This is to certify that the

thesis entitled

VLSI SYSTOLIC ARRAY FOR
MATRIX TRIANGULATION IN
LOAD FLOW ANALYSIS

presented by

Yu-Ying Jackson Leung

has been accepted towards fulfillment
of the requirements for

M. S. degree in Electrical Engr.

Major professor

Date 2/9/83

VLSI SYSTOLIC ARRAY

FOR MATRIX TRIANGULATION

IN LOAD FLOW ANALYSIS

By

Yu-Ying Jackson Leung

A THESIS

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

MASTER OF SCIENCE

Department of Electrical Engineering and
Systems Science

1983

ABSTRACT

VLSI SYSTOLIC ARRAY
FOR MATRIX TRIANGULATION
IN LOAD FLOW ANALYSIS

By

Yu-Ying Jackson Leung

The computational bottleneck incurred in power system load flow analysis is due to the cumbersome solution of a large, highly sparse set of linear equations. A VLSI systolic array structure for band matrix triangulation, utilizing concurrent Gaussian elimination, is applied to this problem to decrease the required triangulation time. Included is a design of an interface system connecting the systolic array structure to a host computer. A circuit simulation of the complete structure indicates a time savings of two orders of magnitude over traditional serial computer methods. These results lead to an assessment of the advantages and significance of this class of special purpose VLSI computing structures for throughput enhancement in load flow analysis.

To my mother and eldest brother

Mdm. Me Chit and Mr. Wai-Ying Tommy Leung

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 Dr. W. C. Hsu and the committee members, Dr. P. D. Fisher and Dr. R. A. Schlueter, for giving the valuable suggestions and comments in this work.

Finally, the author owes a special thanks to Miss Jasmine Lam for her emotional encouragement and support.

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

Electrical energy generation and delivery systems, generally known as power systems, are and will continue to be of fundamental importance to the technological and engineering community. For this reason, power systems, one of the biggest "systems" in the world, have been explored and studied since the late nineteenth century [1]. In particular, the load flow problem is the most basic and essential study because it provides information for the continuous evaluation of the current performance of the system and for analyzing the effectiveness of alternative plans for system expansion to meet increased load demand.

The load flow problem is the evaluation of power flows and voltages of a network for specified bus or terminal conditions. It provides the solution for the static operating conditions of the power transmission system. It is one of the most important of many current engineering problems that require a very rapid routine solution before system failures occur.

The Newton-Raphson method has been traditionally applied to solve the load flow problem by iteratively solving a set of non-linear equations expressing the specified real and reactive power in terms of bus voltages and phase angles. The limiting factor in problem throughput, from the standpoint of a traditional serial computer, is the time required to solve the large (often $n > 1000$) set of non-linear

equations via triangular factorization, a form of Gaussian elimination. This limitation is due to the fact that the number of computations is directly proportional to the square of the matrix dimension. Also, it is due to the cumbersome data transfers between the arithmetic-logic unit and memory (the Von Neumann bottleneck) as well as the constant packing and unpacking of operands during the elimination.

Recently, VLSI (Very Large Scale Integration) technology has emerged. An enormous number of computer algorithms and architectures have been proposed which show great promise in overcoming this computational bottleneck [2, 3, 4]. Utilizing this new VLSI technology, the purpose of this thesis is to study aspects of the design of a high-speed dedicated processor architecture, realizable in VLSI, tailored towards the load flow problem.

The dedicated processor structure is mainly composed of arrays of processing elements which can be implemented directly using low cost, high speed VLSI circuit technology either on a single chip, or perhaps in a modular fashion on a few chips. This VLSI computing array, based on concurrent Gaussian elimination for triangulating band form systems and composed of fixed-point processing elements, is to be simulated by a Fortran-coded program. The simulation will be tied into a proposed interfacing design between the VLSI processor and a host computer.

Special topics to be discussed include details of the interfacing system, the simulation and results pertaining to load flow throughput enhancement. Furthermore, a basic constraint of the number of fixed-point bits (word size) inside the computing array must be determined. This is because the word size of the operands has an

essential effect on load flow convergence. Too few bits will cause undue rounding error and lead to divergence of the overall Newton-Raphson formulation.

Another detail to be discussed is how to interconnect the processing elements within the array such that it can efficiently carry out a strict upper triangular factorization of the coefficient matrix. In the past, designs of this type have carried out a form of factorization which is not immediately conducive to back-substitution [2, 3, 4]. The goal of the design to be presented here is an array in which the resolved upper triangle has a unit diagonal therefore prepared for immediate back-substitution.

In addition, methods for adjusting the complex Jacobian coefficient matrix, of the type encounted in load flow analysis, to the systolic array must be reviewed. In this case, single matrix elements are actually sets of 2x2 complex sub-entries relating real and reactive power to changes in nodal voltages and phase angles.

The requirement of a band form system for triangulation, as mentioned before, is still applicable to this complex system. It is assumed that the system has been permuted, a priori, to a minimum band form.

1.1 Statement of Problem

It is well known that the bottleneck operation encountered in large scale power load flow programs is the solution of the system of complex simultaneous equations derived from a Newton-Raphson formulation [5]. A

huge amount of research effort has been spent investigating new generation computer architectures in order to obtain a faster and more efficient solution [2, 3, 4]. Likewise, the aim of this research is to determine if a VLSI systolic array structure can be applied to an existing load flow program, via simulation, for effectively reducing the solution time.

1.2 Approach

The primary goal of this thesis is to obtain an accurate quantification of processing time (throughput) for a VLSI systolic processing array imbedded, via simulation, in an industry standard load flow program to promote its efficiency in improving load flow computation time. In addition, the same simulation will provide an estimation of the chip edge size of the VLSI array obtained as a function of word size, matrix bandwidth, matrix dimension and minimum lithographic linewidth parameters. Details of the approach towards this goal will be described in this section.

Foremost, the structure of an industry standard, large-scale (1500 bus), serial computer load flow program is explained. The program is modified to run on the CDC CYBER-750 computer at Michigan State University and verified against benchmark results.

For the purpose of benchmarking, the program is executed with several different sets of data independently to produce various records of solution time, mainly the elimination time (or the triangulation time) and iteration counts. The time records are known to be high when

compared to those available for a scientific-attached array processor and are expected to be tremendously high when compared to the VLSI systolic array processor. This is because the data processing inside either array processor is in parallel and pipeline fashion and thus inherently faster than Von Neumann type architectures.

As a second step a subroutine is designed to normalize all numbers that are involved in the calculation of the resolved Jacobian elements. This subroutine is to be annexed to the MSU load flow program version. Whenever a number is transferred to this subroutine, it is first changed to a fixed-point binary number and then truncated to a predefined number of bits. Finally, this truncated binary number is converted back to the original representation and returned back to the calling program thus simulating the effect of fixed-point round off.

The whole program (the full load flow program plus the rounding subroutine) is then executed to ascertain the necessary number of fixed-point bits required for load flow convergence.

The third step involves restructuring the version of a VLSI systolic array for triangulating large augmented band form coefficient matrices based on a recently proposed algorithm [4]. This restructuring includes the addition and revision of some hardware in the suggested array structure such that it can do strict upper triangulation (i.e. resolve the diagonal to unity).

Fourthly, the development of a hardware interface system between the host computer and the processing array will be discussed. This interface is responsible for the adjustment of operands transferring back and forth between the host computer and the peripheral array

processor. The interface is necessary since the number format of these two computing structures is different. The overall block structure of the interface will be presented and explained.

Finally, the proposed VLSI circuit model is simulated with variuos parameters of word size, matrix bandwidth, matrix dimension and minimum lithographic linewidth. The word size, which determines the width in bits of the systolic array processing elements, is found from the second step of the approach. The matrix bandwidth is found by reducing the original bandwidth as much as possible. The matrix dimension is dependent on the size of the electrical utility network. The minimum lithographic linewidth, λ, is varied to project the trend of I. C. technology. Lastly, in order to determine if the goal of load flow throughput enhancement is achieved, the obtained time results are to be compared with the results from the previous works [7, 8].

1.3 Contributions

The major contributions of this thesis are summarized as follows:

1. An existing VLSI systolic array algorithm based on concurrent Gaussian elimination is successfully improved such that it can effectively triangulate any augmented band form matrix to a strict upper triangle and a unit diagonal.

2. The VLSI algorithm is successfully applied to a representative

load flow program and the accuracy of the load flow convergence results are within tolerance.

3. A suggested interface is presented for the proposed special purpose processor attached to a host computer.

4. The significance of using a VLSI algorithm to relax the computational bottleneck that occurs in many engineering problems, specifically the load flow problem, is reinforced by the data collected.

CHAPTER II

BACKGROUND

2.1 Load Flow Analysis

Load flow analysis involves calculating power flows and voltages for a specified utility system subject to the regulating capability of electric generators, condensers, and tap changing under load transformers. This procedure is used to determine a set of complex bus voltages and line power flows representing the static operating condition of the power system network. The analysis involves solving a set of non-linear algebraic equations which is typically large and sparce (< 3 % non-zero elements). The set of equations, obtained from nodal voltages and the network admittance, is solved for bus voltage magnitudes and phase angles. Then, the voltages and phase angles are used to calculate the power flows.

Although a power system is operated with three-phase generation and loads, a single-phase representation is adequate because the system is considered balanced. For each bus there are four possible parameters, namely the voltage magnitude, the phase angle, and the real and reactive power. Two of these four quantities are specified at each bus representing one of three bus types: a voltage controlled bus, a load bus or a swing bus. A swing bus, also known as a slack bus, must be designated to supply the additional real and reactive power for accommodating transmission losses inasmuch as these quantities are

unknown prior to the final solution.

Although there are many numerical methods that can be used to solve the load flow equations, the most economical and effective one, from the standpoint of computer memory storage and solution time, is an approach employing the bus admittance matrix and the Newton-Raphson method [5]. This approach involves the derivation of a set of non-linear equations which is used to express the specified real and reactive power in terms of bus voltages and phase angles. A Jacobian matrix equation in polar notation is derived which is in the form of $\underline{A}\underline{x}=\underline{b}$ as follows:

$$\begin{bmatrix} \frac{\partial P}{\partial \delta} & \vdots & \frac{\partial P}{\partial |E|} \cdot |E| \\ \cdots & \vdots & \cdots \\ \frac{\partial Q}{\partial \delta} & \vdots & \frac{\partial Q}{\partial |E|} \cdot |E| \end{bmatrix} \bullet \begin{bmatrix} \Delta\delta \\ \cdots \\ \frac{\Delta |E|}{|E|} \end{bmatrix} = \begin{bmatrix} \Delta P \\ \cdots \\ \Delta Q \end{bmatrix} \qquad (2\text{-}1)$$

where ΔP = real power mismatch,

ΔQ = reactive power mismatch,

$\Delta\delta$ = correction to the nodal voltage phase angle,

$\Delta|E|$ = correction to the nodal voltage magnitude.

To initiate the procedure all bus voltages and phase angles are set to the swing bus value. Next, real and reactive bus power and current are evaluated. Then, the Jacobian matrix equation is formed and the linear system is solved for the nodal voltage and phase angle corrections. Now, a new set of bus voltages is derived using the calculated corrections and the real and reactive bus power are reevaluated. The process continues with convergence checked by examining a tolerable power mismatch. If the tolerence is not

satisfied, the process continues iteratively until convergence results.

The computational bottleneck in the overall load flow method lies in the solution of the Jacobian matrix equation at each iteration of the Newton-Raphson approach. Many procedures have been developed to ease this hindrance as much as possible [6, 7, 8]. One of these methods is first to reorder the system of equations by a strategy which attempts to locally minimize the number of new non-zero elements which are introduced into the matrix during the elimination process. Then the network variables are packed into a sparse data structure and a specific form of Gaussian elimination, operating on one row at a time, is used to acquire the upper triangular factorization [6].

Another technique makes clever use of a one-dimensional array processor to achieve fast elimination [7, 8]. In this method, peripheral attached array processors, such as the AP-190L, are utilized in the solution of the highly vectorizable elimination procedure. With proper restructuring of the computer algorithm, savings of up to 64.5 % of the overall solution time, compared to the traditional serial computer approach, have been achieved [8]. However, as the power networks grow much larger, a bigger array processor with larger internal memories have to be designed for solving the problem at the same speed.

2.2 Traditional Load Flow Program Structure

As an initial step in this research, a traditional and representative load flow computer program was brought up on the Michigan State University CYBER-750 computer. Serving as a basic reference

program, a version of Philadelphia Electric Company's POWERFLO program was obtained [9]. This program was originally modified at the University of Pittsburgh to faciliate studies of this type*.

This program utilizes the Newton-Raphson approach with the solution of the resulting linear equations by row-ordered Gaussian elimination and subsequent back-substitution of the resulting strict upper triangular matrix.

2.2.1 Program Structure

Although there are many procedures inside a standard load flow package, only those steps involving the generation of the Jacobian matrix and subsequent solution by Gaussian elimination are of interest in this study. A brief explanation of these two steps follows.

1. The MAIN program calls a subroutine MATGEN which oversees the matrix generation and factorization.

2. Inside MATGEN, the program starts looping over all the nodes in the system. A "working row" of the Jacobian is generated for each node.

---

*We gratefully acknowledge the help of Professors M. H. Mickle and W. G. Vogt, Department of Electrical Engineering, University of Pittsburgh, for supplying us with the working version load flow package.

3. For each "working row" the subroutine ELIM is called to eliminate all the elements of that row up to the diagonal. The eliminated coefficients (factors) of the upper triangle are then packed into a condensed and indexed form.

4. After all rows have been eliminated, subroutine BAKSUB is called for back-substitution and thus the nodal voltage corrections are obtained.

5. Finally, subroutine ADJUST is called to perform inter-iteration adjustments and convergence is checked before the next Newton-Raphson iteration is initiated.

6. If convergence is not obtained, the loop is either terminated (divergent case) or steps 1 through 5 are repeated until a satisfactory solution is obtained.

The program flowchart is illustrated in Figure 2.1.

2.2.2 Jacobian Matrix Triangulation by Gaussian Elimination

Since the main concern of this research is in reducing the time required in the factorization aspect of a load flow program, the exact details of the elimination process will be explained in this section.

