in an intermediate form; for example, the CIF (Caltech Intermediate Form) [9]. Then, the intermediate form file is transferred to a pattern generation file which in turn is used in the mask making process.

2. Design regularity -

The purpose of incorporating design regularity is to minimize the design time. In bottom-up synthesis, if only a few primitive cells have been used for creating function blocks, then the design time will decrease tremendously. Also, the area geometry of the interconnection wires can be reduced by manipulating possible processing element configurations.

3. Design testability -

In the past, SSI and MSI logic designers were trained to design with the fewest gates. In this simple situation AC design parameters such as rise time, fall time and circuit delay could easily be tested. The inherent complexity of a VLSI chip, however, now makes the testability consideration much more important. The task is is even more complex since, with VLSI, it is impossible to test every circuit. Consequently, circuit or logic

simulation is required to create a fault-free model, which forms a benchmark by which actual results may be judged. An actual circuit is then exercised by means of a set of stimuli called an input vector. Then, the result or output vector is measured and compared with the result of the logic simulation.

Eichelberger has suggested two new concepts of logical structure design conducive to VLSI design and testability [19]. First, one must structure the design so that the subsystem operation time is calculated by the output stabilization time due to the changes of input states. For synchronous logic, the output response time is the obvious limiting factor in determining minimum clock cycle time. Second, one should design the internal storage elements as shift-register type latches, which greatly simplifies the testing procedure.

These two concepts can be especially beneficial for testing high speed pipelined structures. Data flow is controlled by a two-phase nonoverlapping clock which controls the latches; one phase for pumping data into latch, the other for refreshing or pumping out data. The clock cycle time is determined by the sum of the steady-state delay of the subsystem and latch. The advantages of this type of VLSI testability are two-fold. First, the internal storage elements (latches) provide the input/output ports for tester's probes and thus, a convenient way to at least

isolate faulty circuits. Second, since data flow is synchronously controlled, the clock rate can be slowed down during the test.

CHAPTER III

COMPUTING STRUCTURE DESIGN

3.1 Band Form Algorithm

Central to the development of a systolic array for band matrix triangulation are some definitions of the type of applicable band forms and the nature of Gaussian elimination for such matrices. The algorithm is most analogous to the row-ordered permutation of the elimination procedure. The following definitions allow a quantification of operative matrix elements as a function of B, the matrix bandwidth. This will ultimately lead to quantification of crucial parameters of the computing structure, also functionally related to B.

An N x N band form matrix will be defined as

$$\underline{A} = \left\{ a_{ij} : a_{ij} = 0, \forall \left| i - j \right| > B \right\} \tag{3-1}$$

where B, the bandwidth is given by

$$B = \max\left\{ \left| i - j \right| : a_{ij} \neq 0 \right\} \tag{3-2}$$

for i, j = 1, 2,..., N. Note that structural symmetry only (not absolute symmetry) is implicit in this definition. Furthermore, B, as defined here, is sometimes referred to as the half-bandwidth, spanning from the matrix diagonal to the rightmost nonzero matrix element.

The entire equation system, $\underline{A} \cdot \underline{x} = \underline{b}$, can then be described by the pair $(\underline{A}, \underline{b})$, where $\underline{A} = \left\{ a_{ij} \right\}$ is an N x N

matrix in structurally symmetric band form of bandwidth B and $\underline{b} = (b_1, b_2, \ldots, b_N)^T$ is a column vector of size N. Conducive to Gaussian elimination, the system can be described as an N x (N + 1) augmented matrix $\left\{ \underline{A} \mid \underline{b} \right\}$.

The elimination procedure will reduce $\left\{ \underline{A} \mid \underline{b} \right\}$ to upper triangular elements $U = \left\{ u_{ij} \right\}$ for i = (1, 2,..., N), j = (i, i + 1,..., i + B), augmented by a revised column vector $\underline{d}$. This can be easily illustrated by considering a 9 x 10 augmented system with a bandwidth of 3 as depicted in Figure 3.1a. After the upper triangulation has been performed, the system appears as illustrated in Figure 3.1b.

For larger systems, some more interesting aspects can be exposed. Consider a large $\underline{A}$ with $B \ll N$. An examination of the elimination of row k from such a system, as shown in Figure 3.2, reveals some useful information.

The shaded areas of this figure represent the total number of matrix elements required to perform the elimination of row k. As depicted, the length, in matrix elements, of row k is 2B + 1. This will be called the working row. Neglecting the zero elements then, the width of an augmented working row is merely 2B + 2 nonzero elements.

Examining the required upper triangular elements needed to eliminate the augmented working row, a simple geometric analysis reveals that $B^2 + B$ elements of $\underline{A}$ are required. In addition, again neglecting null entries, B elements of the $\underline{d}$ vector (resolved $\underline{b}$ vector elements) are required. Thus the total number of active nonzero elements involved in the

$$
\begin{array}{ccccccccc|c}
a_{11} & a_{12} & a_{13} & a_{14} & & & & & & b_1 \\
a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & & & & & b_2 \\
a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & & & & b_3 \\
a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & & & b_4 \\
 & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} & & b_5 \\
 & & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} & a_{69} & b_6 \\
 & & & a_{74} & a_{75} & a_{76} & a_{77} & a_{78} & a_{79} & b_7 \\
 & & & & a_{85} & a_{86} & a_{87} & a_{88} & a_{89} & b_8 \\
 & & & & & a_{96} & a_{97} & a_{98} & a_{99} & b_9
\end{array}
$$

Figure 3.1a Augmented matrix $\{ \underline{A} \mid \underline{b} \}$.

$$
\begin{array}{ccccccccc|c}
u_{11} & u_{12} & u_{13} & u_{14} & & & & & & d_1 \\
 & u_{22} & u_{23} & u_{24} & u_{25} & & & & & d_2 \\
 & & u_{33} & u_{34} & u_{35} & u_{36} & & & & d_3 \\
 & & & u_{44} & u_{45} & u_{46} & u_{47} & & & d_4 \\
 & & & & u_{55} & u_{56} & u_{57} & u_{58} & & d_5 \\
 & & & & & u_{66} & u_{67} & u_{68} & u_{69} & d_6 \\
 & & & & & & u_{77} & u_{78} & u_{79} & d_7 \\
 & & & & & & & u_{88} & u_{89} & d_8 \\
 & & & & & & & & u_{99} & d_9
\end{array}
$$

Figure 3.1b Upper triangular elements of matrix $\{ \underline{A} \mid \underline{b} \}$.

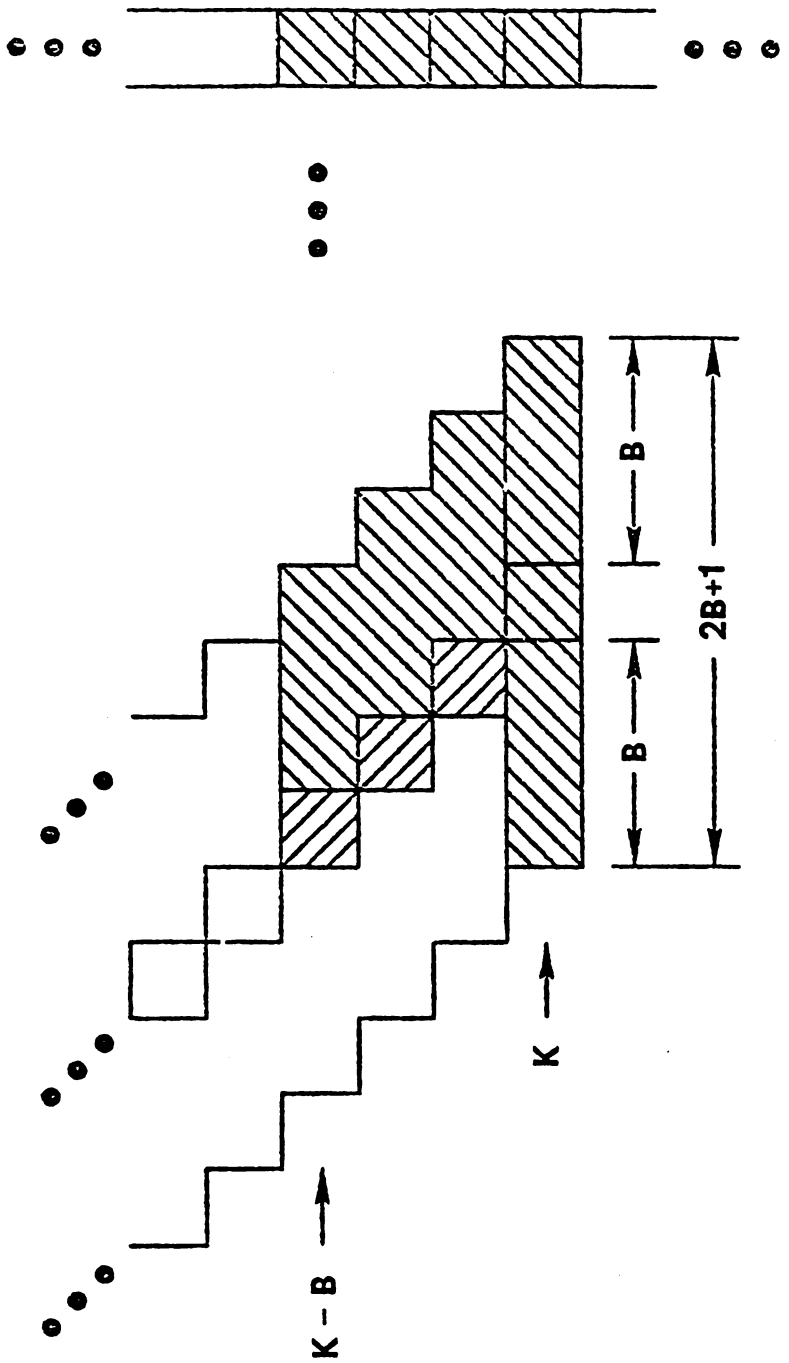

Figure 3.2 Large band form matrix.

elimination of augmented row k is given

$$E = B^2 + 4B + 2 \tag{3.3}$$

These elements, i.e., the shaded areas of Figure 3.2, will be referred to as the "window" of elements required to eliminate any given row k.

Now, focusing on the development of an ideal systolic computing array for such a band form system, a reasonable objective is to have the size of the array, in processing elements, related to this window of elimination. The window can then be viewed as "sliding down" the main diagonal eliminating rows as it passes. This is analogous to the function of the systolic array; row k is eliminated while the new augmented working row k + 1 is brought in; simultaneously, the resolved elements just above the window are placed on the output lines.

The advantage of such a scheme is simple and of great consequence. The size of the computing array, as will be shown, is limited by B, the bandwidth, in both width and depth.

3.2 Array Structure and Timing

Creation of the actual systolic array structure required manipulating a set of possible coefficient input patterns and arithmetic processing cells in order to produce properly resolved upper triangular elements at the output ports. Restrictions were based on structural simplicity,

regular connectivity and an ideal upper bound of $O(B^2)$ processing cells. Actual input alignment, as will be seen, required the injection of dummy zero and one elements, used as "spacers" in the operand string. The output string, also regularly interlaced with spacers, contains the necessary resolved upper triangular factors and the resolved column vector $\underline{d}$ for subsequent back-substitution.

The systolic array operates synchronously with latch arrays controlling operand flows between processing cells. There are no storage elements, save the latches, in the array.

The most difficult part of the design was to correctly align the $\underline{b}$ vector element stream through the array. Many of the possible input patterns and cell permutations presented data interference and conflict problems. This was eventually overcome by isolating those processing cells which perform the actual operations of resolving $\underline{b}$ into $\underline{d}$; this portion of the hardware is called the D section. It is, however, constructed of precisely the same cells as the rest of the systolic array, differing only in interconnection pathways.

The array structure for a matrix of arbitrary dimension, N, and of bandwidth B = 3 is presented in Figure 3.3. Two types of processing cells are used, corresponding to the two operations of the Gaussian elimination procedure.

The first and majority cell is labeled MAC (Multiply and Add Cell). It has previously been called an inner

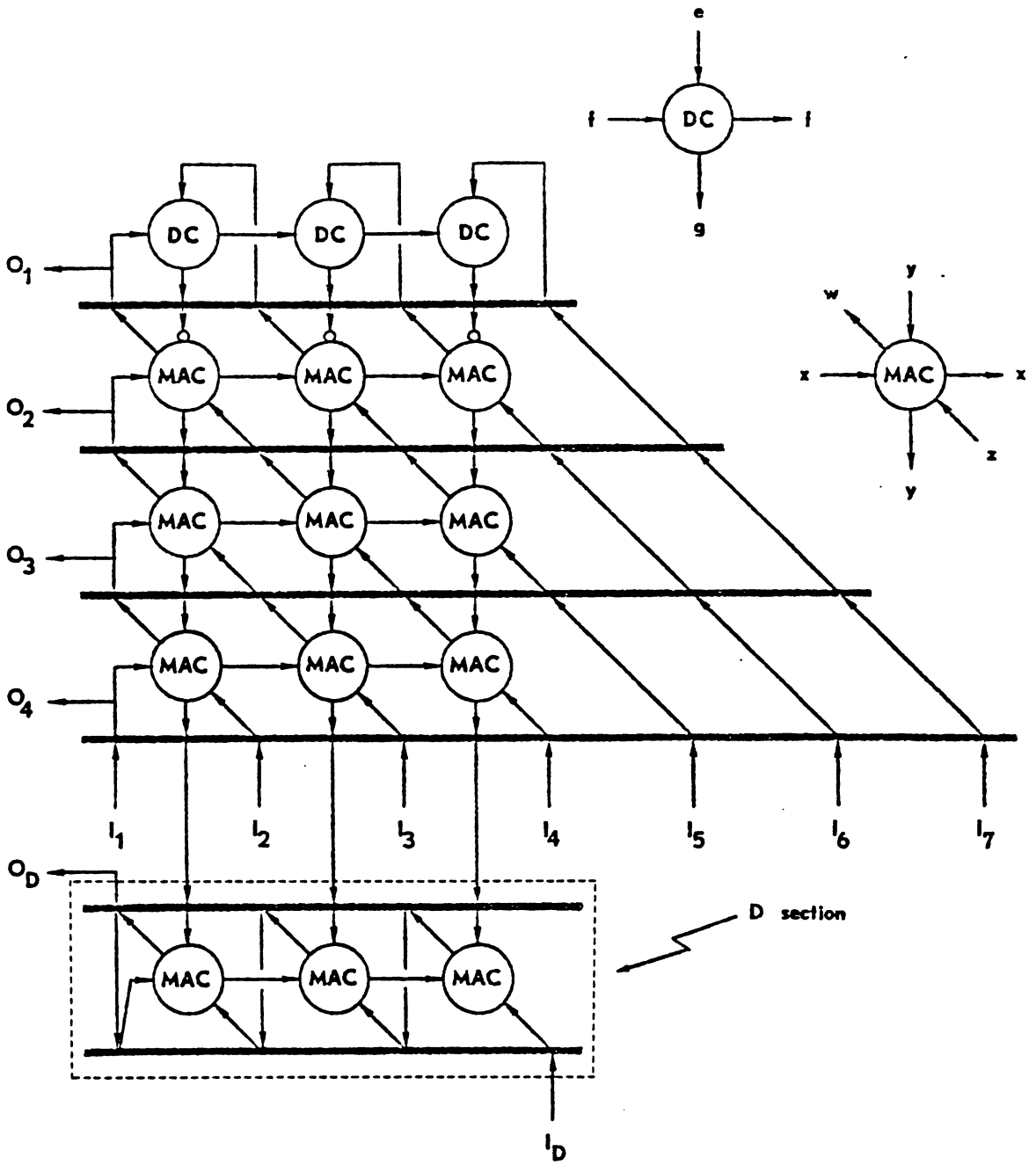

Figure 3.3 Computing array structure for matrix of arbitrary dimension with B = 3.

product step processing element, and is a 3-input, 3-output cell calculating $w = xy + z$ and performing the transfer $x = x$ and $y = y$. The second cell performs the diagonal divide operation and is labeled DC (Division Cell). It is a 2-input, 2-output cell. The DC performs the division $g = e/f$ and the transfer $f = f$. Both cell types are illustrated in Figure 3.3.

The complementation circle shown on the input of the topmost row of MAC's refers to a two's complement operation. Inter-row registration, providing operand synchronization, is depicted by the thick black lines.

The array structure supports separate upward and downward traffic flows. Input coefficients are synchronously pumped into ports I_1 through I_7. The number of required input ports is given by the full breadth of the matrix band or $2B + 1$ (refer to Figure 3.2). Columns of $\underline{A}$ enter the ports every two cycles interspersed with the proper spacer elements. Table 3.1, a timing table, illustrates the operand strings for the example $B = 3$ corresponding to Figure 3.3.

Once this initial upward stream reaches the DC level it proceeds downward toward the D section. At this point in time, t_3 in the example, $\underline{b}$ vector operands begin to enter port I_D of the D section. At the end of t_3, b_1 reaches the w output of the rightmost MAC in the D section and latches appropriately. During t_5, b_2 pumps in from I_D and b_1 is transferred to the bottom latch of the D section waiting to

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_0	O_1	O_2	O_3	O_4	O_0
t_1				a_{11}	a_{21}	a_{31}	a_{41}						
t_2				↓									
t_3			a_{12}	a_{22}	a_{32}	a_{42}	a_{52}	b_1					
t_4				↓					u_{11}				
t_5		a_{13}	a_{23}	a_{33}	a_{43}	a_{53}	a_{63}	b_2		u_{12}			
t_6				↓					u_{22}		u_{13}		
t_7	a_{14}	a_{24}	a_{34}	a_{44}	a_{54}	a_{64}	a_{74}	b_3		u_{23}		u_{14}	
t_8				↓					u_{33}		u_{24}		d_1
t_9	a_{25}	a_{35}	a_{45}	a_{55}	a_{65}	a_{75}	a_{85}	b_4		u_{34}		u_{25}	
t_{10}				↓					u_{44}		u_{35}		d_2
t_{11}	a_{36}	a_{46}	a_{56}	a_{66}	a_{76}	a_{86}	a_{96}	b_5		u_{45}		u_{36}	
t_{12}				↓					u_{55}		u_{46}		d_3
t_{13}	a_{47}	a_{57}	a_{67}	a_{77}	a_{87}	a_{97}		b_6		u_{56}		u_{47}	
t_{14}				↓					u_{66}		u_{57}		d_4
t_{15}	a_{58}	a_{68}	a_{78}	a_{88}	a_{98}			b_7		u_{67}		u_{58}	
t_{16}				↓					u_{77}		u_{68}		d_5
t_{17}	a_{69}	a_{79}	a_{89}	a_{99}				b_8		u_{78}		u_{69}	
t_{18}				↓					u_{88}		u_{79}		d_6
t_{19}								b_9		u_{89}			
t_{20}									u_{99}				d_7
t_{21}													
t_{22}													d_8
t_{23}													
t_{24}													d_9

Table 3.1 Port-coefficient timing table.

enter input z of the second rightmost MAC.