The solution scheme used in the standard load flow program processes and stores only non-zero elements. This is done by storing a

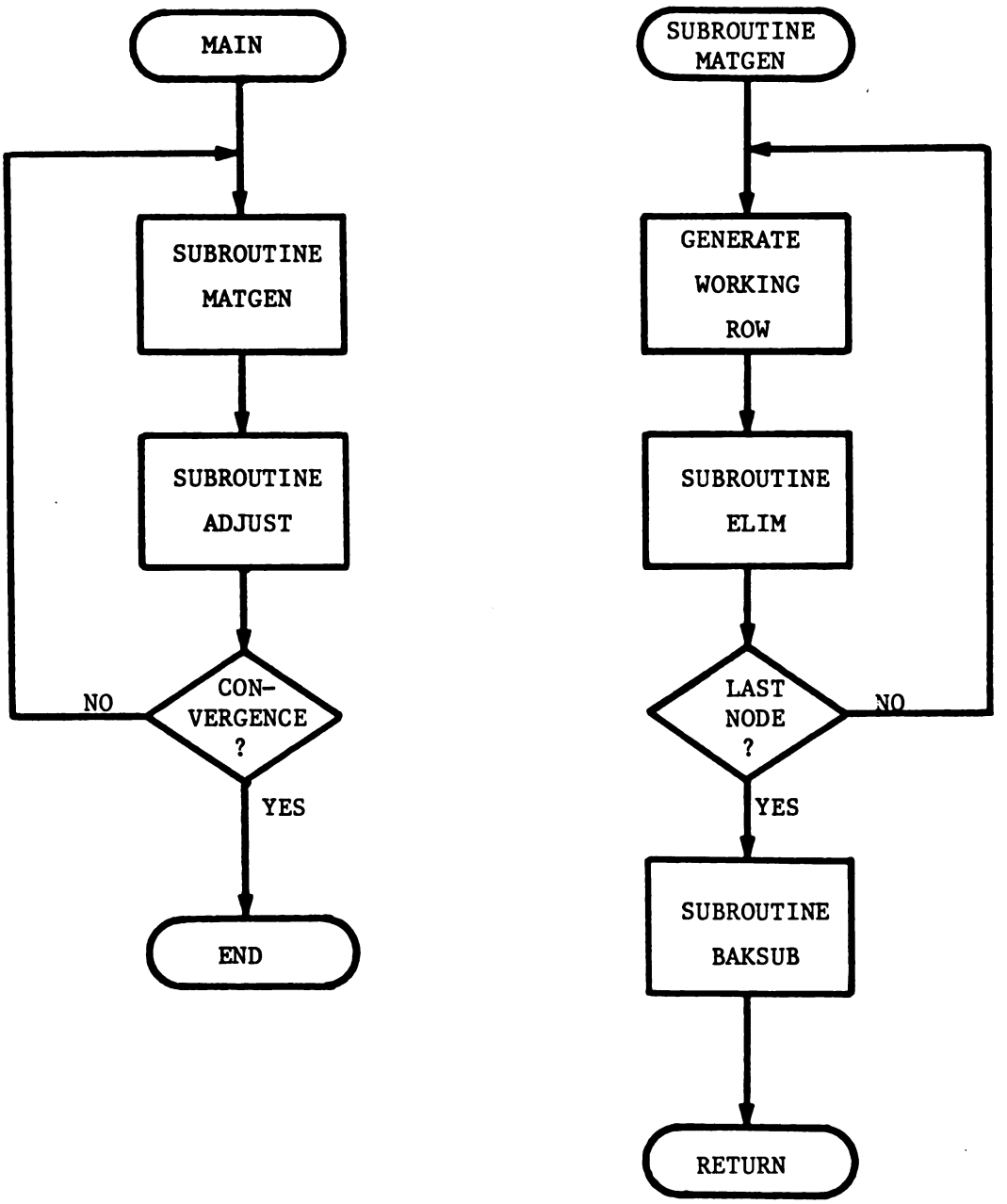


Figure 2.1 Structural flow of POWERFLO program.

compacted table of factors (upper triangle) and a compacted working row. In addition, a set of pointers and counters for tracking the process are required.

The elimination process is actually a triangular decomposition of the Jacobian matrix by Gaussian elimination which is described in many books on matrix analysis [10, 11]. However, in here, only non-zero elements of each row up to the diagonal are eliminated and stored according to the current pointers and indexed counters before proceeding to the next row. This is shown pictorially in Figure 2.2.

The figure depicts the matrix form just prior to the elimination of row i. Since only the non-zero elements of the upper triangle are stored, it is not necessary to store all the resultant elements depicted in the figure.

The form of each element shown in Figure 2.2 is actually a set of 2x2 complex sub-entries, expressing the partial derivatives of real and reactive power with respect to nodal voltage and phase angle (See Equation 2-1). For easy reference, Equation 2-1 can be rewritten in the form of Equation 2-2.

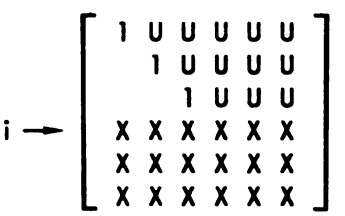


Figure 2.2 Row-ordered elimination pattern in the POWERFLO program.

$$\begin{bmatrix} H & N \\ J & L \end{bmatrix} \bullet \begin{bmatrix} \Delta\delta \\ \Delta V \end{bmatrix} = \begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} \tag{2-2}$$

where ΔP, ΔQ and $\Delta\delta$ are defined as before

and $\Delta V = \Delta|E|/|E|$,

$H = \partial P/\partial\delta$,

$N = (\partial P/\partial|E|)\cdot|E|$,

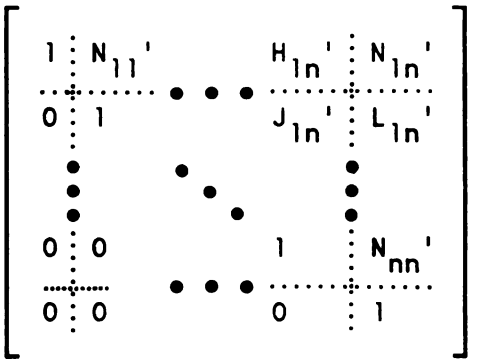
$J = \partial Q/\partial\delta$,

$L = (\partial Q/\partial|E|)\cdot|E|$.

Therefore, the Jacobian matrix derived from a system of $n+1$ nodes can be expressed in the form

$$\begin{bmatrix} H_{11} & N_{11} & \cdots & H_{1n} & N_{1n} \\ J_{11} & L_{11} & \cdots & J_{1n} & L_{1n} \\ \vdots & \vdots & & \vdots & \vdots \\ H_{n1} & N_{n1} & \cdots & H_{nn} & N_{nn} \\ J_{n1} & L_{n1} & \cdots & J_{nn} & L_{nn} \end{bmatrix}.$$

After the elimination is complete, the matrix is of the strict upper triangular form



and thus ready for immediate back-substitution.

2.3 Banded Matrix

In this section, the description and advantages of reordering a matrix to band form prior to performing Gaussian elimination in an array processing environment are presented.

Any matrix, $\underline{A}$, is said to be banded, or in band form, if all the non-zero elements are clustered about the main diagonal. The bandwidth, B, a measure of this clustering, is defined by

$$B = \max \left\{ |i-j| : a(i,j) \neq 0 \right\}. \qquad (2\text{-}3)$$

For a structurally symmetric matrix, as in the case of Jacobian matrix derived from the Newton-Raphson approach to the load flow problem, the above bandwidth definition implies that the maximum width, in number of elements, between the first and last non-zero entry of any row is 2B+1. This maximum width will be referred to as the "full breadth" of the matrix. The bandwidth and full breadth are illustrated in Figure 2.3.

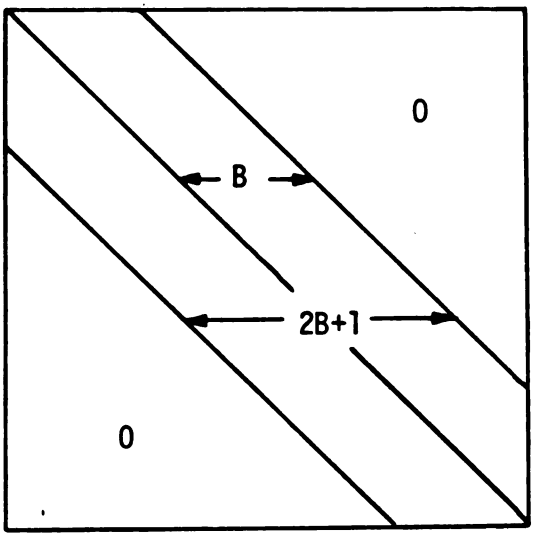


Figure 2.3 Bandwidth and full breadth of a band form matrix.

There are several heuristic algorithms that have been developed to reorder the matrix $\underline{A}$ to a reduced band form [12, 13, 14]. The two most commonly used are the Cuthill and Mckee algorithm [13] and an improved version described by Gibbs, et al [14]. These two algorithms were basically developed with the same fundamental graph theoretic concepts.

It has been shown that matrix banding is theoretically the optimal network ordering for systems to be processed on an attached array processor architecture [7]. Furthermore, in one particular study the banded form ordering on the array processor yields the fastest solution time for networks less than about 1300 busses [7, 8]. This is reasonable because for any sparse matrix, regardless of the non-zero pattern, the total computing time required to resolve the upper triangle by using vector row-ordered elimination on an array processor, is

bounded by a function of B and in the limit by N.

This can be demonstrated by considering an N-dimensional matrix $\underline{A}$ of minimum bandwidth B. The time required to eliminate one variable from any given row is

$$T = T_s + B \cdot T_{op}, \tag{2-4}$$

where T_s and T_{op} are the setup and unit operation times of the array processor function providing scalar-vector multiply followed by vector-vector subtract. This corresponds to the operation illustrated in Figure 2.4.

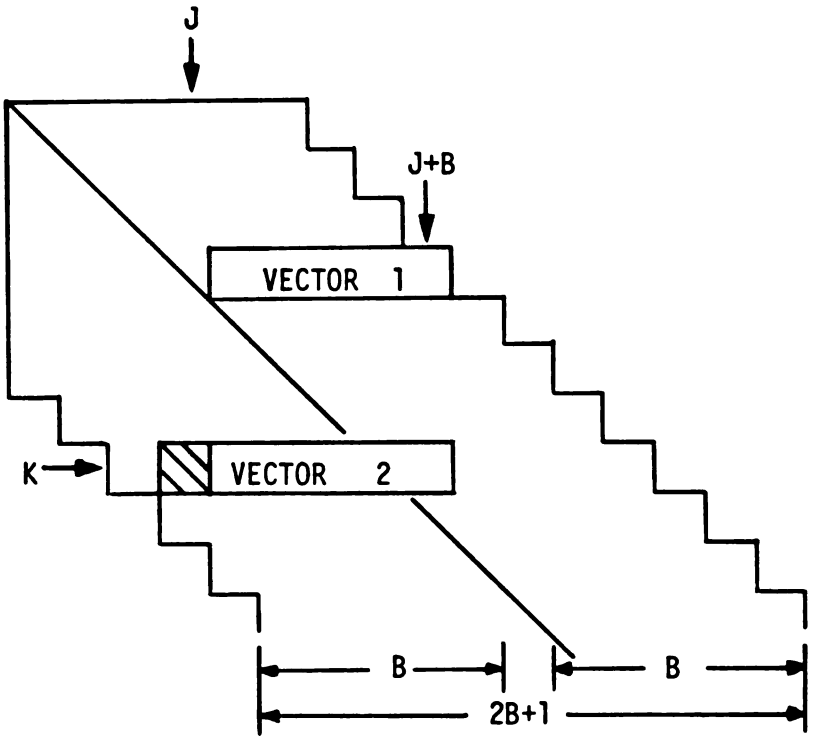


Figure 2.4 Single variable elimination from a band form matrix.

In this figure, only the shaded element and vectors 1 and 2 are required to complete a given inner loop step. This is analogous to the loop on index I of a FORTRAN coded procedure for the same row-ordered elimination as follows:

```
      DO  10  K = 1, N
      DO  10  J = 1, K
      DO  10  I = J+1, J+B
10    A(K,I) = A(K,I)-A(K,J)*A(J,I).
```

Building on this, the time required to eliminate all the variables of any row K is given by

$$T = B \cdot T_s + B^2 \cdot T_{op} \tag{2-5}$$

corresponding to those elements shown shaded in Figure 2.5.

The resolved elements to the right of the main diagonal must now be divided by the diagonal entry. This adds one setup time and B more unit operations and Equation 2-5 becomes

$$T = (B+1) \cdot T_s + B \cdot (B+1) \cdot T_{op}', \tag{2-6}$$

where T_{op}' is now the maximum operation time between the multiply-subtract and division vector functions.

For an N-dimensional matrix, N such row eliminations are required. Thus, an upper bound on the total elimination time is given by

$$T_N(B) = N \cdot (B+1) \cdot T_s + N \cdot B \cdot (B+1) \cdot T_{op}'. \tag{2-7}$$

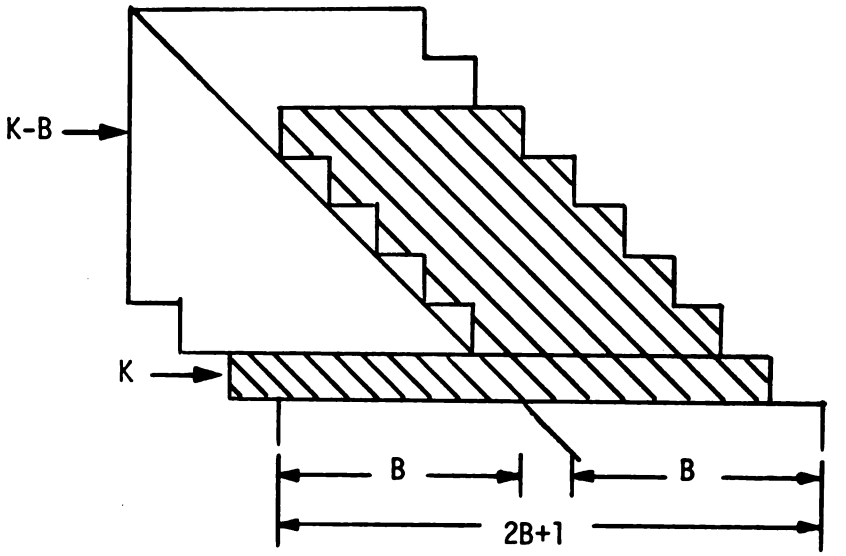


Figure 2.5 Full row elimination from a band form matrix.

Now, if a parallel pipeline structure such as a systolic array is used, the total elimination time is given by

$$T_N(B) = (B+1)\cdot T_s + N\cdot T_{op}'. \qquad (2\text{-}8)$$

As a result, since T_s, N and T_{op}' are defined variables, to minimize the elimination time implies the minimization of B for both array and parallel pipeline architectures.

In order to achieve the most efficient operation inside the proposed VLSI array structure for matrix triangulation, all original load flow bus and line data must be renumbered to reduced band form beforehand. Also, as will be discussed later, the size of the systolic computing structure is dependent on the bandwidth of the matrix due to

the structural properties of the hardware algorithm [4]. In conclusion, reordering a matrix to reduced band form prior to triangulation not only reduces the solution time but will also yield bounds on hardware dimensions in an array or pipeline processing environment.

2.4 VLSI Systolic Algorithm for Matrix Triangulation

A systolic array processor is a system having a two-dimensional configuration of processing elements (PE's) in a parallel pipeline fashion. The processor synchronously pumps data between levels of PE's performing part of an overall computation at each time step such that a regular flow of data is kept up in the network. The algorithms use distributed control achieved by simple local control mechanisms such as stacked one-dimensional arrays of PE's located between rows of latches. Ideally, these hardware algorithms are to be implemented on a single chip or perhaps in a modular design on a few chips with the use of VLSI technology. The modular approach will not be addressed here.

VLSI technology, however, places new constraints on computer architects. For high-performance algorithms to be implemented at low cost, the algorithm must possess the properties of regularity, local communication, and parallelism and pipelining [15]. An early example of this type of algorithms is the systolic array for solving linear systems of algebraic equations proposed by Kung and Leiserson [16]. This structure, built of simple inner-product step and division function processors, can be used to carry out L-U decomposition on a full matrix, $\{\underline{A}\}$. A revised version appeared, advancing Kung's design, in which the

proposed VLSI architecture would perform L-U decomposition of an entire linear system of equations, $\{\underline{A}|\underline{b}\}$ [3]. However, in both of these examples, practical details such as input/ouput circuits and processing cells designs were not considered.

Recently, an improved systolic structure was developed by Hsu and Shanblatt [4]. This structure, by isolating a row of processing cells, will triangulate an arbitrarily large augmented band system $\{\underline{A}|\underline{b}\}$ to upper triangular form $\{\underline{U}|\underline{d}\}$ aligning the $\underline{b}$ and $\underline{d}$ vectors. Moreover, in this work, the potential I/O bottleneck was studied with respect to the overall solution time, and, the processing cells as well as I/O port circuit candidates were designed and evaluated.

The structure proposed by Hsu and Shanblatt utilizes both parallel and pipeline concepts and performs the triangulation in 2N+2B time steps. Moreover, it requires B(B+1) inner-product cells and B division cells, where N and B are the full dimension and the reduced bandwidth of the coefficient matrix, respectively.

In order to illustrate this, consider an Nx(N+1) augmented matrix system $\{\underline{A}|\underline{b}\}$ with a bandwidth of 3 as depicted in Figure 2.6a. After the upper triangulation has been performed, the system appears as illustrated in Figure 2.6b. The array structure corresponding to this matrix example is shown in Figure 2.7. A port-coefficient time table for the triangulation of this system is given in Table 2.1.

In Figure 2.7, the inner-product cells are called MAC's (Multiply and Add Cells) and perform w=xy+z, x=x and y=y. The complementation circle shown on the input of the topmost row of MAC's refers to a two's complement operation. The DC's (Division Cells) perform g=e/f and f=f.

$$\left\{\begin{array}{ccccccccccc|c}
a_{11} & a_{12} & a_{13} & a_{14} & & & & & & & & b_1 \\
a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & & & & & & & b_2 \\
a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & \cdot & & & & & & b_3 \\
a_{41} & a_{42} & a_{43} & a_{44} & \cdot & & \cdot & & & & & b_4 \\
 & a_{52} & a_{53} & \cdot & \cdot & & & \cdot & & & & b_5 \\
 & & \cdot & & & \cdot & & & & \cdot & & \cdot \\
 & & & \cdot & & & \cdot & & & \cdot & & \cdot \\
 & & & & \cdot & \cdot & & \cdot & & \cdot & & \cdot \\
 & & & & & a_{N(N-3)} & a_{N(N-2)} & a_{N(N-1)} & & a_{NN} & & b_N
\end{array}\right\}$$

Figure 2.6a Augmented matrix $\{A|b\}$.

$$\left\{\begin{array}{ccccccccccc|c}
u_{11} & u_{12} & u_{13} & u_{14} & & & & & & & & d_1 \\
 & u_{22} & u_{23} & u_{24} & u_{25} & & & & & & & d_2 \\
 & & u_{33} & u_{34} & u_{35} & \cdot & & & & & & d_3 \\
 & & & u_{44} & u_{45} & \cdot & & \cdot & & & & d_4 \\
 & & & & u_{55} & \cdot & & & \cdot & & & d_5 \\
 & & & & & \cdot & & & & \cdot & & \cdot \\
 & & & & & & \cdot & & & \cdot & & \cdot \\
 & & & & & & & \cdot & & \cdot & & \cdot \\
 & & & & & & & & & u_{NN} & & d_N
\end{array}\right\}$$

Figure 2.6b Augmented upper triangular matrix $\{U|d\}$.

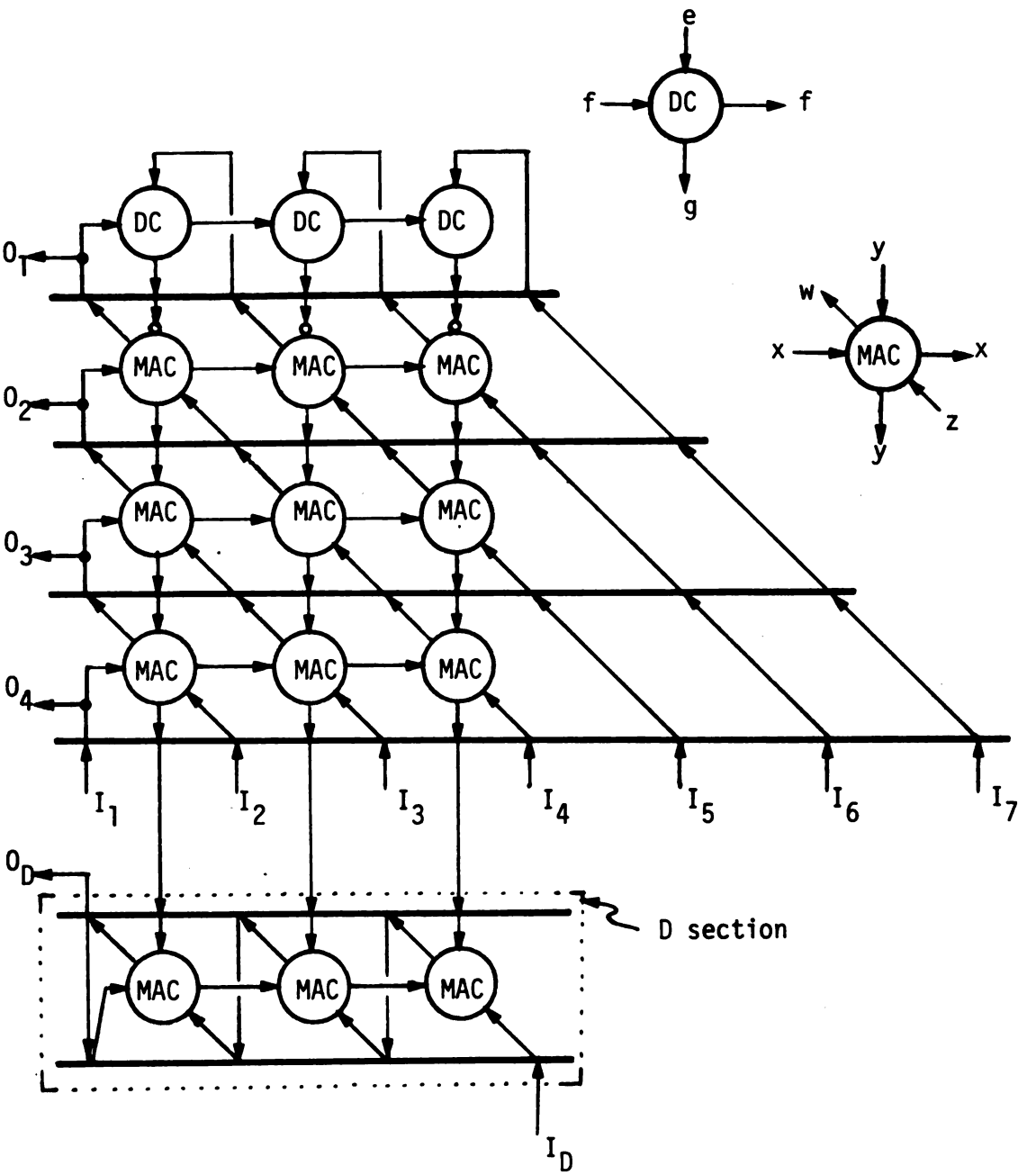


Figure 2.7 Computing array structure for matrix of arbitrary dimension with B=3.

Table 2.1 Port-coefficient time table.

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_D	O_1	O_2	O_3	O_4	O_D
t_1				a_{11}	a_{21}	a_{31}	a_{41}						
t_2				1									
t_3			a_{12}	a_{22}	a_{32}	a_{42}	a_{52}	b_1					
t_4				1					u_{11}				
t_5		a_{13}	a_{23}	a_{33}	a_{43}	a_{53}	·	b_2		u_{12}			
t_6				1					u_{22}		u_{13}		
t_7	a_{14}	a_{24}	a_{34}	a_{44}	a_{54}	·	·	b_3		u_{23}		u_{14}	
t_8				1					u_{33}		u_{24}		d_1
t_9	a_{25}	a_{35}	a_{45}	a_{55}	·	·	·	b_4		u_{34}		u_{25}	
t_{10}				1					u_{44}		u_{35}		d_2
t_{11}	a_{36}	a_{46}	·	·	·	·	·	b_5		u_{45}		·	
t_{12}				1					u_{55}		·		d_3
t_{13}	a_{47}	·	·	·	·	·	·	·		·		·	
t_{14}				1					·		·		d_4
t_{15}	·	·	·	·	·	·	·	·		·		·	
t_{16}				1					·		·		d_5
t_{17}	·	·	·	·	·	·	·	·		·		·	
t_{18}				1					·		·		·
·													·
·													·
·													·
t_{2N+6}													d_N

The D section is isolated in order to correctly align $\underline{b}$ vector elements, which subsequently became $\underline{d}$ vector elements, during the triangulation. This right-hand-side vector is pumped into input port I_D and out of output port O_D.

Input and output operands are pumped in and out of the processing array through the I/O ports, I_1-I_7, I_D and O_1-O_4, O_D. Synchronism is provided by latch arrays which are depicted by thick black lines between the rows of PE's. The number of required input ports is given by the full breadth of the matrix and right-hand-side vector input port, or 2B+2, while the number of output ports is defined by the bandwidth plus a diagonal and right-hand-side output port, or B+2.

The operands shown in the time table (Table 2.1) are interspaced with zeros and ones to provide proper synchronism among array coefficients. The use of these "spacers" will be explained in next chapter.

As the time table shows, 2N+2B (2N+6 in this example) time slots are required to obtain $\{\underline{U}|\underline{d}\}$. The algorithm may therefore be classified 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 [3].

In conclusion, all of the above mentioned algorithms have the same important features: regularity, expandability, and extensive use of parallel and pipeline concepts. This makes them particularly well suited for VLSI implementation. Based on these algorithms, a revised version of a VLSI algorithm for band matrix triangulation more conducive to use in the load flow program will be presented next.

CHAPTER III

SYSTOLIC ARRAY FOR TRIANGULATION OF LOAD FLOW MATRIX FORM

The development of an improved version of the VLSI algorithm for fast band matrix triangulation will be presented in this chapter. This algorithm is to replace the routine for matrix triangulation inside a serial computer load flow program such that the overall solution time is reduced at no loss of accuracy. As a starting point towards practical realization, the number of fixed-point bits for load flow convergence is determined because the present algorithm is constrained to a fixed-point architecture due to chip size and lithography limitations. This is done by executing a modified MSU load flow program version on CYBER-750 computer. A method for combining, or interfacing, this structure to the serial host computer will be explained in next chapter.

3.1 VLSI Array Structure Modification

In standard load flow studies triangulating the $\underline{A}$ matrix by a serial computer program results in a strict upper triangle with a unit diagonal [9]. Therefore, it is necessary to modify the algorithm of [4] to match this requirement.

Referring to the structure and time table shown in Figure 2.7 and Table 2.1, the diagonal elements of the resultant upper triangle (u_{ii},

i=1,2,...N) pumped out of the O_1 port, are not necessarily equal to one. In order to obtain a definate unit diagonal, all of the elements in each row must be divided by the diagonal element of that row before they are pumped out of the array. That is

$$u_{ij}' = u_{ij}/u_{ii}, \tag{3-1}$$

and

$$d_i' = d_i/u_{ii}, \tag{3-2}$$

for all i=1,2,...N and j=i,i+1,...i+B.

Since the diagonal elements must be equal to one after the final division, at any time slot, the maximum number of the elements which must be divided by the diagonal is given by B+1. This includes B $\underline{A}$ matrix entries and the right-hand-side entry. As a worst case result, by adding an extra row of B+1 division cells to the array, a strict upper triangle may be otained. However, the size of the structure will be increased tremendously since the size of a DC is much larger than that of an MAC [4].

Fortunately, by close examination of the structure, it is found that the additional division as described above can be done by "cycle stealing" instead of the addition of another row of DC's.

Focusing now on the present row of DC's in the example, a_{21}, a_{31} and a_{41} must be divided by a_{11} at time t_4. At t_5, though, null divisions take place (0÷1) and this is the case during every second time slot from then on. In other words, at every t_{i+B+1}, where B=3 and i=1,3,5,...2N-1, the operation inside the DC's is zero-divided-by-one.

Indeed, the null operations are unnecessary for the process of calculation because the results of these operations are always subsequently multiplied by zero later in the procedure. Therefore, no matter what value is obtained from the division, it has no effect on the overall solution. As a result, cycles stolen during these operations can be used to perform the additional division required to unitize the diagonal. This is accomplished by adding multiplexers (MUX's) and latches to the array structure. However, only B DC's are available and B+1 divisions are required at a time. Therefore, one additional DC must be added to the structure.

In order to illustrate this modified algorithm, the same augmented system $\{\underline{A}|\underline{b}\}$ with a bandwidth of 3, as depicted in Figure 2.6a, is considered. After the upper triangulation is complete, the system of equations is as shown in Figure 3.1. The modified computing structure

$$\left\{\begin{array}{ccccccccc|c}
1 & u_{12}' & u_{13}' & u_{14}' & & & & & & d_1' \\
 & 1 & u_{23}' & u_{24}' & u_{25}' & & & & & d_2' \\
 & & 1 & u_{34}' & u_{35}' & . & & & & d_3' \\
 & & & 1 & u_{45}' & . & . & & & d_4' \\
 & & & & 1 & . & & . & & d_5' \\
 & & & & & . & & & . & . \\
 & & & & & & . & & . & . \\
 & & & & & & & . & . & . \\
 & & & & & & & & 1 & d_N'
\end{array}\right\}$$

Figure 3.1 Augmented strict upper triangular matrix $\{U'|d'\}$ with unit diagonal.

corresponding to this example is illustrated in Figure 3.2. A port-coefficient time table for the triangulation of this system is given in Table 3.1.

The multiplexers (MUX's) shown in Figure 3.2 are controlled by a simple true and false signal synchronous with the clock. At every even time slot, t_2, t_4, t_6..., all lines marked L_1 are selected to pass the data. However, at every odd time slot the L_2 lines are selected.

The stack section (S section) depicted in the same figure is composed of arrays of latches which serve as an FIFO (First-In-First-Out) memory stacks. The number of stacks, as well as the number of latches inside each stack, varies with B. The details of their relationship will be explained in Section 3.2.2.