Once the first column of lower triangular factors reaches the y inputs of the D section, the bottommost latches are filled with the required b_i elements so that the generation of the new right hand side vector elements, d_i, i = 2, 3,..., N, can begin. Note that during the previous time step, d_1 (which equals b_1) appeared at port O_d of the D section. Then, D section outputs appear at port O_d every two time slots.

3.3 Cell, Port and Time Slot Counts

The salient property of this new algorithm is that crucial counts of input-output ports and processing cells are found to be functionally related to the matrix bandwidth and not the full matrix dimension. This fact is quite important as many systems of large dimension yield tightly banded matrices. Furthermore, many sparse coefficient matrices can be reordered into band form which for some problem types have been shown to yield B = 0.10 N [8].

It has been determined that the number of input/output ports are 2B + 2 and B + 2, respectively. It must be remembered that each port is m bits wide, m being the number of bits per word. This introduces a very critical pin limitation problem which will be addressed in the input/output circuit design chapter.

The algorithm requires far less processing cells than previously published schemes which could be classified as

requiring $O(N^2)$ cells, N being the full matrix dimension [16]. Specifically, B(B + 1) MAC's and B DC's are required, advantageously classifying the algorithm as needing just $O(B^2)$ cells. Of course, this can only be applied to those sparse coefficient matrices for which bandwidth reduction is effective.

As far as time requirements are concerned, the new algorithm requires 2N + 2B time slots, thus classifying it as an O(N) algorithm. This is on the same order as other systolic array algorithms, which require many more processing cells for this type of problem [16].

3.4 Floating-point Considerations

The computing structure presented in Section 3.2 is basically constrained to a fixed-point binary number system due to the nature of its processing element designs. It is possible, however, to consider processing fixed-point numbers which are the pre-adjusted mantissas of previous floating-point coefficients. This possibility, of course, is a crude approach to a true floating-point solution, but, due to current limitations of chip size and lithographic linewidth, it deserves further investigation for, at best, a near term application. It is realized, and must not be understated, that this approach is extremely vulnerable to the dynamic range of the input matrix entries and will work only for those problems which possess well tempered coefficients of tight dynamic range. A possible candidate for

this dedicated computing structure is the class of matrices with all positive and strictly diagonally dominant coefficients, such as the admittance matrix of a power system load flow problem. For a well conditioned matrix with these properties, no unpredictable overflow problem will occur during the elimination procedure if the following proposed fixed-point scheme is considered.

In fixed-point number systems, the two most commonly selected positions for the radix point are either at the left extreme or at the right extreme of the magnitude position of the number [20]. To allow the PE's of the computing array to operate in both number systems, the left extreme is a more logical choice. Here, the radix point lies between the sign bit and most significant bit. Moreover, this dictates that all fixed-point numbers be strictly less than one. To adjust floating-point numbers to this fixed-point scheme, the host processor (or some intermediate processor which will not be considered here) must first sort out the maximum operand. For the matrices under consideration, this element lies on the matrix diagonal. The mantissa value of this maximum operand is normalized and the exponent value is stored. The other operands in the matrix are then pre-adjusted in floating-point to the same exponent value as the maximum operand.

As it is assumed that the latches of computing structure are reset before any operands are pumped in, it is obvious that the first significant arithmetic operation,

which could possibly offset the exponent, is at the top row of processing elements, the DC's. From the example of Figure 3.1a it can be seen that during t_4, the operands working in the leftmost DC are dividend a_{21} and divisor a_{11}. If we express the pre-adjusted floating-point operands as,

$$a_{11} = M_1 \times 2^e \tag{3-4}$$

$$a_{21} = M_2 \times 2^e \tag{3-5}$$

where e is the exponent value of the maximum operand, the quotient

$$Q = \frac{A_{21}}{A11} = \frac{M_2 \times 2^e}{M_1 \times 2^e} = \frac{M_2}{M_1} \tag{3-6}$$

has an exponent value of zero. This quotient becomes the multiplier for the next downstream MAC. The MAC algorithm, $w = xy + z$, restores the exponent value to e. Moreover, zero exponent quotients continue to be pumped out of the DC's every time slot and these factors propagate vertically down through the array. Consequently, the exponent values of the resolved upper triangular elements at the output port are all equal to the exponent value of the original input elements.

A computing structure with this potential can operate with only the adjusted mantissas of the input operands. The fixed exponent value can be retained by the host processor and used for post-adjustment upon return of the upper triangular elements prior to back-substitution.

3.5 Function Block Design

Examination of the computing array depicted in Figure 3.3 reveals that only three schematic logic circuit diagrams of function blocks (MAC, DC and latch) are needed to estimate the area geometries and propagation delays. Again, the highly regular, localized wire routing among these building blocks will reduce the design complexity, delay time and layout area of the computing structure.

Many addition, multiplication and division algorithms can be called computationally efficient, but, for exclusive hardware implementation, especially in this class of a data oriented VLSI chip, the final algorithm selecting criteria must also be judged in consideration of the design task.

It is most desirable to choose MAC and DC algorithms based on existing arithmetic algorithms that can be implemented in simple and regularly structured combinational circuits. For circuit and logic simulation, design verification and test validation, combinational circuitry is preferable to sequential circuitry [19]. Another important criterion is that those MAC and DC algorithms which are best realized in locally connected array networks of unique, simple and regular functional cells will considerably ease the design of the building blocks themselves. This type of network will have several other advantages. First, it can provide increased throughput due to the parallel and pipelined processing structure of the algorithm. Second, function cells which only communicate with nearest neighbors are

very advantageous in that they minimize interconnection requirements.

With these criteria in mind, the MAC and DC algorithms were designed as follows:

A. MAC Algorithm -

The high speed cellular array multipliers such as Wallace tree [21], Pezaris array [22], and Baugh-Wooley array [23] were evaluated based on the design criteria discussed above. The Baugh-Wooley algorithm, with its fast multiplication speed (only 2n full-adder delay time for an n x n multiplication, where n is word size), and regular full adder structure [20], was determined to be the best among these three. Furthermore, this algorithm allowed for straightforward extension of the design to the desired MAC function of multiplication followed by addition by placing an extra row of full-adders at the bottom edge of the Baugh-Wooley multiplier. Thus the MAC function is obtained in only one more full-adder delay time. The MAC array is shown in Figure 3.4. Note that the portion within the dotted line is the original Baugh-Wooley multiplier.

For design layout, consider the following equations of the one-bit full-adder function for a standard gate-level implementation:

$$S_i = A_i + B_i + C_i \tag{3-7}$$

$$C_{i+i} = A_iB_i + B_iC_i + A_iC_i \tag{3-8}$$

where A_i and B_i are inputs to the current stage and C_i is

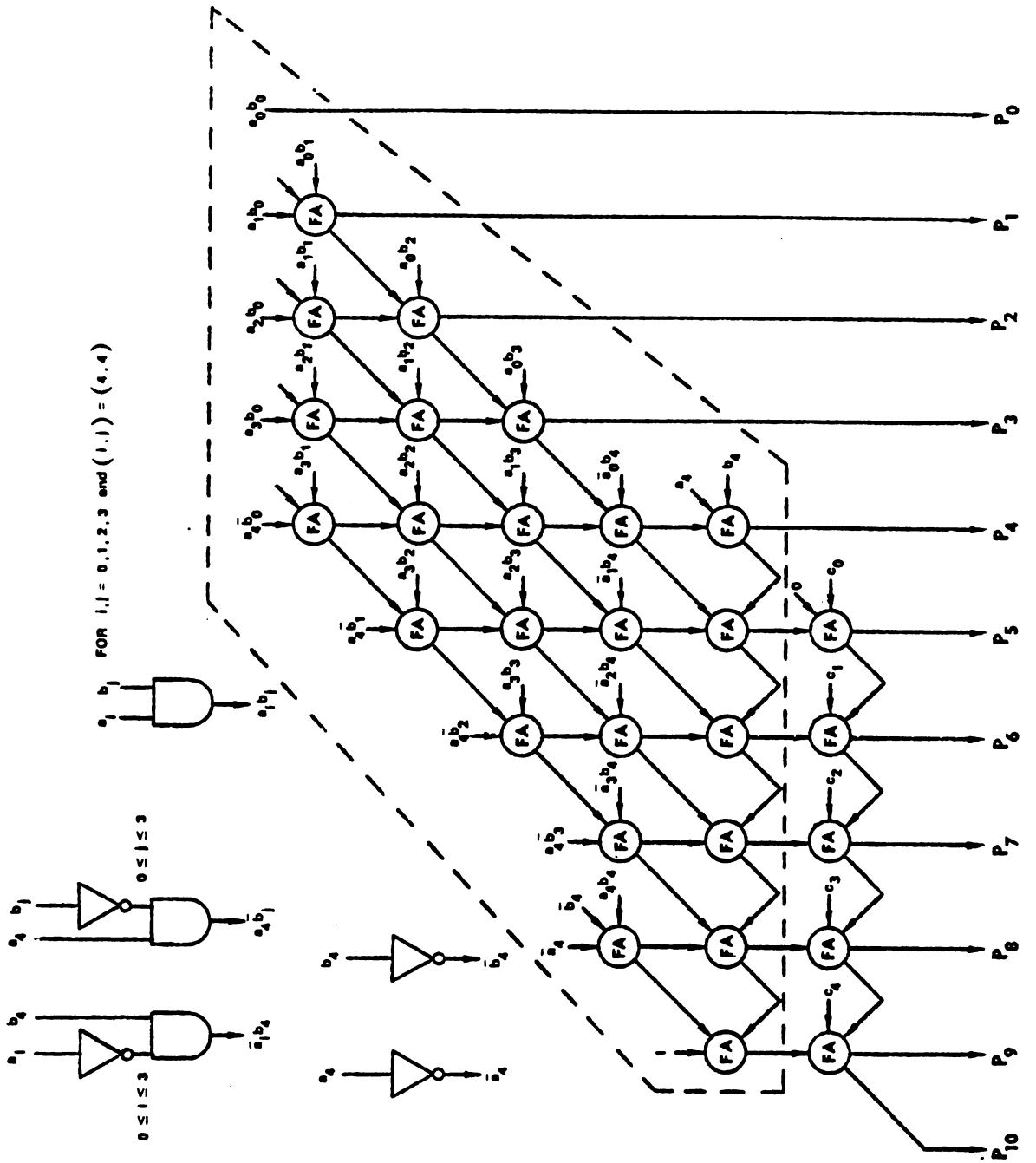

Figure 3.4 Logic circuit diagram of a 5-by-5 Baugh-Wooley based MAC.

the carry from the previous stage.

The wire-logic gate implementation of this full-adder is shown in Figure 3.5. This design, with only a two gate level delay, provides a minimal propagation delay.

B. DC Algorithm -

Several existing high speed division algorithms are considered feasible for VLSI implementation. These include the Cappa-Hamacher array divider [20, 24, 25] and two types of multiplicative division schemes, the convergence division algorithm and divisor reciprocation division algorithm [20].

In a pipelined structure, the total propagation delay of the system is a function of the maximum delay of the longest segment. However, the Cappa-Hamacher divider algorithm's computational time, in terms of word size, is at least two times that of the Baugh-Wooley algorithm [20]. A consistency of computational speeds between MAC and DC algorithms is crucial. Furthermore, it is ideal to strive to make the MAC propagation delay the limiting factor of clock rate as it will most likely be the fastest element. Limiting the clock rate by a much slower element will degrade the computational efficiency of the entire chip. Therefore, it is desirable to set the clock rate as the reciprocal of the sum of MAC time plus a latch time and find a DC algorithm within this bound. An alternative algorithm which partitions the complex division function into several subtasks, serving as segments of a pipelined structure, is the prime candidate.

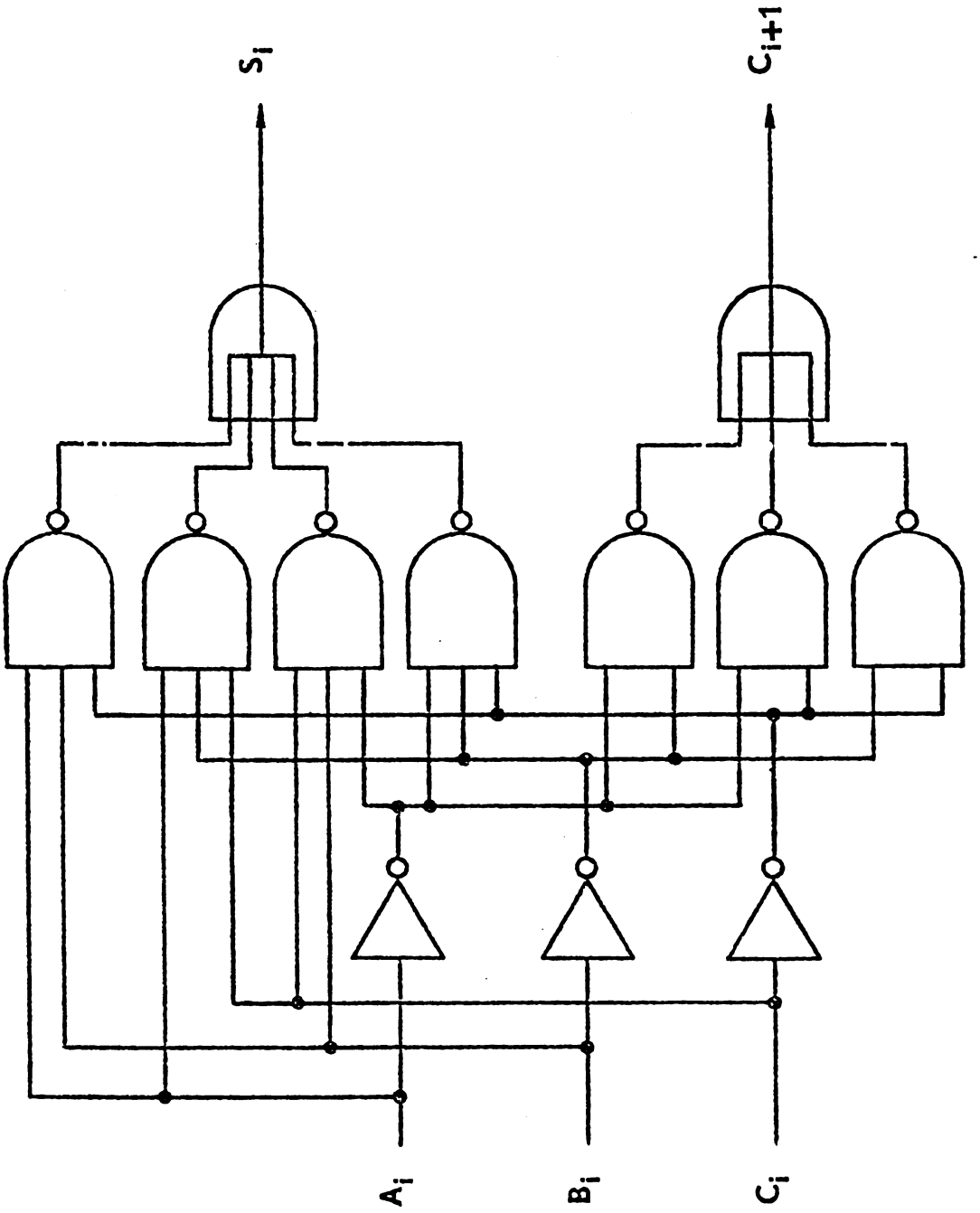

Figure 3.5 Logic gate diagram of a one-bit full adder [20].

Both the convergence division and divisor reciprocation division algorithms can be implemented in segmented pipeline structures. They both approach the final result of quotient or divisor reciprocal quadratically. Through detailed study of these two algorithms, the hardware requirements are found to be approximately equivalent. But, there is a key advantage to the convergence division algorithm; that is, the number of iteration steps can be determined a priori in terms of word size. This factor is very important in an exclusive hardware implementation because no software convergence checking is required.