Referring to this structure (Figure 3.2) and the corresponding time table (Table 3.1), during the period t_4 all lines marked L_1 are selected to pass the operands a_{11}, a_{12}, a_{13}, and a_{14} to the row of DC's and enter a_{11} (same as u_{11} as defined in Table 2.1) into stack S_5. At the next slot, t_5, L_2's are selected to pass and store operands. Simultaneously, u_{11} propagates to the next stage of S_5 and u_{12} is put onto S_4. At t_6, L_1's are selected again to pass operands as well as push u_{22} onto S_5. At the same time, while u_{11} and u_{12} are pushed further along in S_5 and S_4, u_{13} is put onto S_3. Eventually, at the end of t_8, d_1, u_{14}, u_{13}, u_{12} and u_{11} are at the topmost latch of S_1 through S_5, respectively. Then, during t_9, L_2's are selected such that d_1, and u_{14}-u_{12} are all simultaneously divided by u_{11} in the DC's. Therefore, u_{12}', u_{13}', u_{14}' and d_1' are pumped out of output ports, O_1-O_3 and O_D at the end of t_9. After this at every two time slots, a new row of the upper triangle

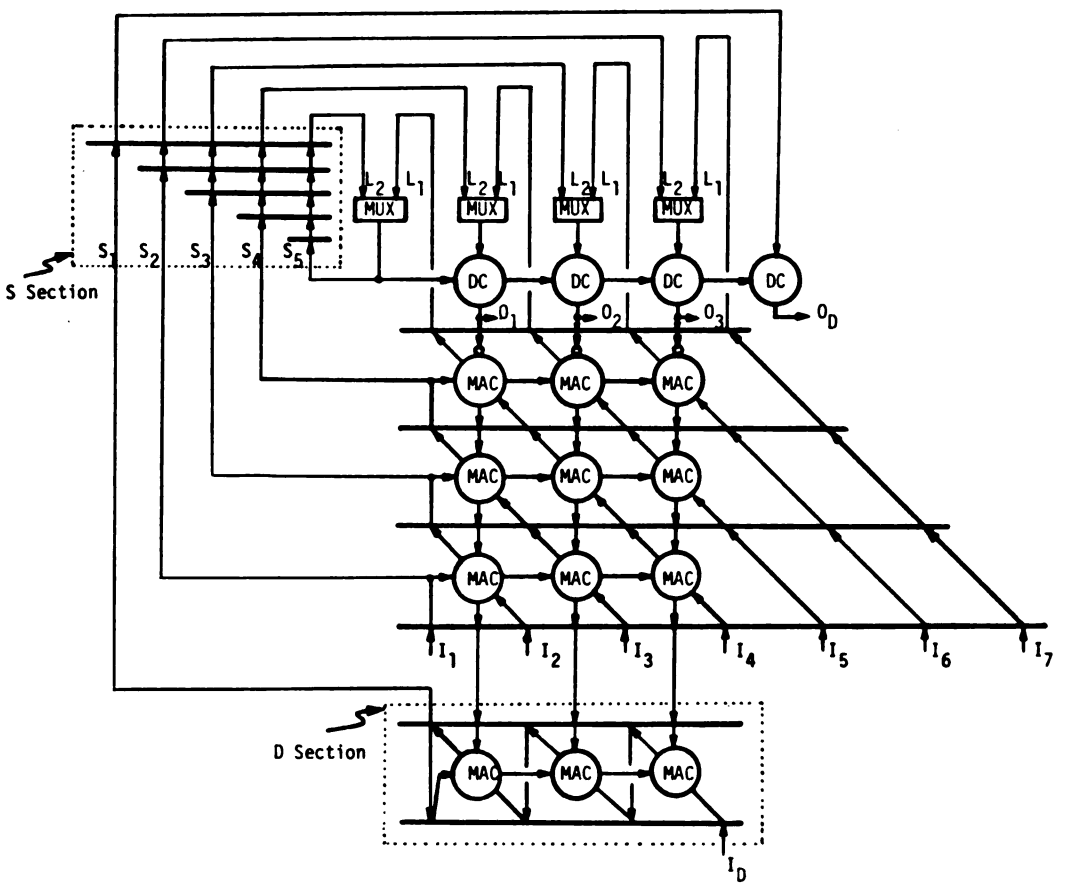


Figure 3.2 Revised computing array structure for matrix of arbitrary dimension with B=3.

Table 3.1 Revised port-coefficient time table.

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_D	O_1	O_2	O_3	O_D
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												
t_5		a_{13}	a_{23}	a_{33}	a_{43}	a_{53}	.	b_2				
t_6												
t_7	a_{14}	a_{24}	a_{34}	a_{44}	a_{54}	.	.	b_3				
t_8												
t_9	a_{25}	a_{35}	a_{45}	a_{55}	.	.	.	b_4	u_{12}'	u_{13}'	u_{14}'	d_1'
t_{10}												
t_{11}	a_{36}	a_{46}	.	.	.	.	.	b_5	u_{23}'	u_{24}'	u_{25}'	d_2'
t_{12}												
t_{13}	a_{47}	.	.	.	.	.	.	.	u_{34}'	u_{35}'	u_{36}'	d_3'
t_{14}												
t_{15}	.	.	.	.	.	.	.	.	u_{45}'	u_{46}'	.	d_4'
t_{16}												
t_{17}	.	.	.	.	.	.	.	.	u_{56}'	.	.	d_5'
t_{18}												
t_{19}	.	.	.	.	.	.	.	.	.	.	.	.
.												.
.												.
.												.
t_{2N+7}												d_N'

elements with an implied unit diagonal, is pumped out. Because of this implied unit diagonal, the number of output ports in this modified structure is one less than that of the original structure.

3.2 Computing Structure and Timing

The special purpose peripherial array processor for band matrix triangulation is mainly composed of input/output circuits and a computing array structure. This processor, consisting of those circuits, is to be attached to a serial host computer. The overhead timing and data controls of the peripheral processor are provided by the host computer through interconnected lines. The internal timing control can be supplied by either a built-in clock circuit or the host computer; however, the former method is preferred because it only requires a small increase in overall hardware size. Internal timing controlled by the host computer immediately increases the connection complexity between the two machines.

Operand I/O for the computing structure fundamentally suffers from a potential bottleneck problem due to practical fabrication limitations of pin-out and packaging considerations. Fortunately, through careful design of I/O circuits, the I/O bottleneck can be avoided [17]. This will be shown among the results presented in Section 6.1.

The computing array depicted in Figure 3.2 requires only five schematic logic circuit diagrams of function blocks, namely multiplexer, latch, stack, MAC and DC. The design of each function block is based on previous work [4] and crucial details of these designs, as well as

timing control, are explained in the following sections.

3.2.1 Multiplexer

A 2-to-1 multiplexer (2:1 MUX) is simply constructed by a pair of pass transistors under mutually exclusive control of a single signal line with an additional inverter. This is illustrated in Figure 3.3. The pulse width and frequency of the control signal (CS) depend on the sequence of data flow or operating time step. Whenever this signal is true (high), the right-hand-side transistor is turned on. This allows the data in line L_1 to pass through. If the signal is false (low), the left-hand-side transistor will be turned on allowing data in L_2 to pass.

The MUX's inside the computing structure are built solely from this basic block and are simply controlled by a signal synchronous with the

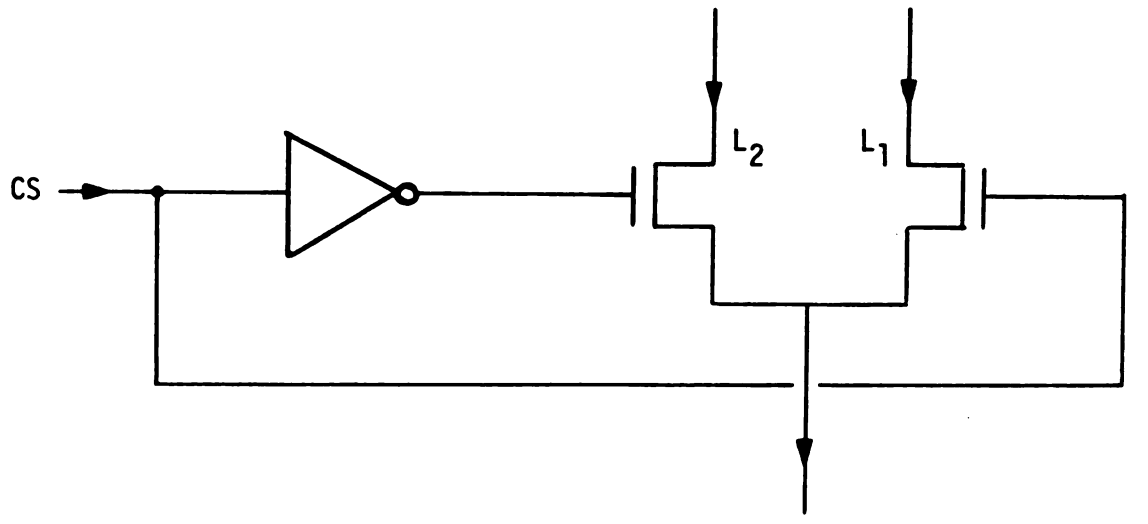


Figure 3.3 2:1 multiplexer.

system clock. The frequency of this signal is half of that of the signal controlling the latches between rows of PE's. Therefore, the L_1 and L_2 lines are selected alternately. This control signal, which will be called α, is shown in the timing diagram presented in Section 3.2.4.2.

Due to the nature of the modified triangulation algorithm, the L_1 lines are selected at either every odd or even time slot depending on the bandwidth. If B is an odd number, as in the previous example where B=3, L_1 lines will be selected at every even time slot. This is because it requires B time steps to initially fill up the topmost row of MAC's. Therefore, at every t_{B+1+i}, where i=0,2,4,...2N-2, all of the L_1 lines must be selected to pass the operands to the DC's as well as to the stack.

3.2.2 Latch and Stack

All latches are dynamic and consist of two pass transistors and two inverters [18]. A two-phase non-overlapping clock is required to load and refresh the latch data.

The memory stacks, S_1-S_5, inside the S section shown in Figure 3.2, are merely arrays of dynamic latches. The main function of these stacks is to consecutively store the individual elements of each row of the resolved upper triangle and right-hand-side vector. When all elements of a complete row are available in the topmost latch of each stack, they are simultaneously enabled into the DC's through the L_2 MUX lines. However, since L_2 lines are selected according to the even/odd control

signals, the operands (a complete row of elements) may have to wait another cycle before they can reach the DC's. This is to avoid a potential data flow conflict that may happen in the MUX's. Therefore, another function of the stacks is to serve as a delay circuit if such conflict exists.

By close inspection of the structure (Figure 3.2) it is seen that the last element of a particular row that is put onto the stack is the $\underline{d}$ vector element. In general, these $\underline{d}$ vector elements are actually available from the upper left corner latch inside the D section at every odd time slot starting at t_{2B+1}. However, L_2 lines are selected only at every t_{B+i} slot and L_1 lines are selected at every t_{B+1+i} slot. Therefore, all the elememts of a complete row can be pumped into the DC's through the MUX's at either t_{2B+1+i} if B is odd or t_{2B+i} if B is even (where i=2,4,6,...2N).

In the example of B=3, L_2 lines are selected at every odd time slot and one complete row of elements is also available at every odd time slot starting from t_7. Therefore, there is one latch inside the stack S_1 and all of the other stacks have an extra latch for producing a delay of one time step such that the sequence of data flow is properly adjusted. But, for an example of B being even, these L_2 lines are selected at every even time slot, and a complete row is obtained at every t_{2B+1+i}, for i=0,2,4,...2N-2, which is always odd. Therefore, this complete row can be latched directly into the DC's at t_{2B+2+i} which is always even. As a result, a row of latches inside the stack section actually can be saved without disturbing the proper sequence of data flow if B is even.

Based on these facts, the number of stacks, j, and the number of latches, L, inside each stack S_k are proportional to the bandwidth, B, and are given as follows:

$$j = 2 \cdot \lceil B/2 \rceil + 1, \tag{3-3}$$

and $$L(S_k) = k, \tag{3-4}$$

where $k = 1,2,\ldots j.$

The notation $\lceil x \rceil$ refers to the smallest integer that is not less than the real number x.

3.2.3 MAC and DC

The structure of an MAC is merely a Baugh-Wooley multiplier with an extra row of full-adders at its bottom edge [4, 19].

The convergence division algorithm [20] is chosen in the DC design because the number of iteration steps can be determined a priori in terms of word size and software convergence checking is not required. In addition, since this algorithm requires mostly iterative multipling procedures, the Baugh-Wooley algorithm can be applied again such that there is no loss in structural regularity.

This division algorithm can be partitioned into several subtasks. Thus, the algorithm can be realized in a pipeline structure having rows of latches located between rows of multipliers. The details of the numerical formulation of this algorithm can be found in [20] and an

example of the pipeline structure is presented in [4].

Due to the constraint of synchronous data flow between the rows of DC's and MAC's, the latch-to-latch time between rows of MAC's is constrainted by that of the DC. This is because the operands that will be pumped into the DC's are obtained from the topmost row of MAC's but the calculation of these operands requires previous results from the DC's first. In other words, there is a tightly coupled linkage between the DC's and the topmost MAC's. Therefore, eventhough the division algorithm can be pipelined, the worst case time slot is still the sum of all segment times. So, pipelining the division algorithm does not help in increasing throughput.

In addition, due to the nature of the convergence division algorithm, the number of iteration steps, m, of multiplication varies with the word size of the operands. The number of iteration steps is given by

$$m = \lceil \log_2(n) \rceil, \tag{3-5}$$

where n is the word size of the operands. Therefore, the number of rows of multipliers in the pipeline structure is also m. This implies that the hardware size of a DC is directly proportional to m times the number of multipliers in each row. As a result, an improved DC structure with drastically reduced hardware having approximately the same speed as before is developed next.