The convergence division algorithm operates by multiplying both the denominator and numerator with the same sequence of convergence factors until the denominator approaches unity. To illustrate this, consider the division operation,

$$Q = \frac{N}{D} . \tag{3-9}$$

The right hand side can be expressed as

$$\frac{N}{D} \times \frac{R_0}{R_0} \times \frac{R_1}{R_1} \times \cdots \times \frac{R_n}{R_n} \tag{3-10}$$

where the R_i's are a set of convergence coefficient factors whose goal is to force the denominator to unity and thus produce the quotient. If the resolved denominator comes up

to be one, that is,

$$D \times R_0 \times R_1 \times \cdots \times R_n \longrightarrow 1 \tag{3-11}$$

then the quotient can be expressed as

$$Q = N \times R_0 \times R_1 \times \cdots \times R_n. \tag{3-12}$$

Assuming that D is positive and fixed-point arithmetic is selected, then,

$$1 > D > 0. \tag{3-13}$$

In order to converge faster, D should be normalized first, that is,

$$1 > D' \geq \frac{1}{2} \tag{3-14}$$

where D' is the divisor after normalization. Then, N must be shifted as many bits as D, thus,

$$\frac{N'}{D'} = \frac{N}{D}. \tag{3-15}$$

Now, a number δ can be defined which makes

$$D' = 1 - \delta \tag{3-16}$$

and then

$$0 < \delta \leq \frac{1}{2}. \tag{3-17}$$

We can choose the first multiplying factor as

$$R_0 = 1 + \delta \tag{3-18}$$

so that $D_0' = D' \times R_0$

$$= (1 - \delta)(1 + \delta)$$

$$= 1 - \delta^2. \tag{3-19}$$

Now choosing

$$R_1 = 1 + \delta^2 \tag{3-20}$$

then $D_1' = D_0' \times R_1$

$$= (1 - \delta^2)(1 + \delta^2)$$

$$= 1 - \delta^4. \tag{3-21}$$

Therefore, in general

$$D_i' = D_0' \times R_0 \times R_1 \times \cdots \times R_i$$

$$= 1 - \delta^{2^{i+1}}. \tag{3-22}$$

In the binary case, $\delta \leq \frac{1}{2}$, therefore

$$\delta^{2^i} \leq 2^{-2^i}. \tag{3-23}$$

Now, consider a word size of 16 bits implying 15 significant digits. From equations 3-22, in order to make $D_i' = 1$ requires

$$\delta^{2^{i+1}} < 2^{-15}. \tag{3-24}$$

Then equation 3-24 gives i = 3. In other words, multiplying factors R_0, R_1, R_2, R_3 are required to make D_3' converge to 1 and

$$Q = N' \times R_0 \times R_1 \times R_2 \times R_3. \tag{3-25}$$

For the same reason, if the word size is 24 bits, 5 multiplying factors are required and so on. Finally, the quotient value should be shifted right by the same number of bits as the divisor, D, has shifted left to normalize the result.

The Convergence Division algorithm was chosen for use in the DC's due to the fact that no software convergence checking is required. The hardware implmentation of a DC working with 16 bits per word is illustrated in Figure 3.6.

C. Latch Design -

There are two basic types of latches -- static and dynamic. A traditional static latch is a D-type flip-flop as shown in Figure 3.7. The disadvantage of this latch is that it requires four 2-input NAND gates and one INVERTER; also four units of gate delay. These area and time requirements are more than those required by a dynamic latch.

Dynamic latches are known to be ideal in high speed, synchronous processor chip designs, especially using VLSI [9, 19]. The dynamic latch, shown in Figure 3.8, consists of two pass transistors and two inverters. It uses a two-phase nonoverlapping clock to load and refresh the input data. The utilization of pass transistors has the advantages of a simple topology of interconnections and elimination of V_{DD} and GND connections, thus reducing wire routing geometry and power requirements.

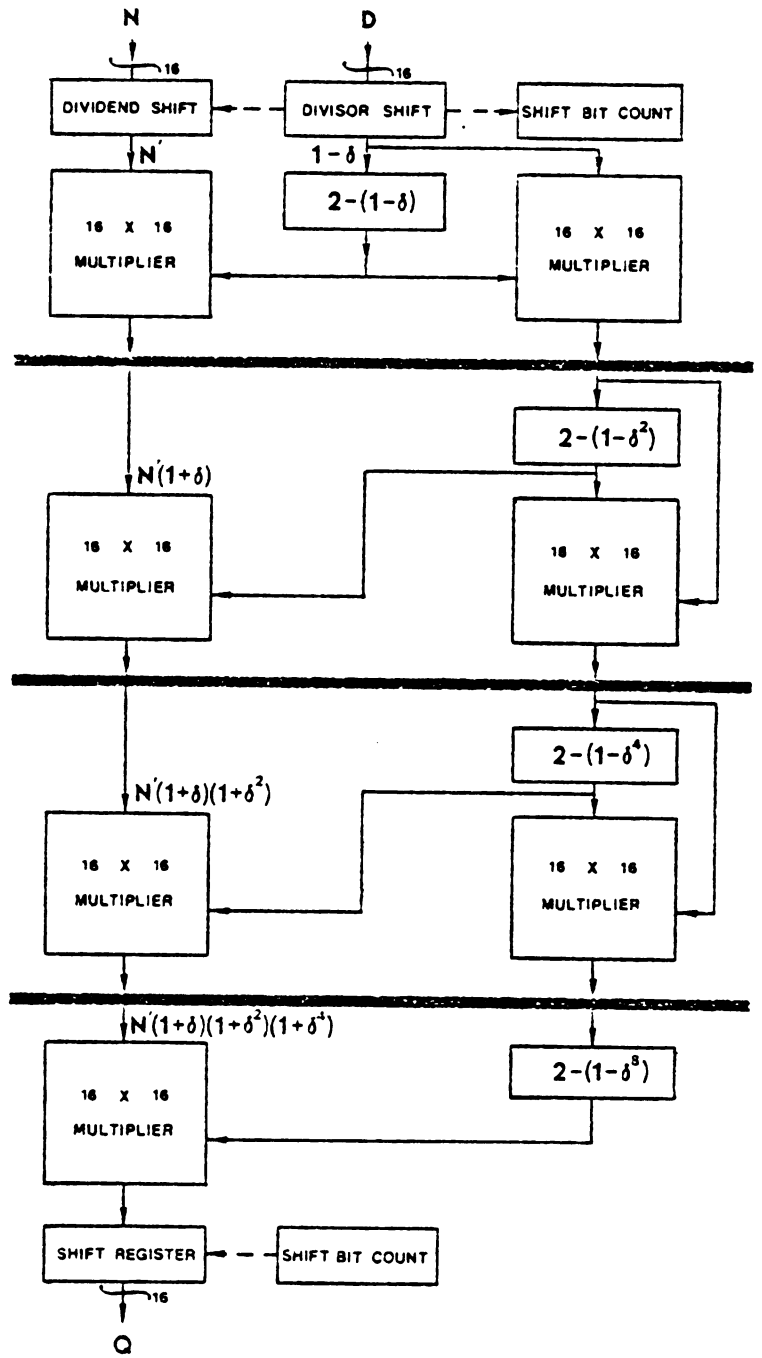

Figure 3.6 A 16-by-16 convergence division algorithm based DC.

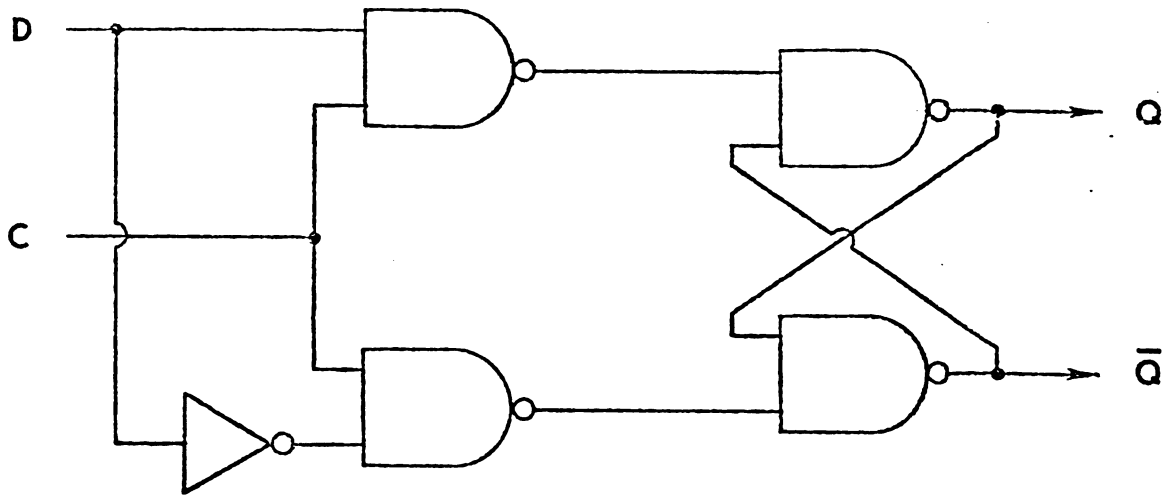

Figure 3.7 D-type flip-flop.

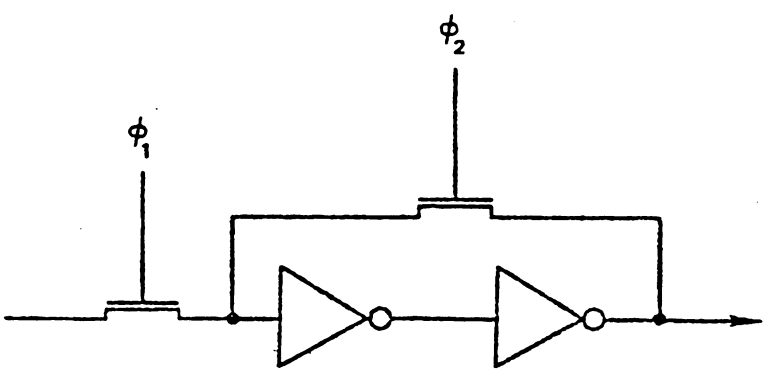

Figure 3.8 Dynamic latch controlled by two-phase nonoverlapping clock [9].

CHAPTER IV

INPUT/OUTPUT CIRCUIT DESIGNS

Operand I/O for VLSI structures inherently suffers from a basic pinout limitation problem due to practical integrated circuit packaging constraints. This necessitates the incorporation of serial-to-parallel demultiplexer and parallel-to-serial multiplexer circuits at the front and rear ends of the computing array, respectively. Furthermore, very high speed DMUX/MUX designs are required to provide sufficient operand broadcasting to the row of input cells which, in the case considered in this thesis, requires 2B + 2 operands simultaneously, where B is the matrix bandwidth. Additionally, any I/O circuit design for efficient VLSI implementation must also be optimized with respect to minimum chip real estate requirements and practical pinout counts. Currently, 64 to 100 pins per chip are considered practical [26]. This limitation also affects the operand word size and consequently the precision of the internal computations.

The purpose of this chapter is to present three possible input/output routing strategies for the VLSI computing array presented in the last chapter. Logic circuit diagrams of these I/O designs are developed and are used in the simulation described in the next chapter. The simulation provides comparative information on propagation delays and chip area requirements.

The design methodology used here is identical to that used in designing the computing array network and processing cells. In the logic circuit diagrams of the DMUX/MUX structures, pass transistors and inverters are used for building switch logic and dynamic latches in place of traditional logic gate structures. The new structures have simpler interconnection topologies and smaller power requirements. Morever, gains in area geometry and speed are also anticipated.

Ideally, the I/O circuit structure should be fast enough to avoid any I/O bottleneck. This must be accomplished with each input and output bus being only one word wide due to pin limitations. The three I/O strategies are presented in the following sections.

4.1 SCS Input/Output Circuit

The first I/O circuit candidates are illustrated in Figures 4.1 and 4.2, respectively. In the input circuit (Figure 4.1), the SCS (Shift-Register Control Sequence) is stored in a one-bit wide control queue. This sequence is synchronously pumped into the one-bit shift register chain and is used to select the proper operand input channel. The first bit of this control queue is logic "1", the others are logic "0". At the end of the first cycle, the output of the first shift register is "1". The load pass transistors of the top input channel are "on". Consequently, the first data operand can propagate through the top channel and is

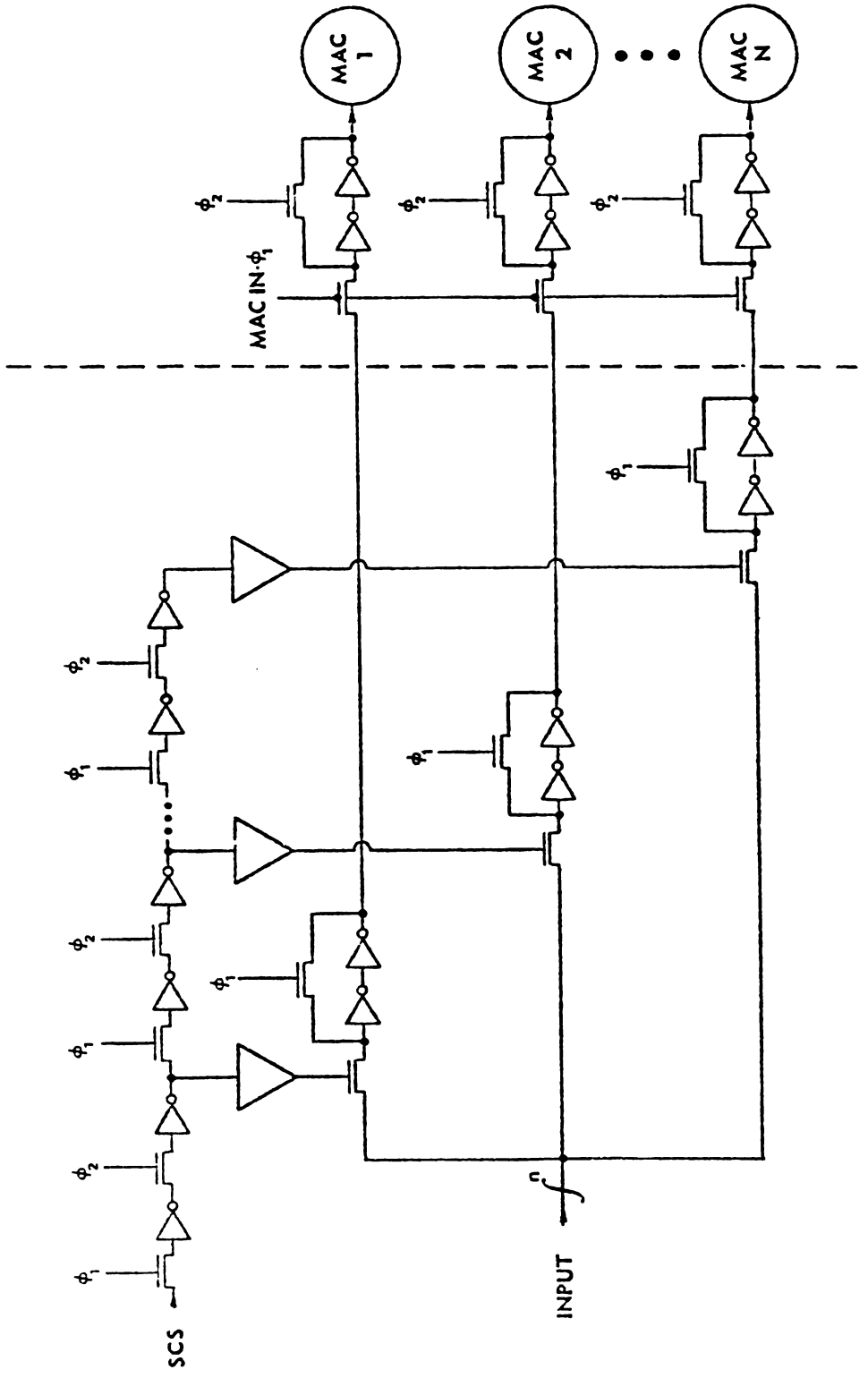

Figure 4.1 SCS input circuit diagram.

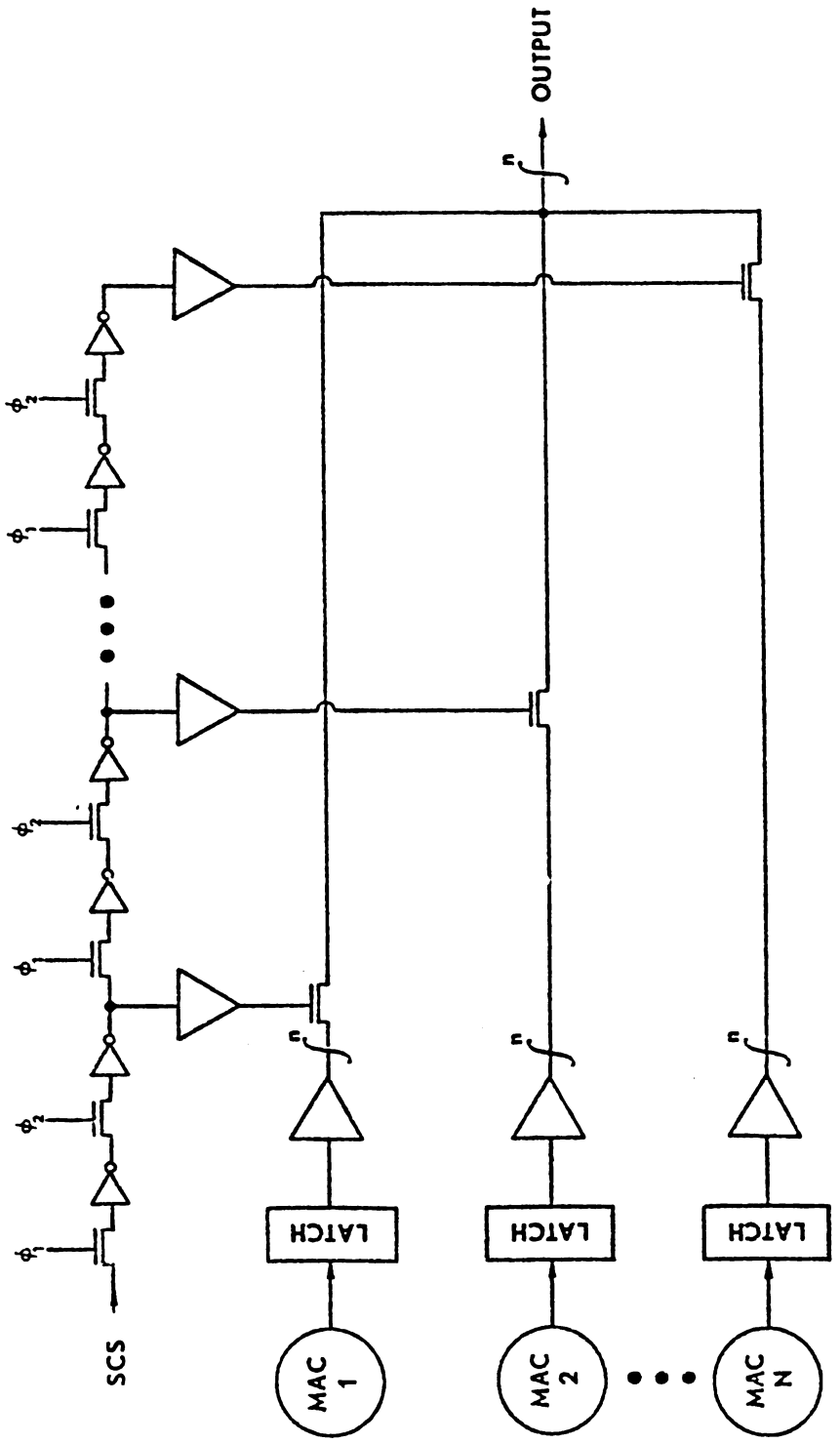

Figure 4.2 SCS output circuit diagram.

refreshed by the ϕ_1 clock signal. At the next clock cycle, logic "1" of the control queue moves onto the output of the second stage of the shift register. Then, the second channel is selected as the input channel. After N data operands are inside of the input structure, refreshing at every ϕ_1, a control signal called MAC IN$\cdot\phi_1$ (see Figure 4.3) enables those N operands into the corresponding latches. These latches are used to control the data flows and to stabilize the operands.