The improved structure has only one row of multipliers and is independent of m. This original design is illustrated in Figure 3.4. In this figure, the latches and MUX's have the same structure as

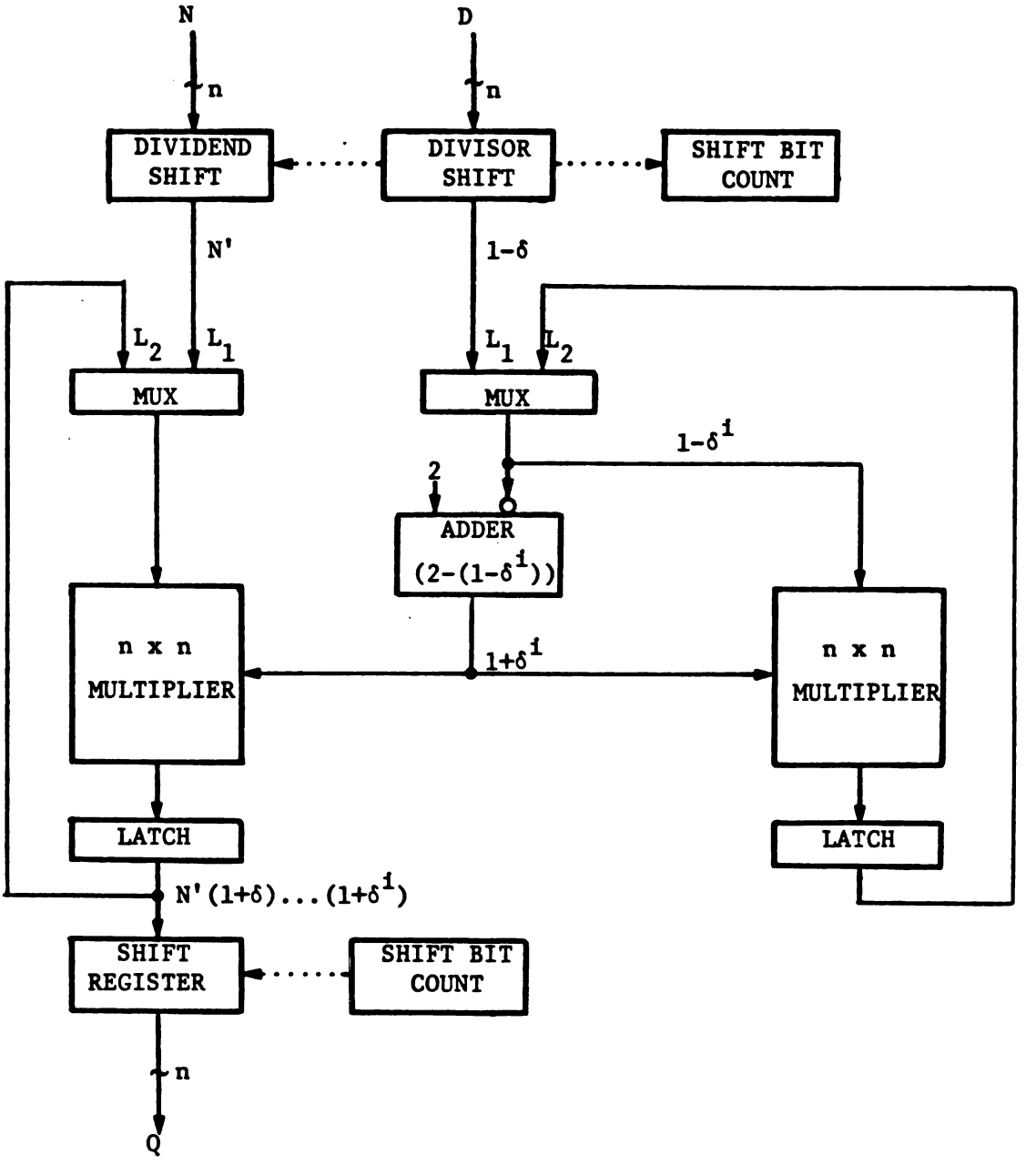


Figure 3.4 An n-by-n convergence division algorithm based DC.

described in Section 3.2.1 and Section 3.2.2. However, the pulse width and frequency of the signals for these latches and MUX's are different. These control signals will be explained and shown in the timing diagram presented in the next section.

After the dividend and divisor, marked as N and D in the figure, have been shifted, lines marked L_1 are selected to pass the shifted results to the adder and subsequently to the multipliers. This shifting, integral to the convergence division algorithm, is necessary in order to obtain a faster convergence [20]. The L_1 lines are selected only once in every PE's segment time slot. Then, the partial results, $N'(1+\delta)$ and $(1-\delta)(1+\delta)$, held by the latches are looped back to the adder and multipliers through the L_2 lines. This loop continues until the result, $N'(1+\delta)\ldots(1+\delta^i)$, where $i=2^m$ (m is defined in Equation 3-5), is obtained. Finally, a convergent quotient, Q, is available after the result has been shifted back according to the original shift bit count.

3.2.4 Timing

Recall that the whole computing array is pipelined in nature and therefore the operation time, t_{op}, must be at least the worst case segment time. In the array structure shown in Figure 3.2, the maximum segment time should be the propagation delay from the MUX's to those latches between the DC's and MAC's. As a result,

$$t_{op} = t_{MUX} + t_{DC} + t_{LATCH}. \qquad (3\text{-}6)$$

Since t_{MUX} and t_{LATCH} are simply the total delay of two pass transistors

and two inverters, the important focus must be placed on t_{DC}.

3.2.4.1 Internal DC Timing

Referring to Figure 3.4, the number of loops routing from the MUX through the adder, multiplier, latch and back to the MUX is given by m (Equation 3-5). Let t_s, t_a, and t_x be the shifting, adding, and multipling time, respectively. Then

$$t_{DC} = 2 \cdot t_s + m \cdot (t_{MUX} + t_a + t_x + t_{LATCH}). \qquad (3\text{-}7)$$

The control signals for the MUX's and latches inside a DC are not the same as those controlling MUX's and latches in the external computing array. The two-phase non-overlapping clock signals of the latches in the DC, ψ_1 and ψ_2, must be at a frequency at least m times faster than $1/t_{op}$ such that m iteration steps, plus the shifting, can be done within t_{op}. Specifically, this frequency, f_L, can be obtained from Equation 3-6 and Equation 3-7.

The signal, β, controlling the MUX's inside the DC must be synchronous to that for the latches as described above. At every first cycle within t_{op}, L_1 lines must be selected in order to pass the initial operands, N' and $(1-\delta)$ (See Figure 3.4). Then, during the remaining cycles, L_2 lines must be selected instead. Therefore, assuming relative high voltage is "true" or "1", the control signal for these MUX's can be veiwed as a pulse signal with a pulse width approximately equal to the reciprocal of the frequency of the control signal of the latches, or $1/f_L$.

3.2.4.2 External Computing Array Timing

The rows of latches amid the PE's, as well as inside the stack section, oversee the operand timing between each row of PE's (See Figure 3.2). These latches are controlled by two-phase non-overlapping clock signals, ϕ_1 and ϕ_2. The time period of these signals, should be approximately equal to the maximum segment time, t_{op}, which implies that the frequency of the clock is $1/t_{op}$.

As a result, based on the above assumption, an approximate function block timing diagram is shown in Figure 3.5. In this figure, ψ and β are the control signals for the latches and MUX's inside the DC, and ϕ and α are those for the latches and MUX's outside the DC, respectively. Notice that 5 iteration steps of multiplication inside the DC (word size of 32 bits) is assumed for illustration.

Finally, the overall throughput time depending on the bandwidth of the system matrix, B, is derived. As shown in Table 3.1, the time required for triangulating an Nx(N+1) matrix with B=3 is 2N+2B+1 or 2N+7 time slots. This is true for all odd B. However, for all even B, the time required for triangulating the same system is 2N+2B. This is because an extra row of latches inside the stack section is removed (See Section 3.2.2). Therefore, a general equation expressing the overall througput time, T(N,B), is

$$T(N,B) = t_{op} \cdot (2 \cdot N + B + 2 \cdot \lceil B/2 \rceil). \qquad (3\text{-}8)$$

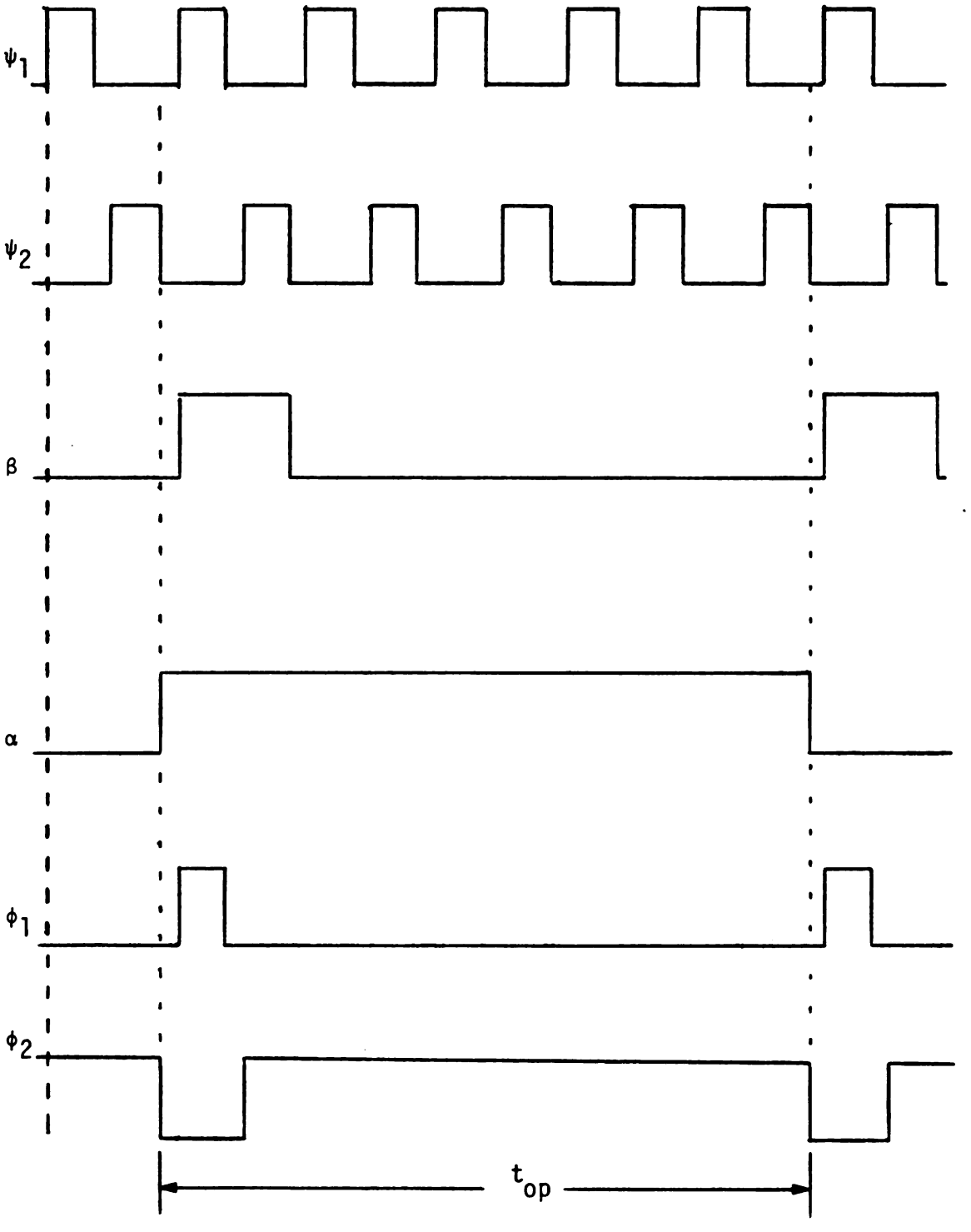


Figure 3.5 Latch and multiplexer control signals timing diagram.

3.3 Fixed-Point Consideration for Load Flow Convergence

The computing structure presented in this thesis is basically constrained to a fixed-point binary number system due to the nature of its processing element designs dictated by available chip real estate. However, an approach circumventing a true floating-point solution is to consider processing large fixed-point numbers which are pre-adjusted mantissas of previous floating-point coefficients.

In order to allow the PE's of the computing array to operate in a common fixed-point number system, the selected position for the radix point is at the left extreme of the magnitude position of the number. Thus, the radix point lies between the sign bit and most significant bit which dictates that all fixed-point numbers be strictly less than one.

In adjusting floating-point numbers to this fixed-point scheme the host computer or an intermediate processor must first sort out the maximum operand among those that will be sent to the processing array. The mantissa value of this maximum operand is normalized and the exponent value is stored. Then, all of the operands are pre-adjusted in floating-point to the same exponent value as the maximum operand. Next, the mantissa of each operand is truncated according to the number of fixed-point bits defined by the array structure design. Upon return of the upper triangle and right-hand-side vector elements, post-adjustment or post-normalization is done by the host computer or the intermediate processor according to the retained fixed exponent value.

Although this approach and the ultimate convergence of the overall non-linear system are extremely vulnerable to the dynamic range of the input matrix entries, the approach still can be applied to the system matrix derived from load flow analysis if all of its coefficients are well tempered and of tight dynamic range. Since the number of fixed-point bits of each word governs this dynamic range, a study is made to determine the minimum number of bits for which load flow convergence is obtained.

First, a subroutine, called NORMAL, is designed and incorporated into the MSU load flow program version. This subroutine serves to normalize all numbers that are involved in the calculation of the upper triangle and right-hand-side vector elements. These numbers are actually the elements of the "working rows" described in Section 2.2.1, plus any numbers that are directly or indirectly calculated from these elements. Specifically, these elements of the working rows are H, N, J, L, ΔP and ΔQ as depicted in Equation 2.2.

The main function of the subroutine NORMAL is to simulate the effect of fixed-point round off. Whenever a number is transferred from ELIM to this subroutine, the number is first converted to a fixed-point binary format and then truncated to a predefined number of bits. Finally, this truncated binary number is converted back to the original representation and returned thus incorporating the numerical effect of truncation.

The whole program (MSU load flow program version with edited ELIM and annexed NORMAL) is then individually executed with networks consisting of 43, 49, 105 and 150 busses. For each network, the program

is consecutively executed varying the number of fixed-point bits and determining whether or not convergence results. Too few bits cause undue rounding error and lead to divergence of the overall Newton-Raphson formulation. Therefore, the purpose of the above procedure is to determine the minimum necessary number of fixed-point bits (word size) required for convergence. Subsequently, this number will be used as the parameter in the simulation of the VLSI array structure.

CHAPTER IV

A PROPOSED HOST PROCESSOR/PROCESSING ARRAY INTERFACE

The most effective near term approach for enhancing the load flow computation speed is to reduce the solution time of the system equations derived from the Newton-Raphson formulation. This can be achieved by attaching the VLSI systolic array processor presented in the previous chapter to a serial host computer such that the triangulation is efficiently executed in a parallel pipeline fashion. Thus, a designate goal of this research is to simulate the interface of these two computing structures and quantify the enhanced processing throughput of load flow solution.

Generally, incompatibility arises when two computing systems, differing in number format or representation, are assembled. For example, the proposed VLSI array structure is limited to two's complement fixed-point fractional binary number operation while the host computer used in this project, the MSU CYBER-750, is basically a one's complement floating-point binary number computing system. In addition, the position of the radix point of floating-point numbers inside the CYBER-750 is at the right extreme of the mantissa. Therefore, pre- and post-adjustments of the data (operands) transferring back and forth between these two systems must be made to avoid a modification or redesign of either system.

As described in Section 3.3, the operand adjustment can be done by either the host computer or an intermediate processor. More details of these two methods are discussed in the following sections.

4.1 Operand Adjustment by Host Processor

An obvious advantage of using the mainframe computer for operand adjustment is a savings of additional hardware cost. This is because the VLSI array processor can be attached peripherally to the host computer by simple interconnection with data and control lines. However, this is under the assumption that the host computer can provide an adequately fast memory access through an operational DMA channel such that data transfer time between the host computer and the VLSI processor is less than the I/O time of the VLSI array.

First, the structure of the load flow program must be reviewed. Inside the subroutine MATGEN, the program starts looping over all the nodes in the system. A "working row" is generated for each node and then eliminated (triangulated) by calling subroutine ELIM. The elements of this working row are actually H's, J's, N's, L's, ΔP's and ΔQ's as depicted in Equation 2-2, which are exactly the elements of the augmented matrix. Therefore, this working row can at most be composed of two single rows of elements. The column size of the Jacobian matrix is, at most, equal to 2N' for a system with N'+1 busses including the slack bus. Thus, the maximum number of single elements is 2N'x2N' plus 2N' right-hand-side (augmented) vector elements. If this system is

reduced to a bandwidth of B', the maximum number of elements will become $2N'(4B'+3)-(2B'+1)(2B'+2)$ plus 2N' augmented vector elements. In other words, the upper bound Jacobian matrix bandwidth, B, and dimension, N, are expressed as follows:

$$B = 2 \cdot B' + 1, \tag{4-1}$$

$$N = 2 \cdot N'. \tag{4-2}$$

The upper bound matrix full breadth then becomes 4B'+3.