The output circuit of this strategy is even simpler. Identical to the input circuit, the pass transistors on the output channels are controlled by an identical SCS. After N data operands are pumped into latches (by the MAC OUT$\cdot\phi_1$ signal (see Figure 4.3)) they are refreshed by ϕ_2. Sequential output channels are then selected from top to bottom until the operand of the Nth MAC flows out. Quite conveniently, channel pass transistors here also serve as the tri-state output gates!

4.2 Binary Tree Structure

The input DMUX circuit, which is illustrated in Figure 4.4, features three important concepts. First, it uses the abstract structure of a binary tree, a basic interconnection network architecture. Second, it is a revised version of the traditional block DMUX tree. Third, and most importantly, it utilizes pass transistor and shift register concepts in the routing circuitry so as to decrease power

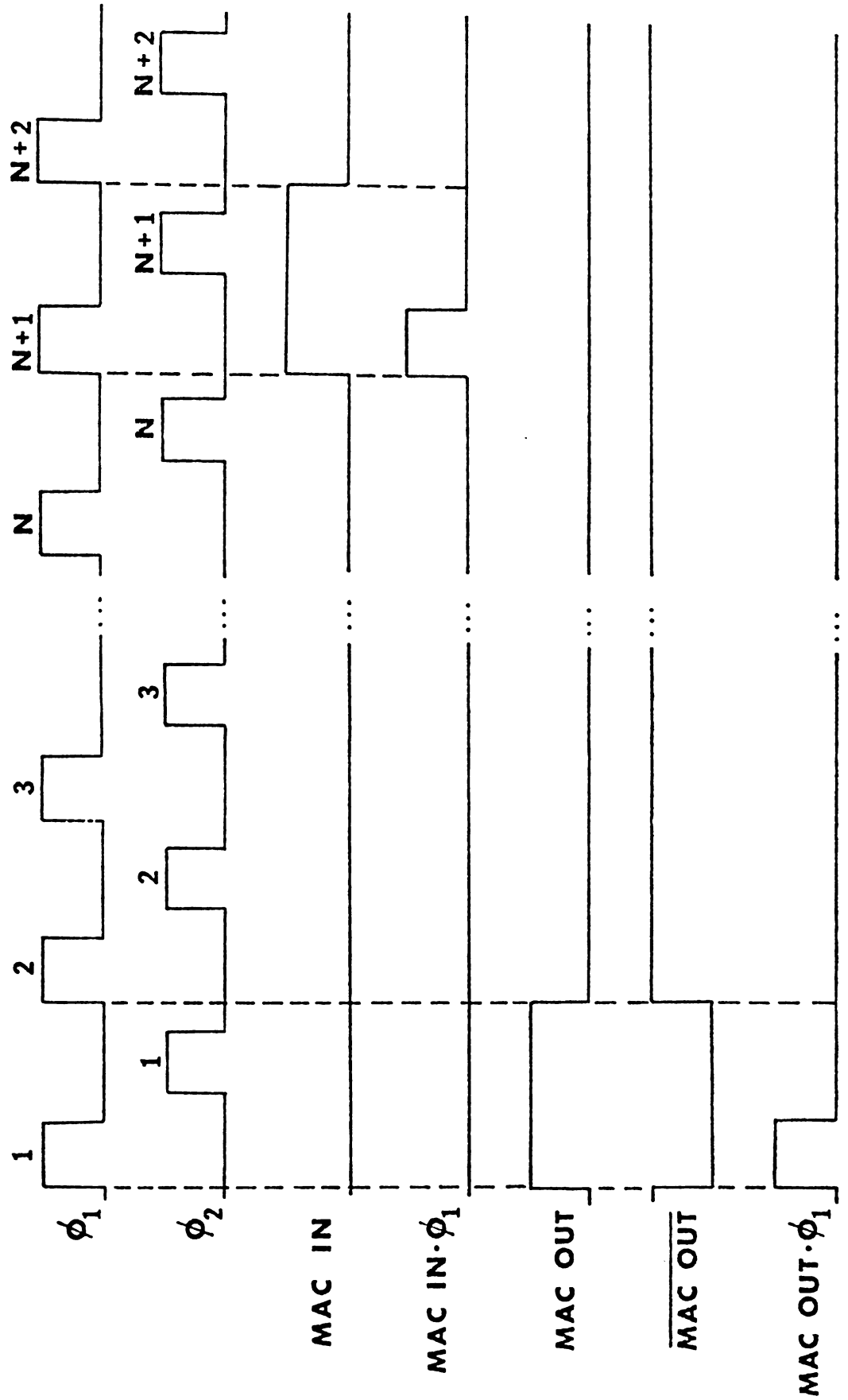

Figure 4.3 Input and output circuit control signal timing diagram.

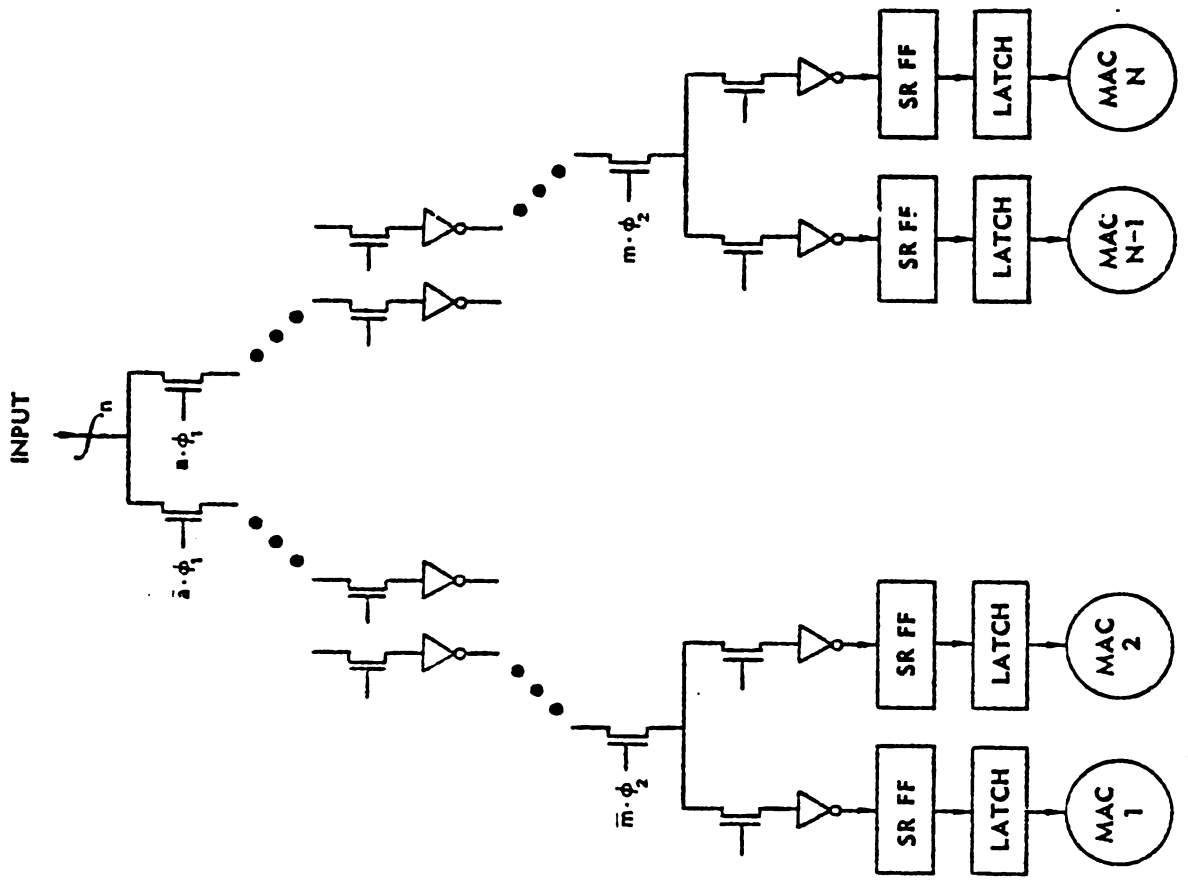

Figure 4.4 Binary tree input structure.

requirements and area geometry. Moreover, this will facilitate better testability for tracking data flow.

This structure uses control signals to select on or off for each pass transistor. When clock signal ϕ_1 rises, a stream of "on" pass transistors generate an input channel for a set (a word) of data which is assumed to be ready at the input port. The operand propagates along and its bits are momentarily held by the input capacitances of the first bank of inverters. Then, when clock ϕ_2 rises, one of the two down-stream pass transistors allows the operand to flow through a branch and it is again held via the input capacitances of the second bank of inverters. The data is then stabilized by the SR flip-flops providing temporary storage. After all data operands are pumped in, they are simultaneously clocked into the latches and refreshed by ϕ_2. The control signals and structure of the latches are identical to those described in Section 4.1. The operands remain in the latches until the next data stream replaces them.

The output MUX circuit, which is shown in Figure 4.5, is basically a mirror image of input DMUX circuit. After the rear edge of MAC's complete their computation, a set of latches are loaded with the resolved operands and are thus isolated from interference with the MAC's further computation. Subsequently, each data operand is pumped out of the chip through an output channel selected by appropriate control signals.

Among the disadvantages of this strategy is the

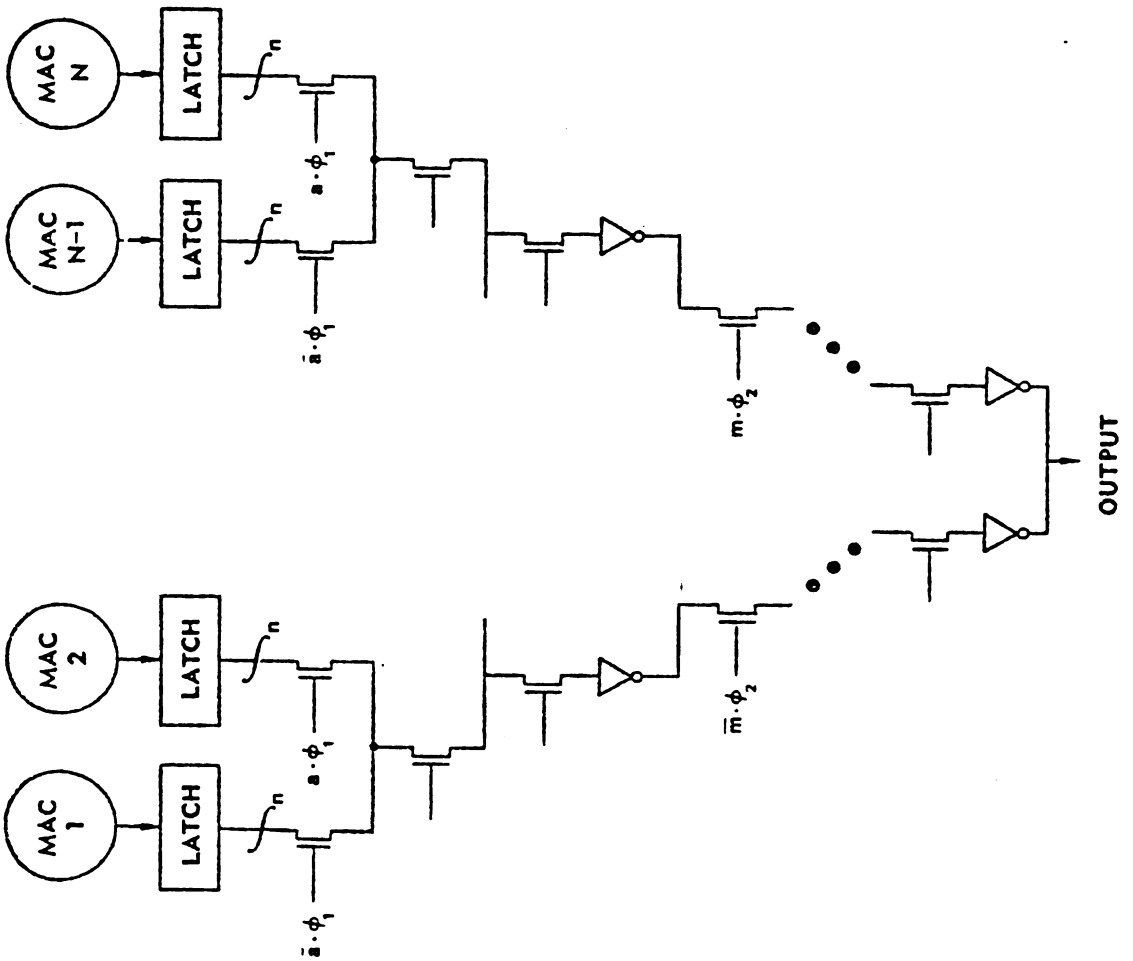

Figure 4.5 Binary tree output structure.

cumbersome control signal requirements, which alone may severely increase the pinout count. Additionally, the nature of the regular binary tree requires that the number of destination nodes be a direct power of 2. This will imply that much of the tree is wasted if the number of MAC's at the input level of the array does not correspond to a power of 2. For example, consider the case where there are 17 first row MAC's. Then a 1 to 32 input tree is implied, resulting in a waste of valuable chip area. Of course, irregular binary tree designs are possible, but as will be shown in Chapter VI this binary tree structure is far from optimal even when a power of 2 match is obtained.

4.3 Data Controlled Input/Output Circuit

The major difference in this routing strategy over the others is that it pumps the data operands into the destination processing elements without using any channel-selecting control signal. It utilizes the FIFO (First-In-First-Out) stack concept employing n sets (where n is the number of bits per word) of N-stage shift register chains (where N is the number of the first level MAC's). This concept is illustrated in Figure 4.6.

The full set of N operands are pumped into the shift register chain during the first N clock cycles. This provides an entire set of operands available to the first row of MAC's at the (N + 1)th clock cycle. At this time, the (N + 1)th clock cycle, a control signal called MAC IN$\cdot\phi_1$

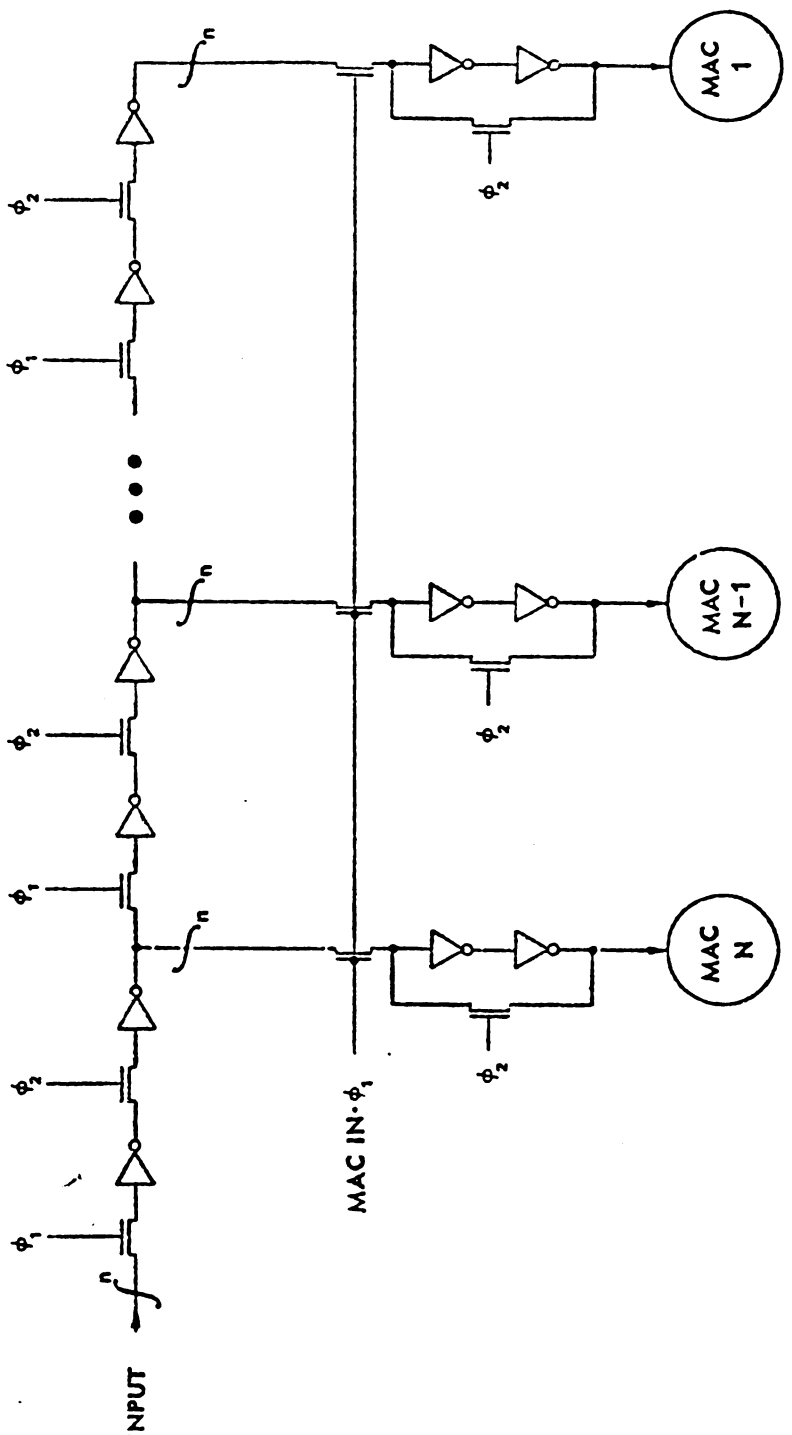

Figure 4.6 Data controlled input circuit diagram.

(see Figure 4.3) will simultaneously pump the data operands into their corresponding latches which are refreshed by ϕ_2. These operands will remain stable at the MAC inputs until the next MAC IN$\cdot\phi_1$ signal.

The data controlled output circut is illustrated in Figure 4.7. After computational results are loaded into the output latches, an output control signal, MAC OUT$\cdot\phi_1$ (see Figure 4.3) enables each operand through the corresponding first two pass transistors and one inverter of the shift register chain. Note that at this time, the pass transistors, which are controlled by the MAC OUT (see Figure 4.3), are off. They serve to eliminate possible interference between the data of each shift register during the data parallel-in operation. The next ϕ_2 cycle completes the parallel shift-in function. After this cycle, the MAC OUT signal is off (i.e., $\overline{\text{MAC OUT}}$ is on). The pass transistors controlled by $\overline{\text{MAC OUT}}$ are on, and they configure the remaining output circuit to be a serial-out circuit only. This output circuit then shifts one data operand out of the chip every clock cycle.

4.4 Discussion

Three input and output strategies as well as their related logic gate structures have been discussed and illustrated in this chapter. The MAC IN and MAC OUT control signals described in these I/O circuits are actually identical signals.

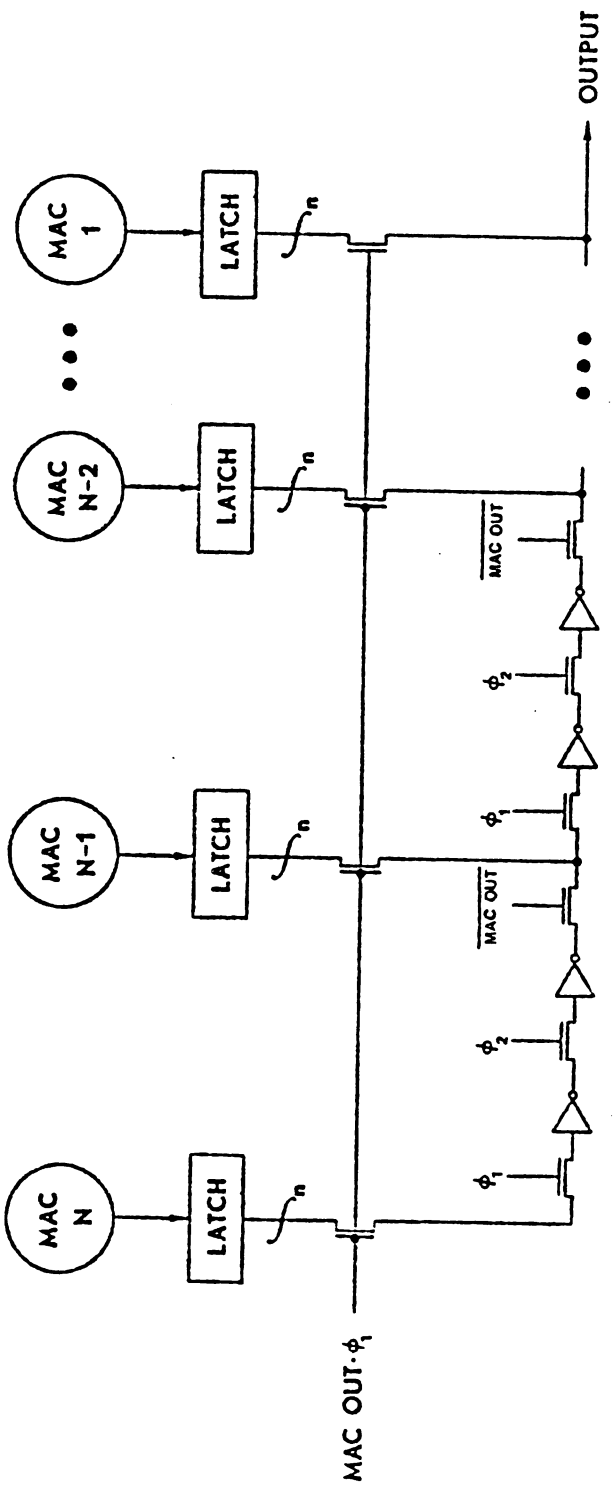

Figure 4.7 Data controlled output circuit diagram.

From the gate count view, input and output circuits of Figures 4.6 and 4.2 occupy the least area and provide minimum delay time. But, at the circuit layout level, the area geometry and propagation delay of each transistor must be estimated under various gate coupling and fan-out considerations. This information can then be extended for calculating total I/O propagation delay and area geometry of the entire circuit.

The next step requires the use of simulation models for accurately computing these area and time results. The simulation models utilized in this work will be discussed in the next chapter and the determination of the best candidate among these I/O designs will follow.

CHAPTER V

SIMULATION DEVELOPMENT

As stated before, the ultimate goal of the circuit simulation is to provide two basic tradeoff comparisons, throughput speed and chip size versus matrix bandwidth. Three fundamental parameters that may be manipulated in the simulation are minimum lithographic linewidth, λ , word size and matrix bandwidth. It will be assumed that the ratio of the band form matrix bandwidth to overall matrix dimension is one-tenth. This ratio is chosen as it has been shown that for a typical electrical power network the nonzero coefficients of the network admittance matrix can be reordered into a band form, whose bandwidth is about 10% of the original matrix dimension [8].

Of particular interest in the simulation is the manipulation of λ which can track current trends in I.C. fabrication technology. The so-called " λ model" and " τ model", introduced by Mead and Conway [9, 27], provide a simple and fast method for users to design and layout VLSI systems. Most advantageously, these models offer designers freedom from working with tedious computer simulations [27].

The λ model, a layout geometry design tool, assumes that the permissible layout linewidths and spaces along the diffusion, polysilicon and metal lines can all be scaled down in linear dimension. Consequently, a set of design rules can be expressed in dimensionless form where the

geometries of layout elements are described in terms of current minimum lithographic linewidth or the current resolution of a given process. As the I.C. fabrication technology advances, λ decreases and thus the layout element geometries, which are a function of λ, also decrease. But, there are some practical limitations to consider such as bonding pad size and spacings which might not shrink linearly [27]. As a result, some parts of the λ model need to be readjusted. Neglecting, however, the small nonlinearities, these basic concepts have proven useful for initial step design in the multi-chip project directed by Caltech [9, 27].

The τ model is a simple mathematic model for computing propagation delay of a transistor level circuit. This model acknowledges that the delay time of a node depends on the total capacitance of that node together with the gate capacitance and transit time of the driving transistor [9, 27]. For example, the propagation delay of an inverter, with gate capacitance C_g, driving a node with capacitance C_{total}, is calculated as follows. At first, it is assumed that the transit time τ is the delay time for an electron to pass through a channel of length L [9], where, for small V_{ds},

$$\tau = \frac{L^2}{\mu V_{ds}} \qquad (5\text{-}1)$$

Here μ is the electron mobility and V_{ds} is the drain to source voltage.

Next, assume that the inverter ratio K is the ratio of

channel length to width of pull-ups, Z_{pu}, to pull-downs Z_{pd}. Then, the "low-going" and "high-going" delay time for this inverter to an identical inverter are τ and $K \cdot \tau$, respectively. Therefore, an inverter with C_{total} load capacitance requires $(C_{total}/C_g) \cdot \tau$ and $(C_{total}/C_g) \cdot K \cdot \tau$ of "low-going" and "high-going" delay times, respectively. Another important feature of the τ model is the propagation delay calculation method for a pass transistor chain. Each transistor in the chain is thought of as a series resistance coupled with a capacitance to ground formed by the gate capacitance. Thus, the delay time through a chain of n identical pass transistors is n^2RC_g, where R is turn-on resistance and C_g is the gate capacitance of each pass transistor [9].

But the transit time, τ , is the fundamental limit on the switching speed of the gates. In a practical circuit, the speed of MOS device operation is determined realistically by the speed with which capacitors can be charged and discharged [28]. Therefore, a revised τ model was used in this simulation. It assumes that T is the discharge time for a basic inverter (with inverter ratio 4:1) coupled with only one identical inverter. Also the gate effective capacitance of every logic gate is assumed equal. Thus, the simulation results in the propagation delay section are more realistic than the τ model due to the fact that it uses discharge time, T, and not transit time, τ . Morever, this simplifies the process of estimating total load capacitance.

When given the desired number of bits per word and B

(matrix bandwidth), the Fortran-coded simulation program will model the required computing and I/O circuit structures. Then, at the transistor level, the total chip area is calculated based on the circuit model and λ. Propagation delay, also calculated at the transistor level, includes both active gate delays and communication path delays for the entire chip.

5.1 Chip Area Computation

The hardware design can be partitioned into three distinct parts, input circuit, computing structure and output circuit. The layout of processing elements and latches in the structure are regularly connected and support only local communication (i.e., nearest neighbors). Furthermore, the fundamental building blocks of the MAC's and DC's are full-adder arrays; also locally connected. Therefore, it is possible and practical to model basic units, such as latches and full-adders, as functional modules and then tesselate these modules into larger structures such as MAC's and DC's. One can then, tesselate these larger structures, which are now processing elements, and latches into the conglomerate computing structure.

Designs of the three different input and output circuits are basically composed of inverters and pass transistors. The fundamental building blocks are chains of pass transistors and inverters, SR flip-flops, one-bit shift registers, buffers and latches. The I/O designs are again

simple and regular block structures. Hence, it is realistic to estimate the area geometries of the I/O circuits by calculating individual areas and tesselating the fundamental building block into the desired configuration. Of course, during the tesselation, local communication geometry must be included. Now, both the computing structure and I/O circuits require that relatively few specific modules be laid out by hand and quantified for initial "seeding" of the circuit model. These modules include a full-adder, one-bit shift register, SR flip-flop, buffer and latch, as well as a pass transistor and inverter chain.

The most primitive component of each module's layout is the basic NMOS inverter. This device has a threshold voltage, V_{th}, of approximately 0.2 VDD and an inverter threshhold, V_{inv}, about midway between V_{DD} and GND. Under these conditions the inverter ratio, K, will be 4:1 as determined by the equation,

$$V_{inv} \approx \frac{V_{DD}}{\sqrt{Z_{pu}/Z_{pd}}} \qquad (5\text{-}2)$$

for $V_{th} << V_{DD}$ [9].

Based on this 4:1 ratio for the basic inverter, the pull-up to pull-down ratio of other primitive gates, such as an n-input NAND, can also be estimated. An important class of circuits used in our designs is composed of two inverters with one or more pass transistors in between. For the case of two inverters coupled by one pass transistor, the pull-up to pull-down ratio of the second inverter is

required to be 8:1 due to the threshold voltage of the pass transistor [9].

Besides knowing these ratio values, parameters giving linewidth and space dimensions of the layout geometry must be incorporated. Strict adherence to the layout design rules of Mead and Conway, however, was avoided as it has been pointed out that this method, while producing very rapid layouts, most often results in sub-optimal use of the available chip area [30]. Therefore, other area estimations were made using more practical rules of thumb [29]. Again, the resolved building block real estates are divided by minimum lithographic linewidth of given technology so as to maintain the dimensionless design rule values for reflecting the predicted decrease in λ as I. C. capabilities advance towards VLSI.

5.2 Propagation Delay Computation

The localized connectivity of the computing array structure and input/output circuitry allows straightforward computation of delay parameters. Quantification of modular delay parameters involved calculation of the active gates' propagation time with consideration of loading (gate fan-out) and internal communication path delays. Utilizing the revised model, the active gate delay time is based on T, the discharge time of a basic inverter with one fan-out to an identical gate. It is assumed that T is linearly dependent on λ and when $\lambda = 3\ \mu m$, T = 0.6 ns [9].

Consequently, if a basic inverter couples with m basic inverters at its output point, the rising time delay for the active inverter gate is mKT or 4mT.

In polysilicon gate NMOS I.C. technology, the communication paths are mostly metal lines and diffusion lines. Since the unit length delay of a metal line is much smaller than that of a diffusion line (approximately 1:1000) [9], the metal line delay is considered negligible. But, it is impractical to measure diffusion line length piece by piece through every module. Therefore, an upper bound was used by taking the total diffusion line length as the length of the longest side of a given module. Then, the delay time is calculated as a function of this line length squared. Using the fact that transit time is 100 ns for a 10 millimeter length line (when $\lambda = 3\mu m$) [9], communication path delays were obtained. Interestingly, it is not necessary to consider intermodule communication line length or delay since modules are considered to tile the plane and, therefore, abut one another.

Finally, the total module delay is the sum of active gate delays and communication path delays. Calculating total chip delay, however, is much simpler once the maximum segment delay is known. This is because the maximum segment delay is the limiting factor in the determination of clock speed. And since the overall chip design is pipeline in nature, the total propagation delay is determined by the maximum segment delay times the number of operands, plus the set-up time.

5.3 Significant Module Data

Chip area and propagation delay computational methods have been discussed in Sections 5.1 and 5.2. Table 5.1 presents results of these calculations for each fundamental building block.

Table 5.1 Significant module data

Module Class	Area (x λ^2)	Active Gate Delay (x T)
Full-Adder	65 x 54	28
One-bit Shift Register	23 x 17	18
Buffer	23 x 17	2
SR Flip-Flop	29 x 19	12
Latch	23 x 17	13
2-Input NAND Gate	19 x 5	8
Pass Transistor	3 x 3	1

CHAPTER VI

SIMULATION RESULTS AND EVALUATION

Chapter V described the simulation models which were used to obtain parameters of chip area and propagation delay. In this chapter, I evaluate the design performance based on these models. The goals of this chapter are two-fold; first, to evaluate the area and time versus matrix bandwidth trade-offs of the three different I/O strategies, and secondly, to evaluate and discuss the entire chip area geometry and propagation delay versus matrix bandwidth by combining the computing structure and the I/O strategy selection.

It has been predicted that the pattern resolution limitation of optical lithography is about 0.5 microns [31]. Recently, however, many I.C. designers have been exploring new and quite promising techniques including electron-beam and x-ray lithography. With the electron-beam method, for example, future linewidths of 0.125 microns have been predicted [32]. For these reasons, 0.8 and 0.5 micron linewidths have been chosen as realistic goals of mid and late 1980's VLSI capability.

Another simulation parameter, word size, was selected to be 8, 16 and 24 bits per word. There are two major reasons for selecting word size in this range. First, this special purpose computing chip will ideally be used mainly as a coprocessor along with a microprocessor-based

microcomputer system. With these word size selections, the simulation results will match the sizes of these systems. Secondly, based on the obtained simulation results, word sizes of more than 24 bits severely decrease the number of processing elements which can be fabricated on a single chip. A small number of processing elements imply that only networks of small bandwidth ($\leq$ 2) can be considered. Bandwidths in this range are considered insignificant for solving linear equation systems of practical size. The limitation is because the maximum chip size with current I.C. technology is about 1 centimeter squared.

In addition to the parameter selections of linewidth and word size, an assumption is made that the matrix dimension-bandwidth relationship is B = 0.1 N. The data and figures presented in the following sections will ultimately lead to a conclusion regarding the best area and time versus problem size trade-offs in the design of the I/O structures.

6.1 Comparison of I/O Strategies

Three different input/output strategies and corresponding circuit designs were explained in Chapter IV. In this section two input port (INPORT) and output port (OUTPORT) graphs comparing chip edge size (in cm) and propagation delay (in μ-sec) versus matrix bandwidth for the chosen linewidth and word size are shown.

6.1.1 INPORT Strategy Selection

Figure 6.1 graphically shows the INPORT edge size requirements of the three algorithms versus various matrix bandwidths at λ = 0.8 microns and 16 bits per word. The required INPORT chip area is the INPORT edge size squared. The graph contains three curves representing the algorithms: 1) SCS, 2) binary tree design, and 3) data controlled strategy. Upon examining the curves in Figure 6.1, it is obvious that algorithm #3, the data controlled strategy, occupies the least area. The small edge difference between algorithm #1 and #3 is due to the fact that algorithm #1 requires N one-bit shift registers and buffers for selecting input channels, whereas algorithm #3 does not (see Figures 4.1 and 4.6). Also, the big edge difference of algorithm #2 over algorithms #1 and #3 is because its many required SR flip-flops occupy an enormous amount of area. It is also noted that the reason for algorithm #2's step curve is due to the regular (power of 2) binary tree structure. Yet from this figure, it is plain that even an irregular binary tree structure would be the most area-consuming.

In the determination of the best input structure, one must also consider Figure 6.2, which illustrates the INPORT delay time versus matrix bandwidth relationship. Again, it is obvious that algorithm #3 requires the least time among these three choices. It is concluded then that algorithm #3, the data controlled strategy, is the best input circuit algorithm of the three, as it requires the least amount of both chip area and delay time.