Since the elimination (triangular factorization only) procedure is carried out for each node, it can be described as a piece-wise elimination process. Therefore, the total elimination time is the sum of all the nodal processing times. Also, the operation of finding the operand of maximum absolute value can be started when each working row is generated.

Finally, according to the normalization procedure described in Section 3.3, the pre- and post-adjustments of the operands carried out by the host computer, the CYBER-750, can be summarized in the program flowcharts illustrated in Figure 4.1 and Figure 4.2. The number of the operands which will be normalized is the only parameter in the program. Unfortunately, as described before, this number is directly proportional to NxB. Therefore, the time required for the normalization increases as the size of the network increases, which will eventually slow down the overall solution throughput.

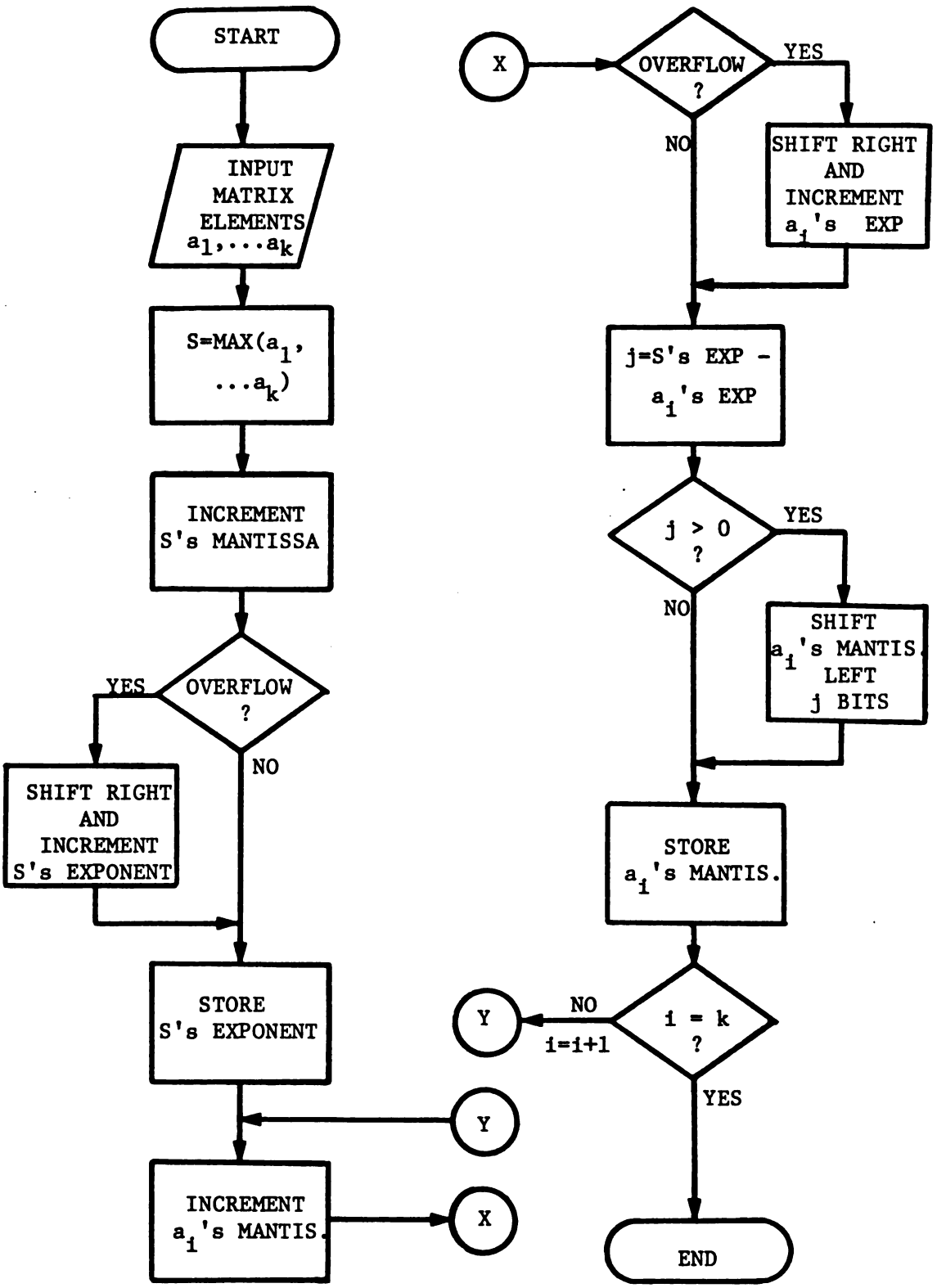


Figure 4.1 Program flowchart for operand pre-adjustment.

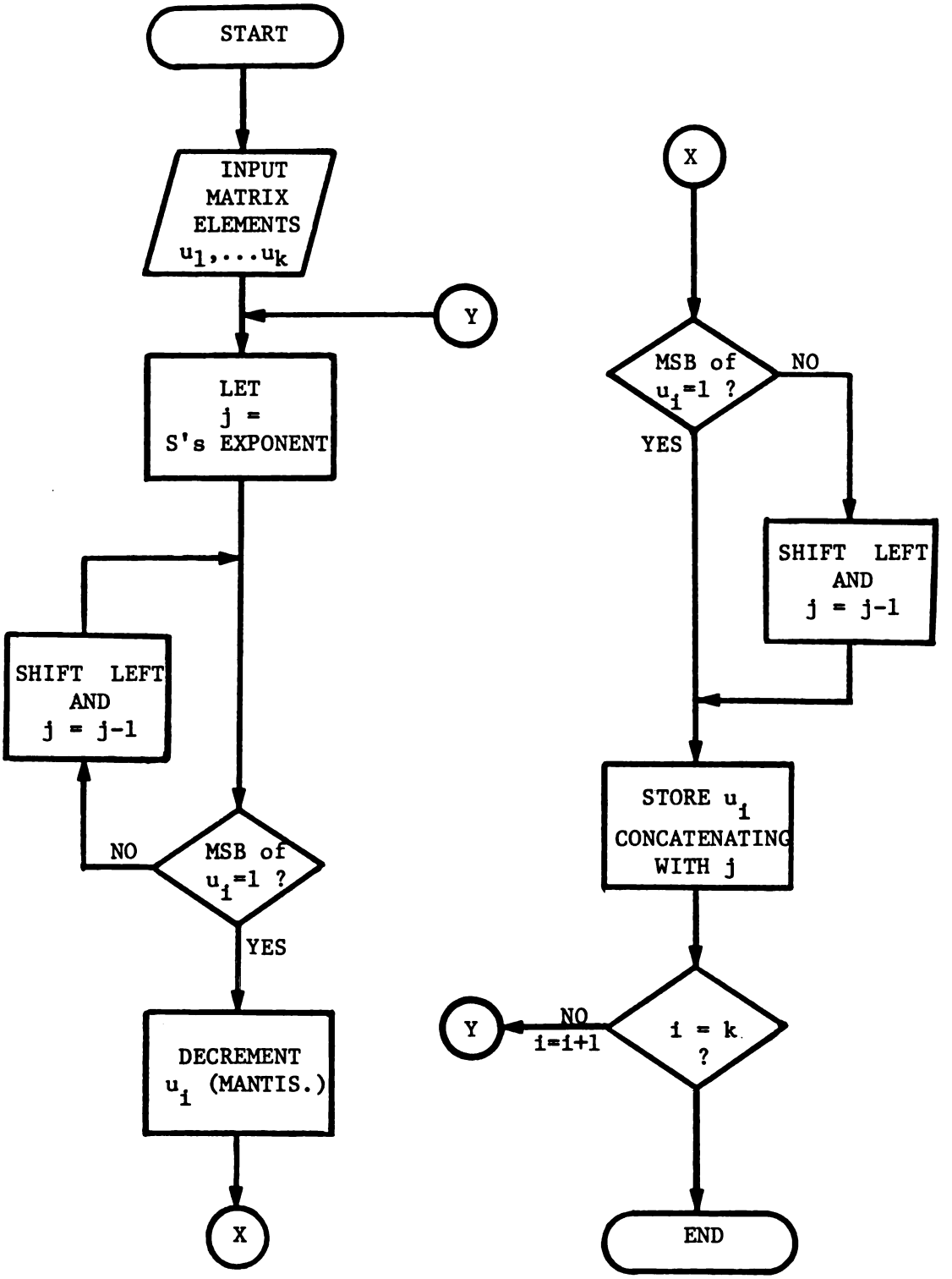


Figure 4.2 Program flowchart for operand post-adjustment.

4.2 Operand Adjustment by Intermediate Processor

In this section, a hardware interface system between a host computer and the VLSI array processor is discussed in terms of function block operation. This interface, an intermediate processor, is responsible for the adjustments described before. It is assumed here that the operation of finding the maximum absolute operand is done by the host computer during the procedure of generating the Jacobian matrix. Therefore, the intermediate processor only serves to pre- and post-adjust the operands. The advantage of using this intermediate processor is a savings of time when compared to the method explained in the previous section.

The overall block structure of the intermediate processor, which operates in pipeline fashion, is shown in Figure 4.3. In this figure, the mantissa and exponent busses have m and e lines, respectively. The number m must be equal to or greater than the number of fixed-point bits, n, of the operand width inside the VLSI array processor. If m is greater than n, these n lines (input to and output from the array processor) must be connected to the most significant bit (MSB) positions (including the sign bit) of m lines. Likewise, the m lines input from and output to the host computer will be connected to the MSB's of the mantissa bus lines inside the host computer. The number e is the number of bits in the exponent field of the host computer. The left half portion of the structure performs the pre-adjustment of the operands

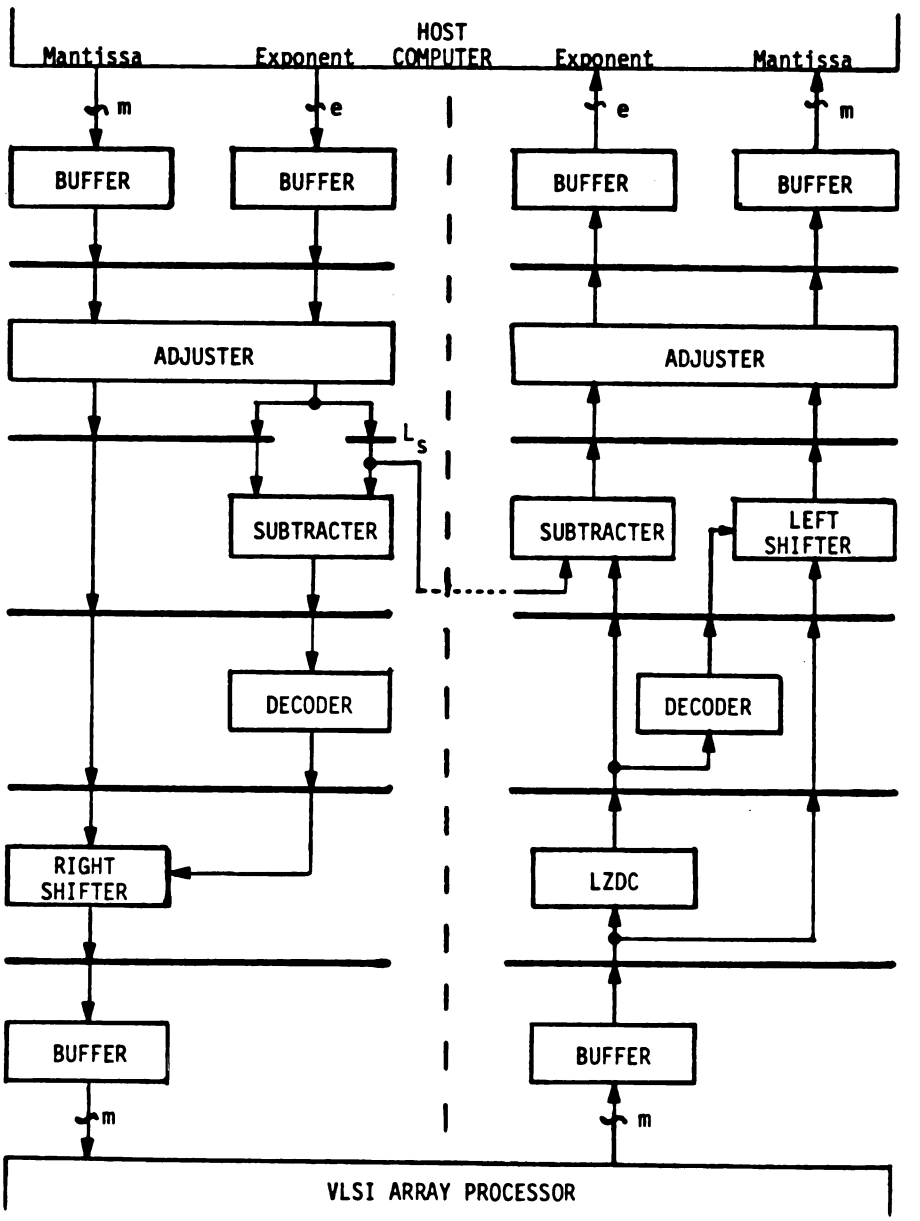


Figure 4.3 Function block structure of an intermediate processor for pre- and post-adjustment.

traveling from the host computer. The right half portion operates the post-adjustment of the operands resulting from the VLSI processor.

Both of the adjusters depicted in the same figure function as a number format adapter. Therefore, the internal structure is dependent on the format of the input and output operands. The adjuster inside the left half portion converts the mantissa of an input operand in any format to a two's complement form. For example, if the normalized mantissa and exponent of the input operands are in one's complement representation (e.g. from the CYBER-750) and the radix point is at either the right or left extreme of the magnitude of the mantissa, the adjuster must include an incrementer for converting the mantissa to two's complement form. No increment of the exponent is required because the exponent will not be transferred to the VLSI processor. Thus, the exponent can be in either biased or unbiased form. In case an overflow results after the mantissa's increment, the whole mantissa except the sign bit will be shifted right once with trailing "0" (if the number is positive) or "1" (if negative) and its corresponding exponent value must be incremented by one. A possible design of the adjuster is illustrated in Figure 4.4 based on the pipeline concept.