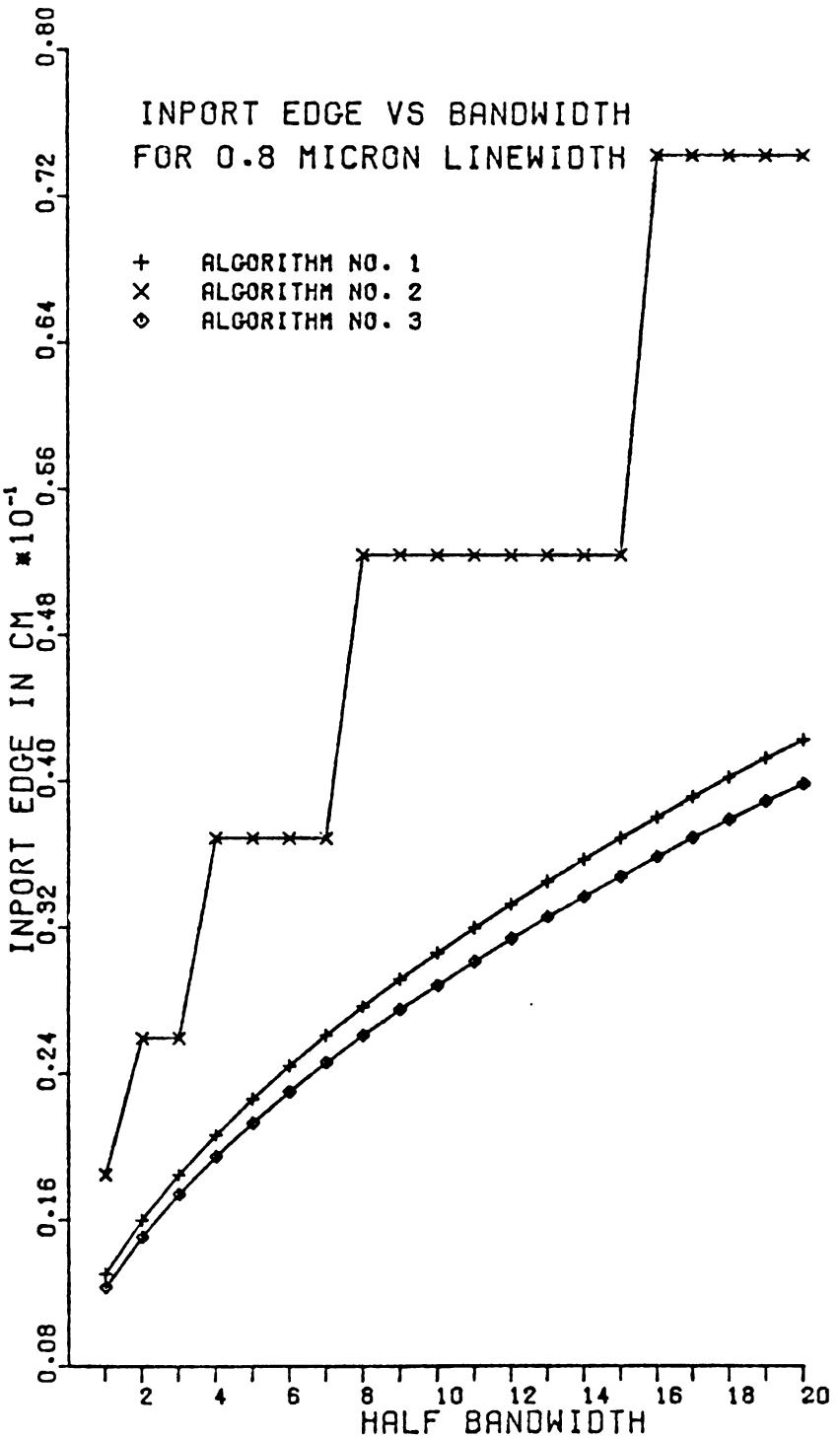

Figure 6.1 INPORT edge size versus matrix bandwidth for 16 bits per word at λ = 0.8 μm.

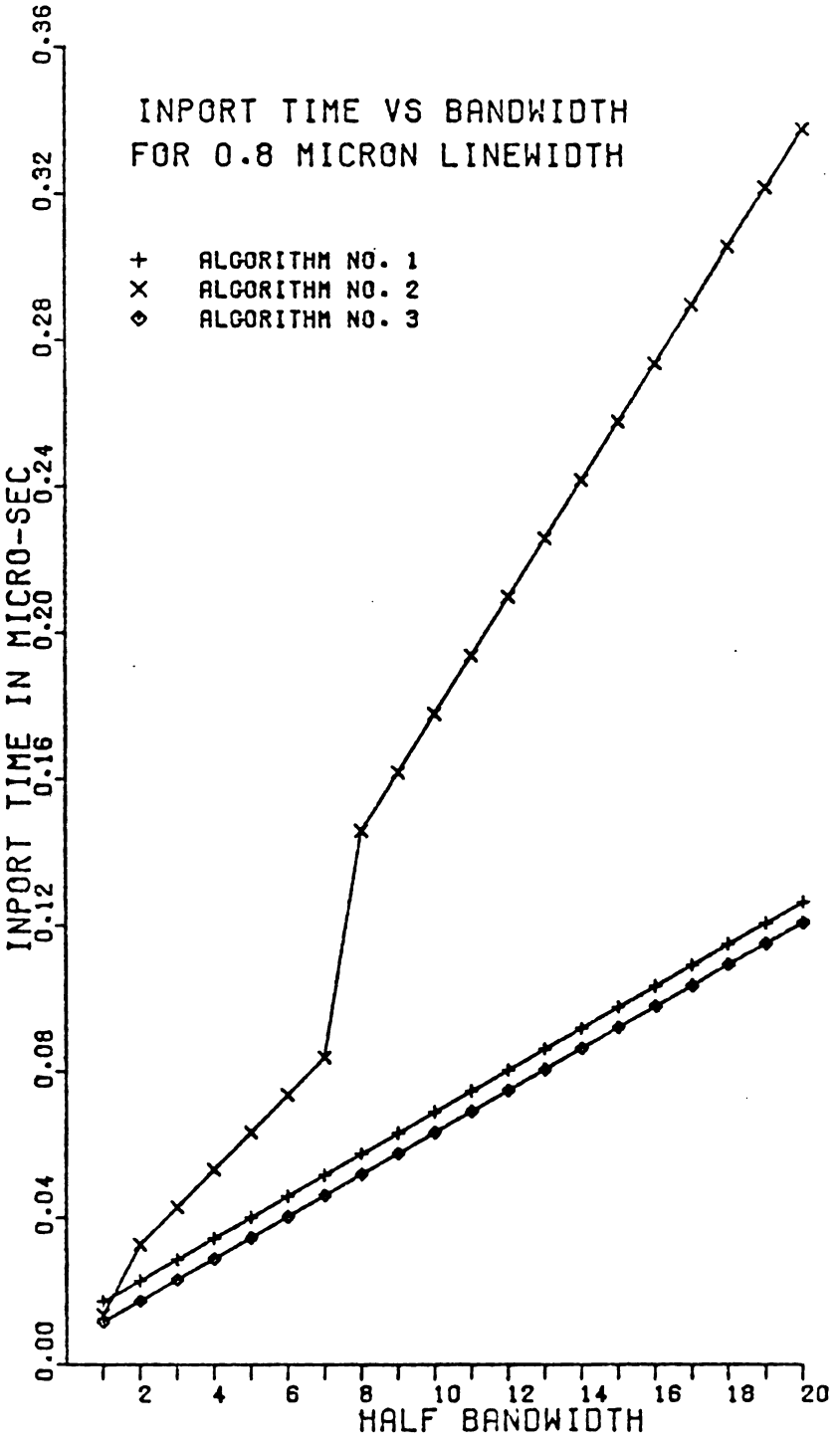

Figure 6.2 INPORT delay time versus matrix bandwidth for 16 bits per word at λ = 0.8 μm.

6.1.2 OUTPORT Strategy Selection

Figures 6.3 and 6.4 illustrate the OUTPORT edge size and delay time versus matrix bandwidth with parameters of 16 bits per word and 0.8 micron linewidth. The OUTPORT edge size of algorithm #2 is the smallest among the three (see Figure 6.3). Algorithm #1 requires the least propagation delay (see Figure 6.4).

From the discussion in Section 6.1.1, it was concluded that the input structure of algorithm #3, the data controlled strategy, dominates both in area and time properties. But, for the output structure, Figures 6.3 and 6.4 convey that algorithm #1 always requires less area geometry and delay time than algorithm #3. At this point, an obvious method for choosing the best output structure is to establish a complete data table of the entire chip delay and chip area. This will incorporate the computing structure, INPORT algorithm #3 and choices of OUTPORT algorithms #1 and #2. If the INPORT and OUTPORT algorithm pair, #3 and #1, is superior to pair #3 and #2, we can conclude that OUTPORT algorithm #1 is the best among the three. Otherwise, OUTPORT algorithms #2 and #3 must be compared in a similar manner.

Tables 6.1 and 6.2 list bandwidth values (BW), INPORT time (ITIME), OUTPORT time (OTIME), computing time (MACTIME), OUTPORT area (OAREA) and total chip time and area. The corresponding data of INPORT and OUTPORT algorithms #3 and #1 respectively, with an 8 bit word size

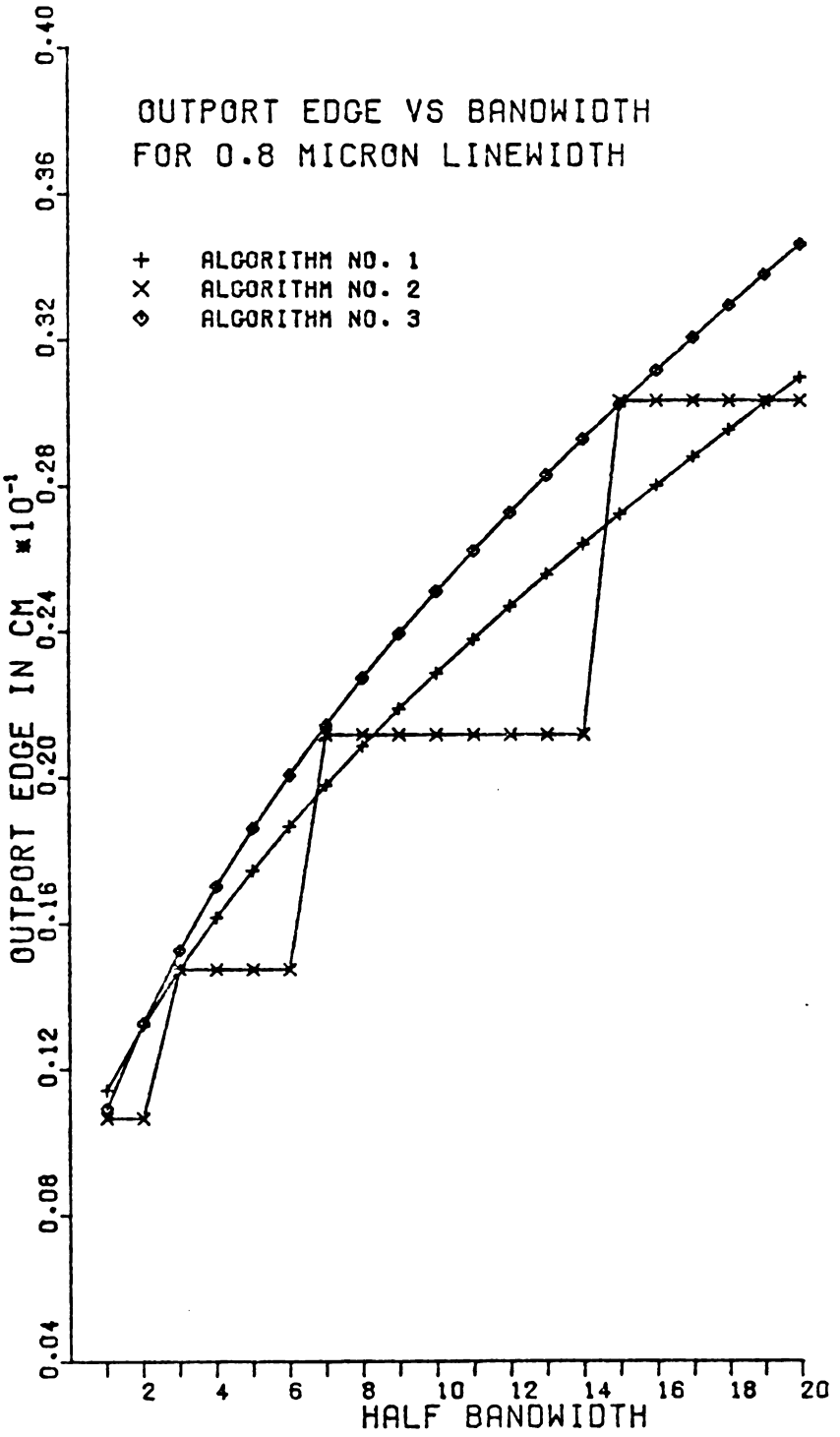

Figure 6.3 OUTPORT edge size versus matrix bandwidth for 16 bits per word at $\lambda = 0.8$ μm.

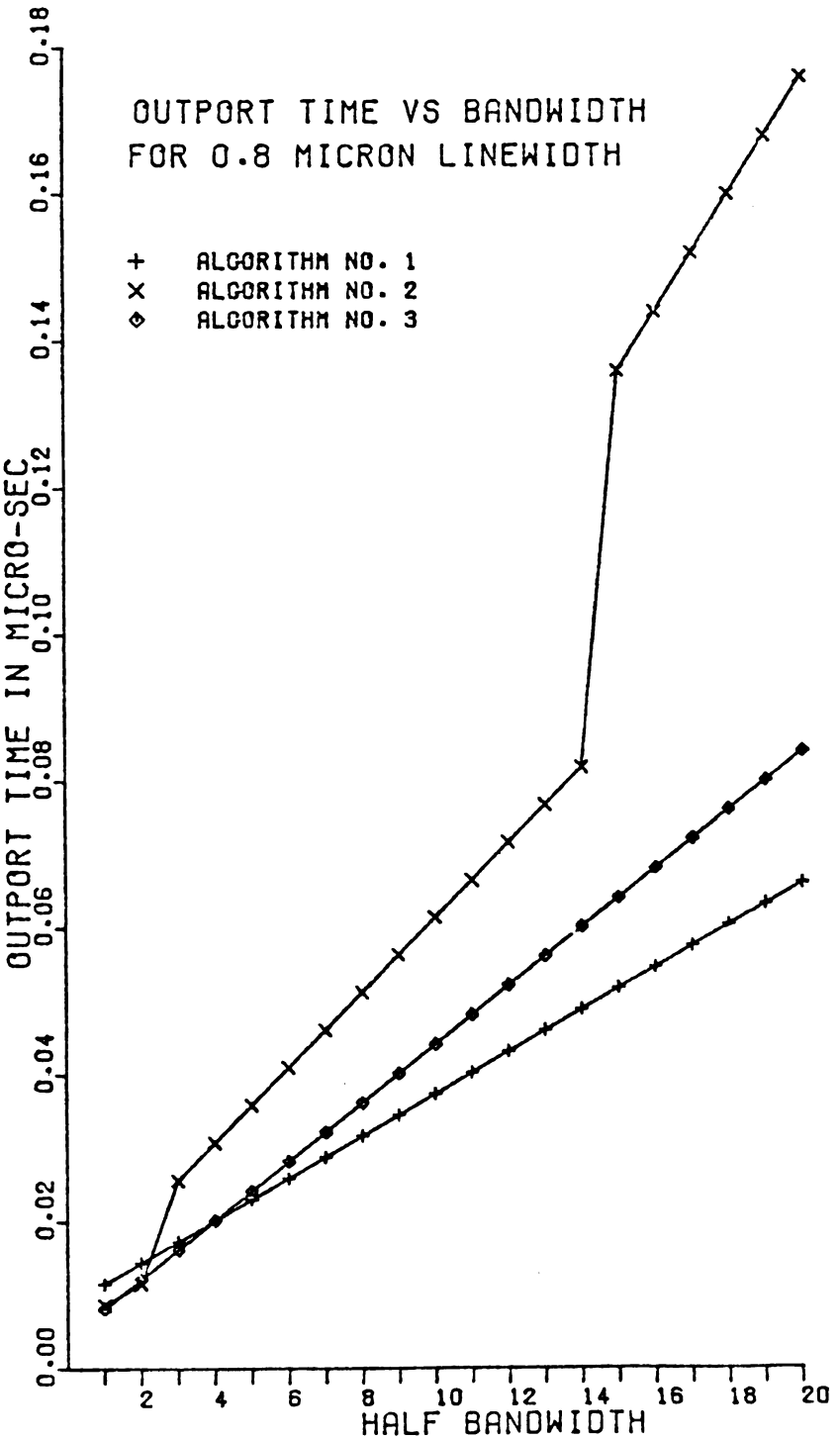

Figure 6.4 OUTPORT delay time versus matrix bandwidth for 16 bits per word at λ = 0.8 μm.

Table 6.1a Total chip area and time results for algorithm #3 (INPORT) and #1 (OUTPORT) for 8 bits per word at λ = 0.8 microns.

BW	ITIME (ns)	OTIME (ns)	MACTIME (ns)	TOTALTIME (ns)	OAREA (cm^2)	TOTALAREA (cm^2)
12	74.9	41.9	78.72	21611.7	0.00034	0.36216
13	80.6	44.8	78.72	23950.9	0.00036	0.41413
14	86.4	47.7	78.72	27565.0	0.00039	0.46946
15	92.2	50.6	78.72	31432.5	0.00041	0.52815
16	97.9	53.4	78.72	35553.4	0.00043	0.59019
22	132.5	70.7	78.72	65601.4	0.00058	1.03288
23	138.2	73.6	78.72	71496.5	0.00060	1.11840
46	270.7	139.8	78.72	277031.8	0.00116	4.01134
47	276.5	142.7	78.72	289009.4	0.00118	4.17738

Table 6.1b Total chip area and time results for algorithm #3 (INPORT) and #2 (OUTPORT) for 8 bits per word at λ = 0.8 microns.

BW	ITIME (ns)	OTIME (ns)	MACTIME (ns)	TOTALTIME (ns)	OAREA (cm^2)	TOTALAREA (cm^2)
12	74.9	71.7	78.72	21641.4	0.00022	0.36204
13	80.6	76.8	78.72	23982.9	0.00022	0.41440
14	86.4	81.9	78.72	27599.2	0.00022	0.46930
15	92.2	136.0	78.72	46072.8	0.00046	0.52820
16	97.9	144.0	78.72	51956.3	0.00046	0.59021
22	132.5	192.0	78.72	94649.4	0.00046	1.03276
23	138.2	200.0	78.72	102997.0	0.00046	1.11826
46	270.7	384.0	78.72	392141.9	0.00093	4.01111
47	276.5	392.0	78.72	408937.4	0.00093	4.17713

Table 6.2a Total chip area and time results for algorithm #3 (INPORT) and #1 (OUTPORT) for 16 bits per word at λ = 0.8 microns.

BW	ITIME (ns)	OTIME (ns)	MACTIME (ns)	TOTALTIME (ns)	OAREA (cm^2)	TOTALAREA (cm^2)
16	97.9	54.7	150.40	54285.1	0.00078	2.45532
17	103.7	57.6	150.40	57648.3	0.00083	2.71757
18	109.4	60.5	150.40	61011.5	0.00087	2.99262
21	126.7	69.1	150.40	71101.1	0.00100	3.89457
22	132.5	72.0	150.40	74464.3	0.00105	4.22082

Table 6.2b Total chip area and time results for algorithm #3 (INPORT) and #2 (OUTPORT) for 16 bits per word at λ = 0.8 microns.

BW	ITIME (ns)	OTIME (ns)	MACTIME (ns)	TOTALTIME (ns)	OAREA (cm^2)	TOTALAREA (cm^2)
16	97.9	144.0	150.40	54374.4	0.00092	2.45546
17	103.7	152.0	150.40	58345.9	0.00092	2.71766
18	109.4	160.0	150.40	64941.4	0.00092	2.99267
21	126.7	184.0	150.40	86840.0	0.00092	3.89449
22	132.5	192.0	150.40	94843.0	0.00092	4.22070

and $\lambda = 0.8\ \mu m$, are shown in Table 6.1a. The data for algorithm #3 (INPORT) and #2 (OUTPORT), are shown in Table 6.1b. Again, the same INPORT and OUTPORT algorithm pairs, with a 16 bit word size, are presented in Table 6.2a and 6.2b. According to these four tables, it was concluded that the utilization of OUTPORT algorithm #2 causes a serious output bottleneck as the matrix bandwidth increases. Thus, the entire chip propagation delay (TOTALTIME) is forced to increase tremendously. Although it is noted that the OUTPORT area (OAREA) of algorithm #2 is less than that of algorithm #1, this small area difference is insignificant when compared to the total area (TOTALAREA) in both cases.

Thus, it is concluded that OUTPORT algorithm #1, the SCS strategy, is "better" than algorithm #2, the binary tree design. In conclusion, the best INPORT and OUTPORT algorithms for our special purpose computing structure are the data controlled and SCS strategies, respectively.

6.2 Entire Chip Simulation Results

The main objective of this section, and more importantly, one of the ultimate goals of this work, is to quantify and discuss the entire chip area geometry and propagation delay in relation to problem size. The input and output strategies used are the data controlled and SCS algorithms, respectively. Graphs showing entire chip edge sizes and propagation delays, with various matrix bandwidths and 0.8 micron linewidth, are illustrated in

Figures 6.5 and 6.6. Currently, the standard I.C. chip size, due to reasons of productivity yield, is commonly limited to about 1 cm^2. To relate the results of these graphs to realistic network sizes, consider, for example, 16 and 24 bit word sizes. Under these conditions the matrix bandwidths are limited to 9 elements for 16 bits per word and 5 elements for 24 bits per word. To extrapolate this result to a realistic problem size, it has been discussed in Chapters III and V that matrix bandwidth is assumed to be about 10% of the original network matrix dimension. Thus, the results indicate that for this class of sparse matrix problems network dimensions of 90 nodes for a 16 bit machine and 50 nodes for a 24 bit machine are possible. From Figure 6.6, the corresponding triangulation times, or total chip propagation delays, for the 16 bit and 24 bit word size examples are 30.7 and 25.6 micro-secs. each.

Next, consider a late 1980's technology linewidth of 0.5 microns. From Figure 6.7 it is noted that 16 and 9 element bandwidths, with parameters of 16 and 24 bits per word respectively, can be fabricated on a 1 cm^2 silicon chip. Therefore, possible network dimensions become 160 and 90 nodes, respectively. The corresponding total chip delays are 33.9 and 28.3 micro-secs. in each case as shown in Figure 6.8.

Previously, it was questionable as to whether this type of systolic array architecture would cause a serious I/O bottleneck due to the number of simultaneous operands

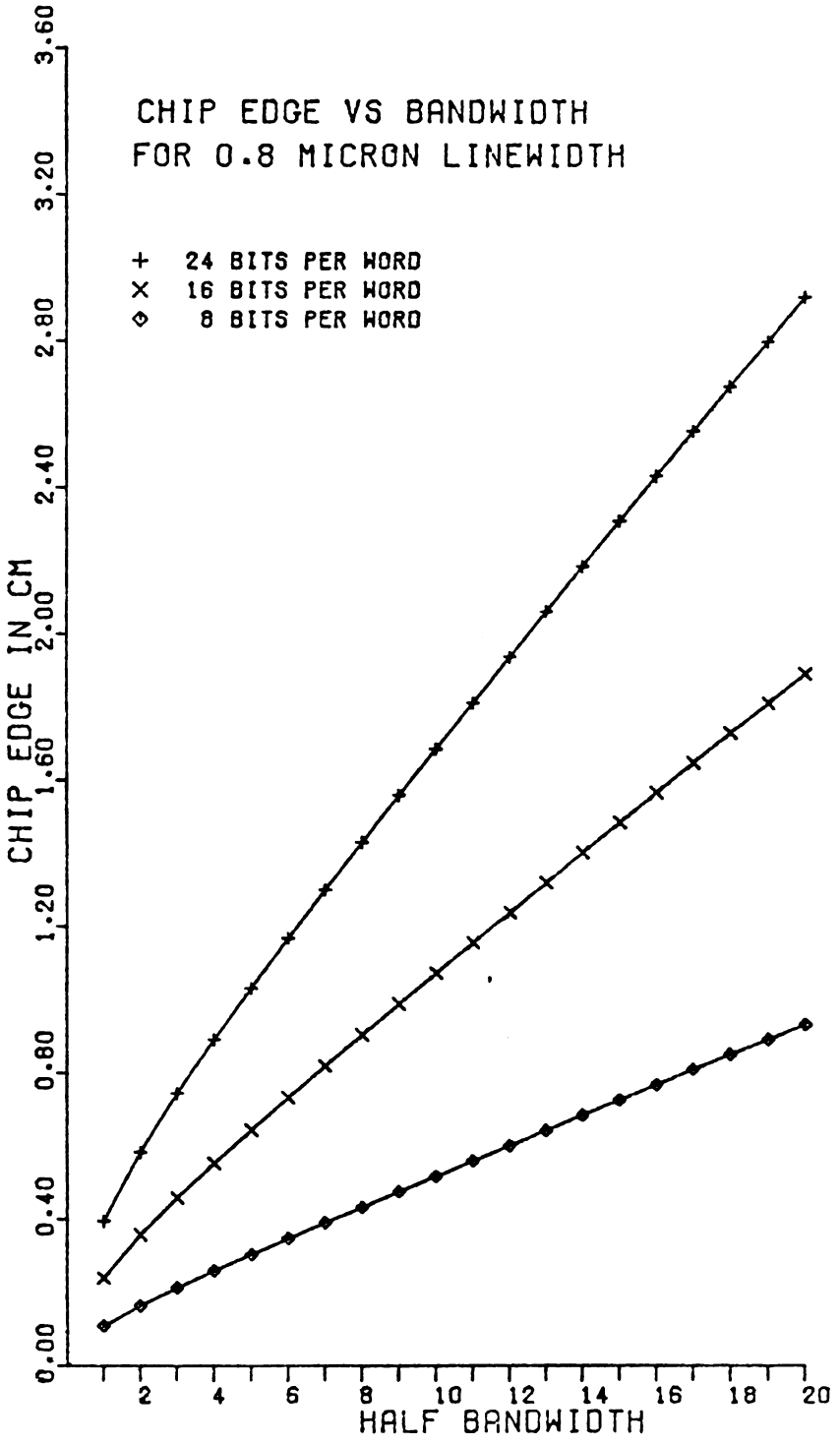

Figure 6.5 Entire chip edge size versus matrix bandwidth for 8, 16 and 24 bits per word at $\lambda = 0.8$ μm.

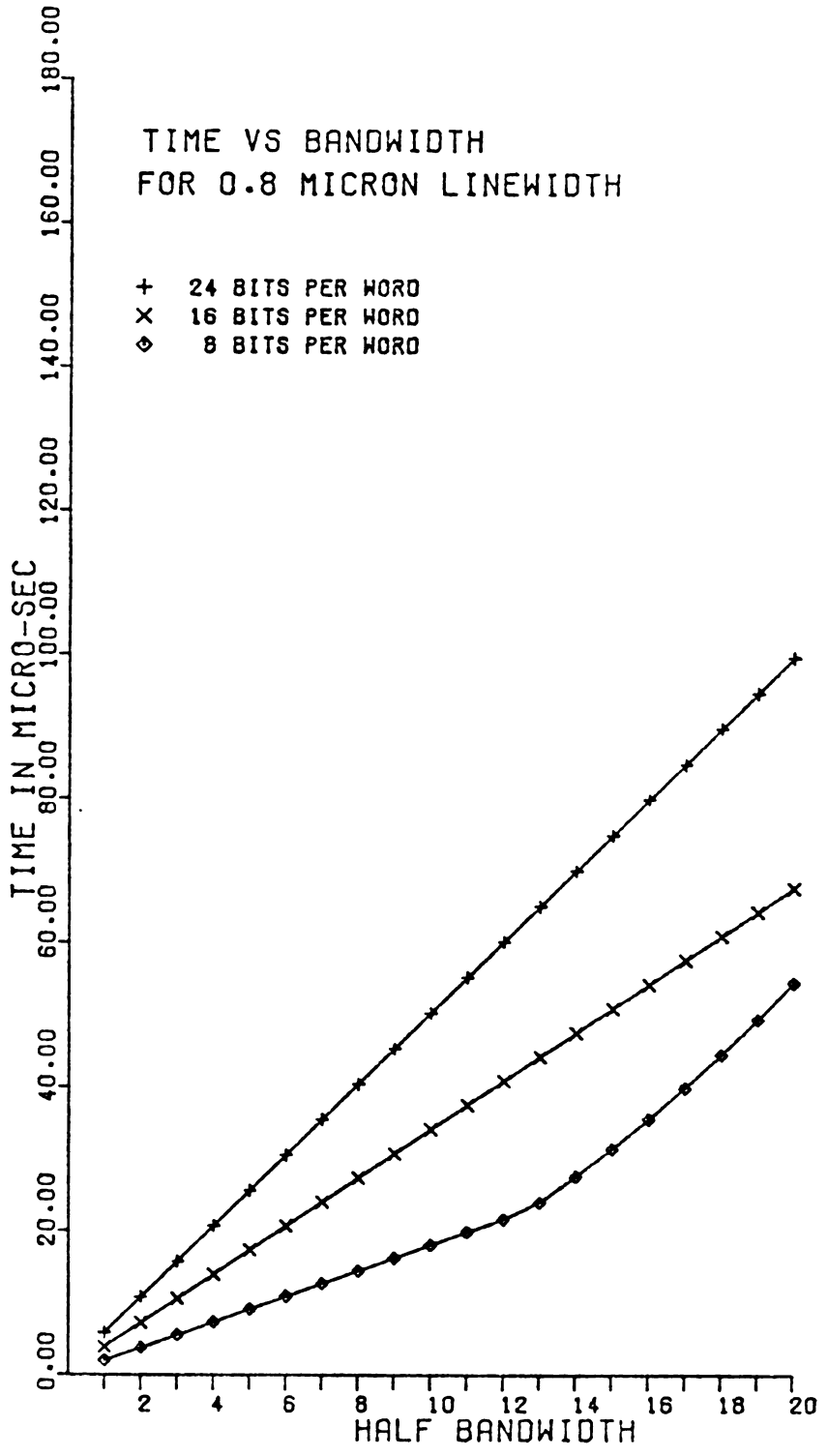

Figure 6.6 Entire chip propagation delay versus matrix bandwidth for 8, 16 and 24 bits per word at λ = 0.8 μm.

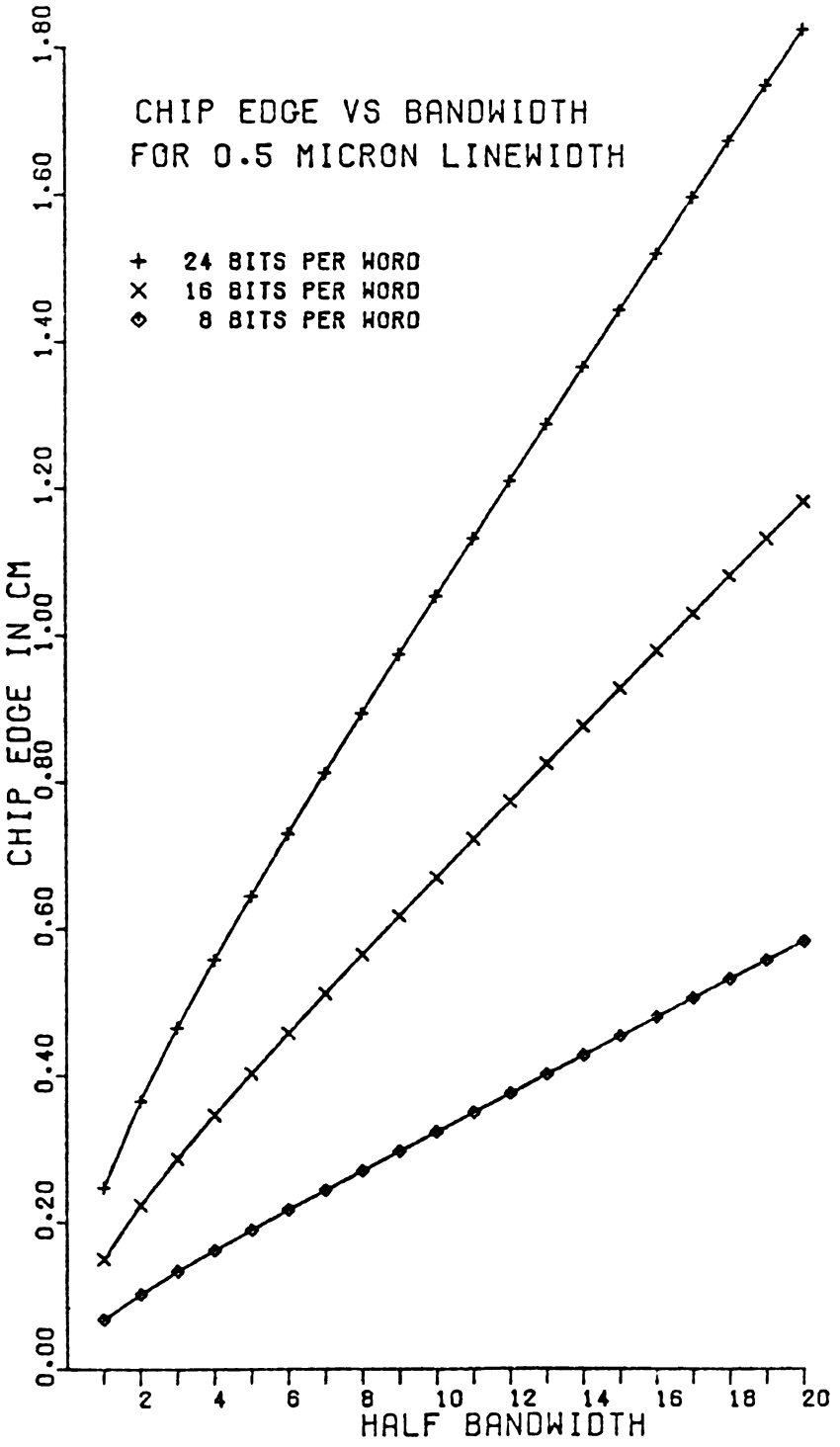

Figure 6.7 Entire chip edge size versus matrix bandwidth for 8, 16 and 24 bits per word at λ = 0.5 μm.