If, however, the input operands are in two's complement format, this portion of the adjuster can be omitted. If the input operands are not in a normalized form, a leading-zero detection circuit (LZDC), a decoder and a left shifter for the mantissa and a subtracter for the exponent may be required additionally. Even in this case the adjuster can still be divided into pipeline segments. Therefore, it is assumed that the adjuster can always be designed in a pipeline structure which

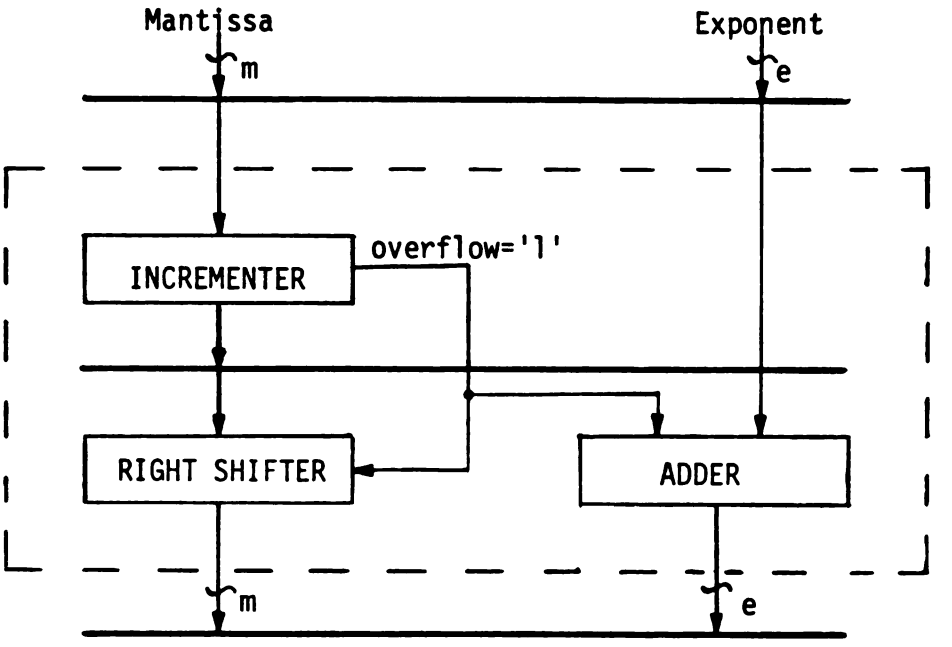


Figure 4.4 A sample internal block structure of an adjuster.

is crucial to avoiding a potential interface bottleneck.

The adjuster inside the right half portion of the processor always changes the input mantissa from two's complement form back into the original number format. For the same example of one's complement format, the input two's complement mantissa is initially decremented and then a test determines if the MSB next to the sign bit is still "1". If not, the mantissa without the sign bit will be shifted left once with trailing "0" or "1" and the corresponding exponent value must be decremented by one. The structure of this adjuster is thus composed of a decrementer, a subtracter (or another decrementer) and a left shifter which are interconnected similar to the structure shown in Figure 4.4.

During the pre-adjustment, the maximum operand found beforehand appears first followed by other operands in row-ordered form. These operands are pumped into the pipeline structure through the receiving buffers. At the second stage, the latches marked as L_s in Figure 4.3 have the same structure as that of the others, but they have a different input control. Once the very first exponent, the exponent of the maximum operand, has been input into these latches, no other exponents can be put into them. Therefore, this exponent will be kept stable by constant refreshing for use as the subtrahend for the subtracter in the next stage as well as in the post-adjustment portion. As a result, the maximum exponent is consistantly subtracted from all other exponents.

Finally, before all of the adjusted mantissas, except the one of the maximum operand, are pumped into the VLSI processor through the output buffer, they are shifted right (with trailing "0" or "1") corresponding to the result obtained from the exponent's subtraction.

During the post-normalization, triangulated results are returned from the array processor through a receiving buffer. These results are pumped into a leading-zero detection circuit (LZDC). Subsequently, the result of the detection is used as a minuend and decoded as a control signal to be sent to a subtracter and a left shifter. Inside the shifter, the mantissa is shifted with trailing "0" or "1" again depending on its sign. Eventually, after the shifted mantissas and subtracted exponents have been adjusted back to the original number format, they are transferred to the host computer through the output buffers. All of these output buffers as well as the receiving buffers can be first-in-first-out memory buffers if necessary for speed matching

(e.g. if a DMA channel is operational).

Latches control the timing of the process in the same way as those in the VLSI array structure. Therefore, the processing time, T_p, can be described by the standard pipeline processing equation,

$$T_p = t_s + (n-1) \cdot t_{op}, \tag{4-3}$$

where t_s = setup time,

t_{op} = longest segment time,

n = number of operands.

The segment time, t_{op}, can be determined from the structure shown in Figure 4.3. Assuming the adjuster can be pipelined (See Figure 4.4), all of the shifters are barrel shifters, subtraction is pipelined into a complementation and addition, and, the decoder and the LZDC are sufficiently fast (a total 5 unit gate delay LZDC has been described by Chang and Fisher [21]), t_{op} is simply the addition time.

As a whole, performing the pre- and post-adjustments of the operands by this intermediate processor is better than utilizing the host computer from the view point of data transfer time and arrangement. This is because execution in this processor is pipelined.

CHAPTER V

SIMULATION DEVELOPMENT

The main purpose of the circuit simulation is to quantify the overall triangulation time as well as the chip area of the improved VLSI array structure including I/O circuits. The triangulation portion represents the actual triangular factorization encountered in the load flow calculation except that fixed-point arithmetic is employed. Therefore, the piece-wise elimination time as discussed in Section 4.1 will be replaced by the simulated processing time.

The estimation of the chip area, though it does not affect the triangulating process, will project a basic trade-off comparison of chip size versus matrix bandwidth.

Several fundamental parameters that may be manipulated in the simulation are minimum lithographic linewidth (λ), word size, matrix or network bandwidth (B or B'), and matrix or network dimension (N or N'+1).

In particular, the manipulation of λ can track current trends in I. C. fabrication technology and thus projects the reality of the implementation of the proposed VLSI systolic array algorithm. A "λ model", a layout geometry design tool introduced by Mead and Conway [18], assumes that the permissible layout linewidths and spaces along the diffusion, polysilicon and metal lines can all be scaled down in linear dimension. Therefore, the geometries of layout elements can be

described proportionally in terms of current and projected minimum lithographic linewidth. As the I. C. fabrication technology advances, λ decreases and thus the layout geometries also decrease proportionally.

In addition to the λ model, a revised "τ model", a simple arithmetic model for computing propagation delay of a transistor level circuit, is used to relate the switching speed of a gate and overall processing time of triangulation [4, 18]. The original τ 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 [18]. However, the transit time, τ, is the fundamental limit on the switching speed of the gate. In a practical circuit, the speed of MOS (Metal-Oxide-Semiconductor) device operation is determined by the speed with which capacitors can be charged and discharged [22]. Therefore, the revised model assumes that "T" is the discharge time for a basic inverter 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 calculation are more realistic than the τ model since it uses discharge time, T, not just transit time, τ. Moreover, this simplifies the process of estimating total load capacitance [4].

The number of bits per word (word size) is determined by the necessary number of fixed-point bits required for load flow convergence (See Section 3.3). The Jacobian matrix bandwidth and dimension, however, are not the same as the network bandwidth and dimension. As mentioned in Section 4.1, corresponding to a power utility network with

a system bandwidth B' and bus dimension N'+1 the upper bound Jacobian matrix bandwidth and dimensions, B and NxN, are given by 2B'+1 and 2N'x2N', respectively. This is of course due to multiple entries (up to 2x2) for the particular partial derivatives. The upper bound dimensions of the right-hand-side vector then becomes 2N'x1.

5.1 Chip Area Computation

The major hardware design of the VLSI array processor in this research is partitioned into three distinct parts, namely computing structure, input and output circuits.

The design of I/O circuits for the array structure is based on the choice from several I/O circuit candidates [17]. The input circuit, cataloged as a "data controlled" circuit, pumps the data operands into the destination PE's without using any channel-selecting control signals. It ultilizes the FIFO stack concept employing n sets (same as the number of bits per word) of N_{in}-stage shift register chains (where N_{in} is the number of input ports or 2B+2). Therefore, this circuit is composed of N_{in} arrays of 1-bit shift registers.

The output circuit, cataloged as an "SCS" (Shift-Register Control Sequence) circuit, is controlled by an SCS signal. This sequence, a single "1" followed by B "0's", is synchronously pumped into a 1-bit shift register chain and is used to select the proper operand output channel. When the N_{out} data operands, where N_{out} is the number of output ports or B+1 (See Figure 3.2), are ready to be clocked out of the array structure, N_{out} output channels are sequentially selected for

routing the operands. Thus, this output circuit consists of N_{out} 1-bit shift registers plus N_{out} arrays of buffers and pass transistors. Finer details of these I/O circuits can be found in [17].

Since the layout of processing elements (PE's), multiplexers and latches in the computing array is regularly connected and supports local communication (i.e., nearest neighbors), it is possible to model each unit as a functional module and tesselate these modules into large building blocks. Also, the fundamental building blocks, such as full-adders in the PE's, can be modeled and likewise tesselated. In the same manner, the I/O circuits are modeled as unit modules such as 1-bit shift registers and buffers.

As a result, the I/O circuits and computing structure taken together require that relatively few specific modules be laid out by hand and quantified (with respect to the parameter λ) as per the technique of Mead and Conway. Specifically, these modules include a full-adder, 1-bit shift register, buffer, latch, 2-input NAND gate, 2:1 multiplexer and pass transistor.

5.2 Throughput Computation

The localized connectivity of the computing and I/O structure allows straightforward computation of delay parameters. Quantification of module delay involved summing the active gates' propagation time, with consideration of loading due to fan-out, and internal communication path delay. Diffusion line length and unit length delay were considered the significant path delay parameters. Moreover, intermodule line

length was negligible due to the tesselated nature of the structures.

Utilizing the revised τ model, the active gate delay time is based on T. It is assumed that T is linearly dependent on λ and when λ=3 microns, T=0.6 nsec [18]. An upper bound on the diffusion line lengths were determined for each module as a function of line length squared. Then, using the fact that transit time is 100 nsec for a 10 millimeter length line [18], communication path delays were obtained.

Finally, the total module delay is the sum of active gate delays and communication path delays. Thus, once the maximum segment delay is found, the total processing time (throughput) for triangulating a band system matrix can be obtained by using the Equation 3-8. However, the I/O delay time must be considered and taken into the account of throughput calculation. Therefore, assuming no I/O bottleneck, the overall throughput time including I/O time, t_{in} and t_{out}, is estimated by modifing Equation 3-8 and is shown as below.

$$T(N,B) = t_{in} + t_{op} \cdot (2 \cdot N + B + 2 \cdot \lceil B/2 \rceil) + t_{out}, \qquad (5\text{-}1)$$

where t_{in} and t_{out} are also known as the pipeline setup time and flush time, respectively.

5.3 Significant Module Data

Chip area and propagation delay computational methods have been discussed in the previous sections. Based on the data from [4], 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
1-Bit Shift Register	23 x 17	18
Buffer	23 x 17	2
Latch	23 x 17	13
2-Input NAND Gate	19 x 5	8
2:1 Multiplexer	6 x 6	1
Pass Transistor	3 x 3	1

CHAPTER VI

RESULTS AND CONCLUSIONS

The minimum number of fixed-point bits per word required for load flow convergence of several different bus networks as determined by running the revised MSU load flow program version described in Section 3.3 will be presented first. As explained in that section, the convergence of the overall Newton-Raphson formulation in the revised program is limited by the number of fixed-point bits (word size). Therefore, by varying this number, several load flow program results are obtained and shown in Table 6.1. This table indicates that the minimum numbers of bits per word required for the convergence of the test networks are 28 and 32.

As a second step, the circuit simulations comparing the throughput speed and chip size versus matrix bandwidth are presented. The time results selected from the simulations are compared with those from previous works to promote the significance of using a VLSI systolic computing array structure for load flow computation.

Finally, conclusions will be drawn and summarized from all of the results obtained from this research.

Table 6.1 Convergence results in number of iterations of MSU load flow program version.

Network Size		Results of Convergence*		
		Original MSU Program Version	Revised MSU Program Version	
Busses	Band-width	Number of Iterations	Number of Fixed-Point Bits per Word	Number of Iterations
43	6	6	16	12
			28	6
49	11	4	16	Divergent
			28	5
			32	4
105	17	6	28	6
			32	6
150	16	6	28	7
			32	6

*Tolerant Real Power Mismatch (PMM) = 0.15
Tolerant Reactive Power Mismatch (QMM) = 0.25

6.1 Circuit Simulation Results

It has been predicted that the patterning resolution limitation of optical lithography will be about 0.5 micron in the early twenty-first century [23]. Recently, however, many I. C. designers have been exploring new and promising techniques including electron-beam and X-ray lithography. With electron-beam methods, for example, future linewidths of 0.2 and 0.125 microns have been estimated as feasible [24, 25]. For these reasons, 0.8, 0.5 and 0.2 micron linewidths have been chosen as the realistic goals of mid and late 1980's and early 1990's VLSI capability, and are used as quantification parameters in the simulations. Another parameter used throughout the simulations is the word size of 32 bits, which is chosen in order to assure convergence of all test networks.

First, the results of a general circuit simulation with the selected parameters and the assumption of B=0.1N (where B and N are matrix bandwidth and dimension, respectively) are illustrated graphically in Figure 6.1 and Figure 6.2.

Figure 6.1 contains three curves showing entire chip edge size versus matrix bandwidth with linewidths of 0.8, 0.5 and 0.2 microns. A projection of a maximum technologically feasible chip size has been made which predicts the future I. C. chip edge to be about 1.5 cm by the late 1980's [23]. Under this assumption, using a 0.5 micron linewidth the matrix bandwidth, B, is limited to about 13. To extrapolate this result

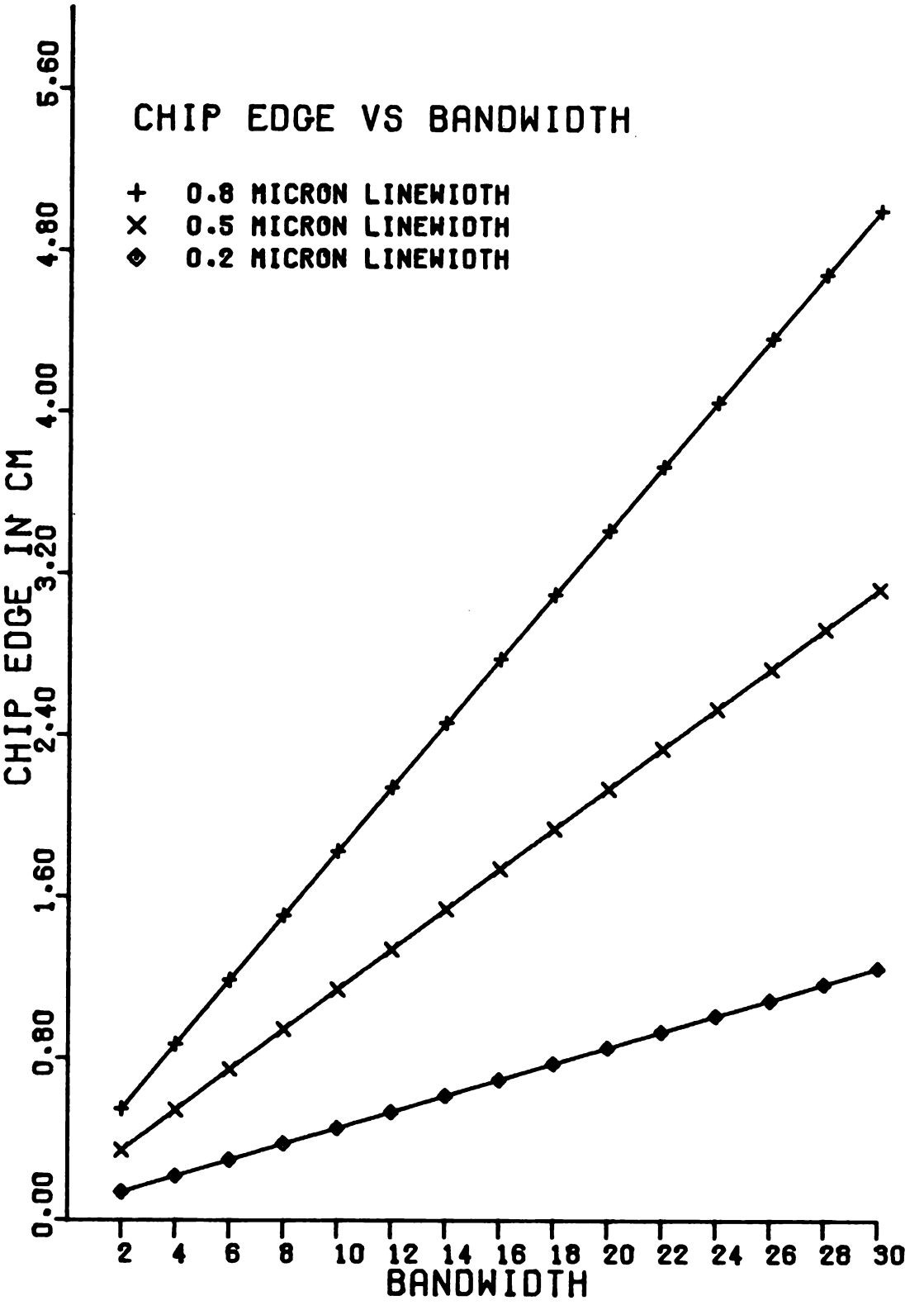


Figure 6.1 Entire chip edge size versus matrix bandwidth for 0.8, 0.5, and 0.2 micron linewidths.