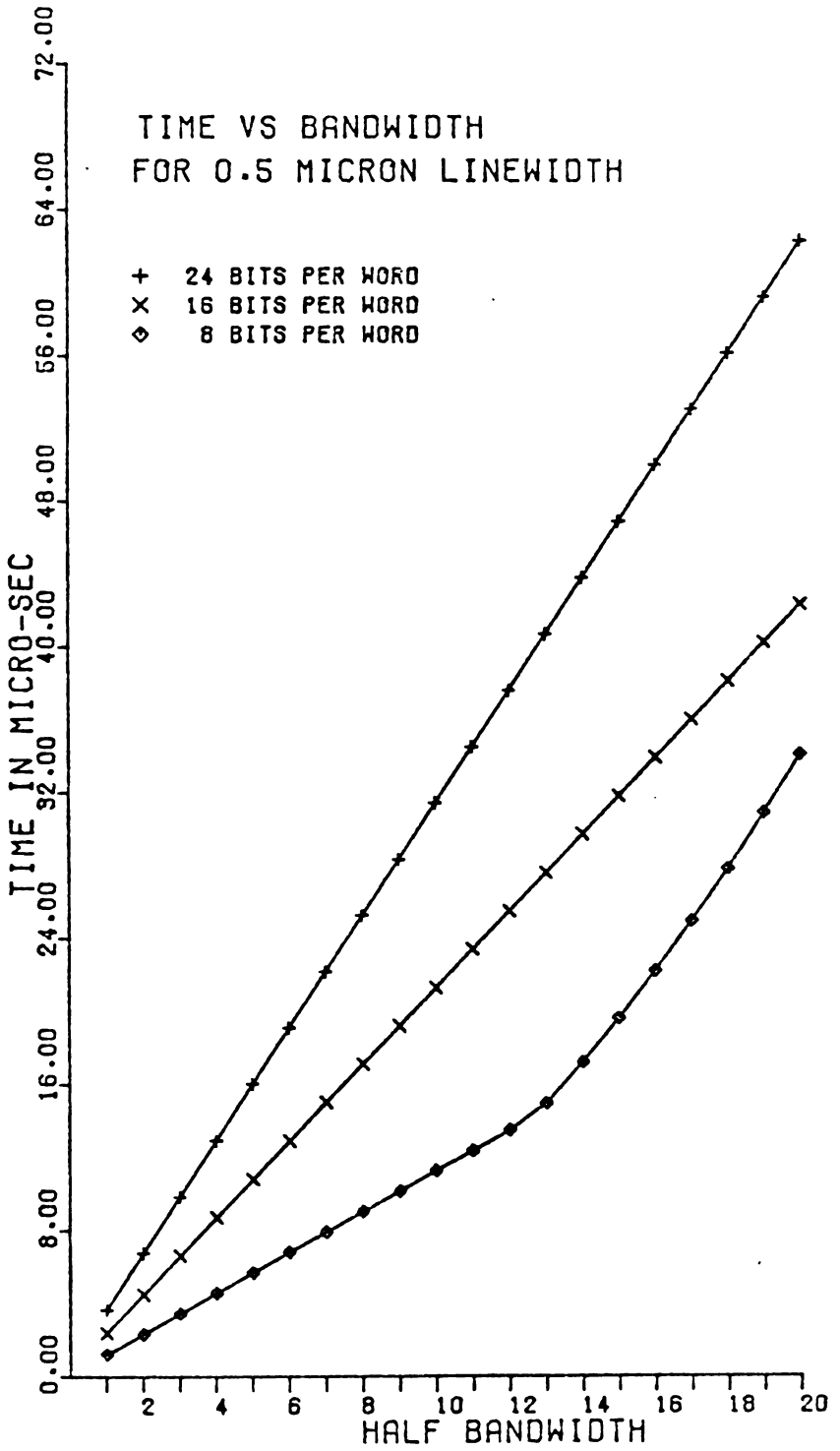

Figure 6.8 Entire chip propagation delay versus matrix bandwidth for 8, 16 and 24 bits per word at λ = 0.5 μm.

required by computing structure [33]. In the 16 and 24 data bit computing structure designs, the maximum segment delay of the pipelined structure for the entire chip is the MAC time, at least for bandwidths less than or equal to 20 elements. Therefore, no I/O bottlenecking is anticipated. Also note that in both the 16 and 24 bit word size designs, the corresponding total chip edge sizes for matrix bandwidths of 20 elements are mostly far beyond the limits of our current I.C. technology (see Figures 6.5 and 6.7).

It is interesting to note here that for the 8 bit designs, the breaks in the slope of the delay curves (see Figures 6.6 and 6.8) are, in fact, due to the suspected input bottleneck. For a more detailed illustration, the INPORT time (IT), OUTPORT time (OT), MAC time (MACT) and maximum segment time (SEGT), with bandwidths up to 20 elements for 8 and 16 bit machines are shown in Table 6.3. The break in the slope is because, for small word sizes, MAC time is proportionally small as it is a function of word size. INPORT time, however, is a function of bandwidth. Thus, for small word sizes and large bandwidths, I/O time will dominate as the maximum segment delay time in the pipeline. But, a major conclusion of this work is that for reasonable word sizes (between 16 and 24 bits per word) and reasonable matrix bandwidths (at least $\leq$ 20 elements) an I/O bottleneck will not occur.

These results are quite encouraging and show great promise for throughput enhancement in a large number of

Table 6.3 Simulation time results for 8 and 16 bit words at λ = 0.8 microns.

	(8 bits)					(16 bits)			
BW	IT (ns)	OT (ns)	MACT (ns)	SEGT (ns)	BW	IT (ns)	OT (ns)	MACT (ns)	SEGT (ns)
1	11.5	10.2	78.2	78.2	1	11.5	11.5	150.40	150.40
2	17.3	13.1	78.2	78.2	2	17.3	14.4	150.40	150.40
3	23.0	16.0	78.2	78.2	3	23.0	17.3	150.40	150.40
4	28.8	18.9	78.2	78.2	4	28.8	20.2	150.40	150.40
5	34.6	21.8	78.2	78.2	5	34.6	23.0	150.40	150.40
6	40.3	24.6	78.2	78.2	6	40.3	25.9	150.40	150.40
7	46.1	27.5	78.2	78.2	7	46.1	28.8	150.40	150.40
8	51.8	30.4	78.2	78.2	8	51.8	31.7	150.40	150.40
9	57.6	33.3	78.2	78.2	9	57.6	34.6	150.40	150.40
10	63.4	36.2	78.2	78.2	10	63.4	37.4	150.40	150.40
11	69.1	39.0	78.2	78.2	11	69.1	40.3	150.40	150.40
12	74.9	41.9	78.2	78.2	12	74.9	43.2	150.40	150.40
13	80.6	44.8	78.2	80.6	13	80.6	46.1	150.40	150.40
14	86.4	47.7	78.2	86.4	14	86.4	49.0	150.40	150.40
15	92.2	50.6	78.2	92.2	15	92.2	51.8	150.40	150.40
16	97.6	53.4	78.2	97.6	16	97.6	54.7	150.40	150.40
17	103.7	56.3	78.2	103.7	17	103.7	57.6	150.40	150.40
18	109.4	59.2	78.2	109.4	18	109.4	60.5	150.40	150.40
19	115.2	62.1	78.2	115.2	19	115.2	63.4	150.40	150.40
20	121.0	65.0	78.2	121.0	20	121.0	66.2	150.40	150.40

scientific and engineering problems. Integrated circuit technologists have predicted that the actual hardware implementation of such a class of high performance algorithms will soon become a reality. These results should benefit and promote further research in this area.

CHAPTER VII

CONCLUSIONS

7.1 Summary

Advances in improving the throughput speed for solving large scale, sparse matrices have been a topic of research for many decades. The fundamental method for direct solution is Gaussian elimination. To improve on this classical (serial) computer technique, Markowitz ordering has been traditionally used to provide a reduced fill-in during the triangulation of a nonstructured coefficient matrix [34]. In recent years, the utilization of parallel and pipelined structures of an existing new generation of array processors have become popular in this area [8, 35, 36]. However, the solution time provided by even the best current approach is still far away from the real-time computational requirements of many scientific and engineering problems.

Cellular-array architectures have emerged and shown great promise in performing highly concurrent numerical algorithms, such as L-U decomposition, matrix inversion, etc. Furthermore, recent improvements in I.C. fabrication techniques have enabled computer architects to consider the design of dense VLSI integrated systems on a single chip. A new class of computing architectures, built of simple inner product step processors, and possessing local communication wires and highly regular elements, appear to be a prime candidate for direct hardware implementation on a VLSI chip.

The high speed, low cost advantages of VLSI techniques should prove to be in the mainstream of next generation special purpose computer design spawning enthusiastic research in this area.

The purpose of this research was first to develop a modified systolic array algorithm to triangulate the augmented coefficient matrix $\{\underline{A} \mid \underline{b}\}$ of large, band form linear equation systems, $\underline{A} \cdot \underline{x} = \underline{b}$. The new algorithm will broaden the application area of VLSI systolic array algorithms to more practical engineering problems by considering such large scale systems.

Chapter III of this dissertation described and illustrated a computing structure design to triangulate a band form augmented matrix, with matrix dimension N and half bandwidth B, in O(N) time and using $O(B^2)$ PE's. The number of PE's required by this new VLSI algorithm is greatly reduced compared to previously published algorithms of this type which have required $O(N^2)$ PE's. This structure is most applicable to large scale, diagonally dominant, highly sparse matrix systems whose nonzero elements naturally occur in band form or can be permuted, a priori, to minimum band form.

In the last section of Chapter III, crucial inner details of the processing elements in the computing structure were explored. Various arithmetic algorithms were evaluated and a modified Baugh-Wooley algorithm was determined to be best for the majority PE, the MAC. Also, a suitable

DC algorithm was designed. The upper bound on the maximum segment delay of the systolic pipelined structure was the MAC time. Dynamic latches were selected for temporary storage of the data operands because of their small propagation delay and area geometry as compared to static latches. Moreover, dynamic latches provide for better field testability.

I then focused on developing the best I/O strategy for this computing structure. The main objective here was minimization of any possible I/O bottleneck. Three high speed input and output circuits were designed and evaluated in Chapter IV. They were the SCS, the binary tree and the data controlled algorithms.

Chip area and propagation delay computation models based on transistor level NMOS technology were presented in Chapter V. These two models provided a simple method of calculating area and time parameters among the candidate building block modules of the chip. The simulation parameters also enabled tracking projected I.C. fabrication technology so that predictions of future capabilities could be made.

In Chapter VI, I/O area and time trade-off results were compared utilizing the simulation models. The best INPORT strategy was determined to be the data controlled algorithm. The best OUTPORT strategy was found to be the SCS algorithm. Furthermore, for reasonable operand word sizes ($\geq$ 16 bits) and chip area (about 1 cm^2), bottlenecking of I/O coefficients was not observed.

The entire chip simulation results showed that for a 1 cm^2 chip size and 0.8 micron linewidth, the matrix bandwidth is limited to 9 elements for 16 bits per word and 5 elements for 24 bits per word. Under the assumption of 10% matrix bandwidth-dimension relationship, a single VLSI chip of the type presented can operate upon a 90-by-90 and 50-by-50 banded matrix $\underline{A}$ for 16 and 24 bits per word, respectively. The propagation delay for the two cases is 30.7 and 25.6 micro-secs., respectively. Furthermore, at the same chip size and a 0.5 micron linewidth, banded matrix dimensions of 160-by-160 and 90-by-90 elements can be triangulated in 33.9 and 28.3 micro-secs. for 16 and 24 bits per word, respectively.

The overall results were very encouraging. The initial work, particularly the PE design evaluation and comprehensive simulation, have helped in understanding the strengths and weaknesses of special purpose VLSI algorithms for large scale scientific and engineering problems.

7.2 Future Research

The VLSI single chip systolic array architecture under investigation here presented a new approach for triangulating practical, large scale, linear equation systems. The utilization of simulation models, based on NMOS technology for quantification of entire chip area geometry and propagation delay, also provided significant results for evaluating the feasibility of VLSI algorithms in this problem area.

Yet, several additional topics are worth future study.

1) One of the main problems for VLSI NMOS technology is power dissipation. NMOS typically requires much more power than CMOS. Future utilization of CMOS technology will provide the needed reduction in power dissipation. The speed and density of the entire chip will also be enhanced by the incorporation of CMOS circuits.

2) Our chip simulation results showed that I/O bottlenecking was not serious for large operand word size and small matrix bandwidth. This is because PE computation time dominated in the pipeline structure. Now, as I.C. technology advances, chip size will increase and resolution will decrease. It will still be many years, however, before we are able to implement a large number of PE's (> 100 or so) on a single chip. Therefore, multi-chip networks of processing elements with large word sizes should be looked into as a near term approach. Problems to be considered here include algorithm decomposition and, most importantly, inter-chip communication.

3) Another crucial step will be the design of true floating-point processing elements for our computing structure. This will make the chip design more universal as it will ease restrictions on problem compatibility.

BIBLIOGRAPHY

1. Kung, H. T. and Leiserson, C. E., "Systolic Arrays (for VLSI)," Sparse Matrix Proc., (I. S. Duff, el. al. editors), Society for Indust. and Appl. Math. (1979), pp. 256-282.

2. Cuthill, E. and McKee, J., "Reducing the Bandwidth of Sparse Symmetric Matrices," Proc. 24th National Conf. ACM, Brandon System Press, New Jersey (1969), pp. 157-172.

3. Gibbs, N. E., Poole, W. G. Jr., and Stockmeyer, P. K., "An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix," SIAM J. Numer. Anal., Vol. 13 (1976), pp. 236-250.

4. Zeleny, P. C., and Nagle, R. E., "Application of an Array Processor in Satellite Image Processing," AD-A056664/6ST, National Tech. Information Service, Springfield VA 22151 (June 1977).

5. Pottle, C., "The Use of an Attached Scientific ("Array") Processor to Speed Up Large Scale Load Flow Simulations," Proc. IEEE ICCC 80, Port Chester, NY (1980).

6. Woo, P. T., "Application of Array Processor to Sparse Elimination," SPE 5th Symposium on Reservoir Simulation, Denver (1979).

7. "Array Processor Does Shape Recognition," Electronic Design, Vol. 26, No. 18 (1978), p. 146.

8. Shanblatt, M. A., "Vectorized Elimination and Ordering Strategy for Load Flows on an Array Processor," Ph.D. Thesis, University of Pittsburgh (April 1980).

9. Mead, C. and Conway, L., Introduction to VLSI Systems, Addison-Wesley Pub. Co., Reading, Massachusetts (1980).

10. Rem, M. and Mead, C. A., "Cost and Performance of VLSI Computing Structures," IEEE J. of Solid State Circuits, Vol. Sc-14 (April 1979), pp. 455-462.

11. Foster, M. J. and Kung, H. T., "The Design of Special-Purpose VLSI Chips," Computer (January 1980), pp. 26-40.

12. Kung, H. T., "Let's Design Algorithms for VLSI Systems," Proc. Caltech. Conf. Very Large Scale Integration (January 1979), pp. 65-90.

13. Kung, H. T., "The Structure of Parallel Algorithms," in Advances in Computers, Vol. 19, Academic Press (1980).

14. Kung, H. T. and Leiserson, C. E., "Algorithms for VLSI Processor Array," Symposium on Sparse Matrix Computations, Knoxville (1978).

15. Preparata, F. P. and Vuillemin, J., "Optimal Integrated-Circuit Implementation of Triangular Matrix Inversion," Proc. of Int'l Conf. Parallel Processing, (August 1980) pp. 211-216.

16. Hwang, K. and Cheng, Y-H, "VLSI Computing Structures for Solving Large-Scale Linear Systems of Equations," Proc. 1980 Int'l Conf. on Parallel Processing (August 1980), pp. 217-230.

17. Hwang, K. and Cheng, Y-H, "Partitioned Algorithms and VLSI Structures for Large-Scale Matrix Computations," Proc. 5th Symposium on Computer Arithmetic (May 1981), pp. 222-232.

18. Oakes, M. F., "The Complete VLSI Design System," Proc. 16th Design Automation Conference (June 1979), pp. 452-460.

19. Eichelberger, E. B., "A Logic Design Structure for LSI Testability," Proc. 14th Design Automation Conference (June 1977), pp. 462-468.

20. Hwang, K., Computer Arithmetic, John Wiley and Sons, Inc. (1979).

21. Wallace, C. S., "A Suggestion for Parallel Multipliers," IEEE Trans. Electronic Computers, Vol. EC-13 (February 1964), pp. 14-17.

22. Pezaris, S. D., "A 40^{ns} 17-bit-by-bit Array Multiplier," IEEE Trans. Computers, Vol. C-20, No. 4 (April 1971), pp. 442-447.

23. Baugh, C. R. and Wooley, B. A., "A Two's Complement Parallel Array Multiplication Algorithm," IEEE Trans. Computers, Vol. C-22, No. 1-2 (December 1973), pp. 1045-1047.

24. Cappa, M., "Cellular Iterative Arrays for Multiplication and Division," M. S. Thesis, Dept. of Electrical Engineering, University of Toronto, Canada (October 1971).

25. Cappa, M. and Hamacher, V. C., "An Augmented Iterative Array for High-Speed Binary Division," IEEE Trans. on Computers, Vol. C-22 (February 1973), pp. 172-175.

26. Lyman, J., "Tape Automated Bonding Meets VLSI Challenge," Electronics (December 18, 1980), pp. 100-105.

27. Fairbairn, D. G., "VLSI: A New Frontier for System Designers," Computer (January 1982), pp. 87-96.

28. Taub, H. and Schilling, D., Digital Integrated Electronics, McGraw-Hill Inc. (1977).

29. Personal communication with Donnie K. Reinhard, Department of Electrical Engineering and Systems Science, Michigan State University (March 1982).

30. LaBrecque, M., "Faster Switches, Smaller Wires, Larger Chips," NSF MOSAIC (January/February 1982).

31. Keyes, R. W., "Physical Limits in Semiconductor Electronics," Science, Vol. 195 (March 1977), pp. 1230-1235.

32. Eidson, J. C., "Fast Electron-Beam Lithography," IEEE Spectrum (July 1981), pp. 24-28.

33. Chang, T. L., "Mixed Systolic Arrays: A Reconfigurable Multiprocessor Architecture," Ph.D. Thesis, Michigan State University (February 1982).

34. Markowitz, H. M., "The Elimination Form of the Inverse and its Application to Linear Programming," Management Science, Vol. 3 (1957), pp. 255-259.

35. Calahan, D. A., "Multi-Level Vectorized Sparse Solution of LSI Circuits," Proc. IEEE ICCC 80, Port Chester, New York (1980).

36. Dembart, B. and Neves, K. W., "Spare Triangular Factorization on Vector Computers," EPRI Special Report El-566-SR, pp. 57-102.

37. Siewiorek, D. P., Bell, C. G., and Newell A., Computer Structures: Principles and Examples, McGraw-Hill Inc. (1982).