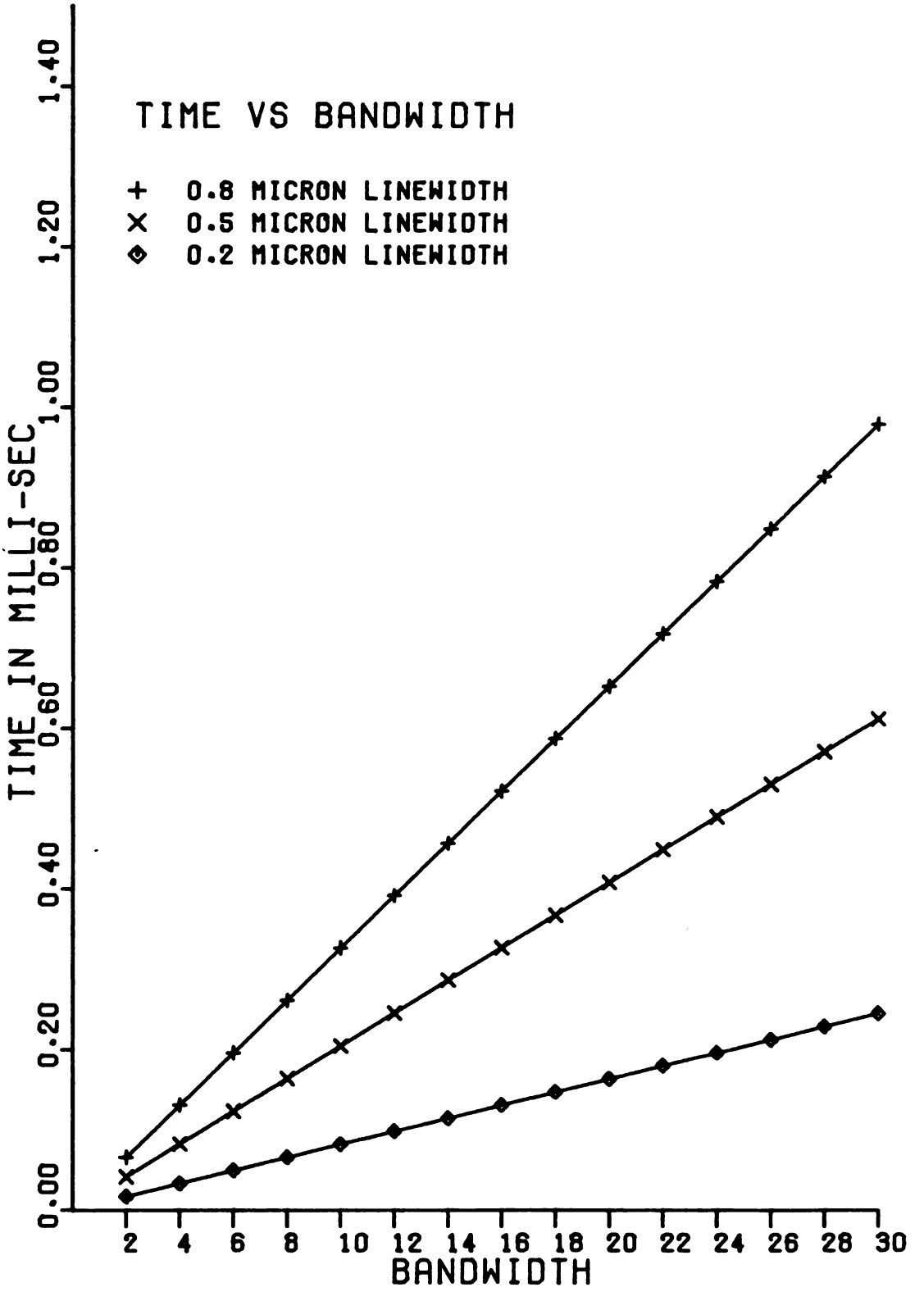


Figure 6.2 Entire chip propagation delay versus matrix bandwidth for 0.8, 0.5, and 0.2 micron linewidths.

to an actual electrical utility network size, an additional aspect must be first considered. As mentioned in Section 4.1, the upper bound of the Jacobian matrix bandwidth and dimension are 2B'+1 and 2N', where B' and N'+1 are the utility network bandwidth and dimension, respectively. Thus, the result indicates that a power system of approximately 61 busses could be solved on a single chip assuming late 1980's technology (0.5 micron linewidth). And, if early 1990's technology (0.2 micron linewidth) with the same chip edge size is assumed, systems of about 171 busses could be solved on one chip.

Of greater significance is the fact that the triangulation time, or total chip propagation delay, is dramatically improved over traditional serial techniques. As depicted in the graph of time versus bandwidth (Figure 6.2), using the same 0.5 and 0.2 micron linewidths as before, the triangulation times of the networks of 61 and 171 busses are 0.266 and 0.286 milliseconds, respectively.

Next, by using the details of the simulation time results of several major block structures, it can be shown that this type of systolic array architecture will not cause a serious I/O bottleneck of operands. As shown in Table 6.2, it is unquestionable that the DC segment time (t_{DC}) completely dominates over the MAC time (t_{MAC}), INPORT time (t_{in}) and OUTPORT time (t_{out}) at the given bandwidths and linewidths. The INPORT time is actually the time required to set up the operands for the first row of input latches inside the computing structure (See Figure 3.2). Therefore, this INPORT time is proportional to the number of input ports or 2B+2. Likewise, the OUTPORT time varies with the number of output ports or B+1. However, the DC time as well as

Table 6.2 Simulation time results of I/O ports, MAC and DC.

B	λ (micron)	t_{in} (nsec)	t_{out} (nsec)	t_{MAC} (nsec)	t_{DC} (nsec)
30	0.8	178.6	94.7	293.8	1480.0
	0.5	111.6	59.2	183.6	925.0
	0.2	44.6	23.7	73.4	370.0
20	0.8	121.0	65.9	293.8	1480.0
	0.5	75.6	41.2	183.6	925.0
	0.2	30.2	16.5	73.4	370.0
10	0.8	63.4	37.1	293.8	1480.0
	0.5	39.6	23.2	183.6	925.0
	0.2	15.8	9.3	73.4	370.0
5	0.8	34.6	22.7	293.8	1480.0
	0.5	21.6	14.2	183.6	925.0
	0.2	8.6	5.7	73.4	370.0

the MAC time is dependent on the word size but not B. As a result, an I/O bottleneck will happen if and only if B is large enough to make $t_{in} > t_{DC}$. As shown in Table 6.3, this I/O bottleneck occurs when B is greater than 255 which implies B' and N'+1 are equal to 127 and 1271, respectively. The bottleneck, though, never occurs in this work since only networks of less than or equal to 150 busses are studied.

Now, utilizing the same simulation program, some practical results tailored towards the load flow study are obtained by specifing the exact dimensions and bandwidths of some typical power systems. The simulation time results, however, when used to replace the complete piece-wise elimination time, must include the time spent on the pre- and post-normalization of operands.

These normalization times are found according to the intermediate processor design discussed in Section 4.2. As shown in Figure 4.3 and Figure 4.4, the numbers of stages (including the buffer stages and the complementation stage for subtracters) in the pipeline structure are 8 (pre-adjustment portion) and 10 (post-adjustment portion). Assuming the adder is a 3-level CLA (Carry Lookahead Adder) which has a 12 unit gate delay [20], the pre- and post-normalization times are simply the setup time of the pre-adjustment portion and the flush time of the post-adjustment portion of the intermediate processor. Therefore, the pre-normalization time (t_{pre}) is given by,

$$t_{pre} = 9 \times 12 \cdot \Delta \qquad (6\text{-}1)$$

where Δ is assumed to be 1 nsec with mid 1980's technology [23]. An

Table 6.3 Simulation time results showing an I/O timing bottleneck.

B	λ (micron)	t_{in} (nsec)	t_{out} (nsec)	t_{MAC} (nsec)	t_{DC} (nsec)
260	0.8	1503.4	757.1	293.8	1480.0
	0.5	939.6	473.2	183.6	925.0
	0.2	375.8	189.3	73.4	370.0
256	0.8	1480.3	745.6	293.8	1480.0
	0.5	925.2	466.0	183.6	925.0
	0.2	370.1	186.4	73.4	370.0
255	0.8	1474.6	742.7	293.8	1480.0
	0.5	921.6	464.2	183.6	925.0
	0.2	368.6	185.7	73.4	370.0
250	0.8	1445.8	728.3	293.8	1480.0
	0.5	903.6	455.2	183.6	925.0
	0.2	361.4	182.1	73.4	370.0

extra segment time is included in t_{pre} because the very first operand pumped into the intermediate processor is the maximum operand which will not be transferred to the VLSI processor. The post-normalization time (t_{post}) is defined by,

$$t_{post} = 10 \times 12 \cdot \Delta. \tag{6-2}$$

Finally, t_{pre}, t_{post}, the VLSI array computing time including I/O time (t_{comp}), overall solution time ($t_{VLSI}=t_{pre}+t_{post}+t_{comp}$) and the corresponding chip edge size of the VLSI structure (S_{VLSI}) are shown in Table 6.4. All of these time results are found with respect to the network bandwidth (B'), network dimension (N'+1) and various lithographic linewidths. This table indicates that t_{pre} and t_{post} are insignificant when compared to t_{comp}. The solution time, t_{VLSI}, will be compared to other results in the next section. Notice that some of the chip edge sizes, S_{VLSI}, are tremendously large compared to that of the projected 1.5 cm x 1.5 cm I. C. chip.

6.2 Comparison with Previous Results

Time results obtained from the previous works which utilized a one-dimensional array processor (AP-190L) to solve the same power systems are compared with the benchmark results from the CYBER-750 and the simulation results in this research. This is illustrated in Table 6.5. This table shows that the triangulation time per iteration spent on the VLSI systolic array structure is dramatically improved over

Table 6.4 Simulation time and chip edge size results for load flow study.

N'+1	B'	λ (micron)	t_{pre} (nsec)	+ t_{post} (nsec)	+ t_{comp} (msec)	= t_{VLSI} (msec)	S_{VLSI} (cm)
43	6	0.8	108	120	0.289	0.289	2.30
		0.5	108	120	0.181	0.181	1.44
		0.2	108	120	0.072	0.072	0.58
49	11	0.8	108	120	0.355	0.355	3.89
		0.5	108	120	0.222	0.222	2.43
		0.2	108	120	0.089	0.089	0.97
105	17	0.8	108	120	0.722	0.722	5.80
		0.5	108	120	0.451	0.451	3.62
		0.2	108	120	0.180	0.180	1.45
150	16	0.8	108	120	0.983	0.983	5.48
		0.5	108	120	0.614	0.614	3.43
		0.2	108	120	0.246	0.246	1.37

Table 6.5 Comparison of triangulation time per iteration of different processors.

N'+1	B'	t_{CYBER}* (msec)	t_{AP} (msec)	t_{VLSI} (λ) (msec (micron))
43	6	5.500	2.717	0.289 (0.8)
				0.181 (0.5)
				0.072 (0.2)
49	11	10.500	4.555	0.355 (0.8)
				0.222 (0.5)
				0.089 (0.2)
105	17	77.500	13.163	0.722 (0.8)
				0.451 (0.5)
				0.180 (0.2)
150	16	55.833	17.646	0.983 (0.8)
				0.614 (0.5)
				0.246 (0.2)

*Average from several runs with Markowitz ordered data sets (Tinney No. 2 ordering [5,6]).
Numbers are absolute lower bounds; real CPU time is considerably higher but could not be accurately trapped.

the benchmark result (t_{CYBER}) and that spent on the array processor (t_{AP}). t_{CYBER} is tremendously large since the program is executed in a serial fashion. Therefore, it is concluded that the VLSI array structure is very efficient in reducing the banded Jacobian matrix triangulation time which in turn enhances the overall computational throughput of large scale load flow problems. In summary, the ratios of $t_{VLSI}:t_{CYBER}$ and $t_{VLSI}:t_{AP}$ (for λ=0.8 micron) are listed in the following table.

Table 6.6 Comparison of time results of different processors in ratio.

N'+1	B'	$t_{VLSI} : t_{CYBER}$	$t_{VLSI} : t_{AP}$
43	6	1 : 19.0	1 : 9.4
49	11	1 : 29.6	1 : 12.8
105	17	1 : 107.3	1 : 18.2
150	16	1 : 56.8	1 : 18.0

6.3 Conclusions

In this work, an existing VLSI algorithm for triangulating large band form matrices was successfully modified and applied to load flow analysis. By adding rows of multiplexers and latches to an existing algorithm, the revised structure can triangulate a band matrix to strict upper triangular matrix with a unit diagonal in O(N) time steps.

The pre- and post-adjustments of the operands transferring between a host computer and the VLSI array structure can be done efficiently by the proposed intermediate pipeline processor described in Section 4.2. The time required for operand adjustment is simply the pipeline setup and flush time which is insignificant to the overall triangulation time.

It is found that the minimum number of fixed-point bits per word for load flow convergence of the test networks used (< 150 busses) is 32 which indicates that all of the coefficients of the system matrix are reasonably well tempered and of adequately tight dynamic range.

The simulation results show that the I/O bottleneck does not occur in this VLSI structure if the maximum segment delay time in the pipeline is completely dominated by the PE time, specifically the DC time.

As shown in Section 6.1, a power system network of approximately 61 busses could be solved on a single VLSI chip with the late 1980's VLSI technology.

The circuit simulation time results reveal that the time of upper triangulation by the designed VLSI systolic array structure is greatly

reduced and hence the overall solution throughput of load flow analysis is improved.

6.4 Discussion

It has been shown that when the system bandwidth is large, the proposed VLSI structure cannot be implemented on a single chip. Therefore, this algorithm must be realized in a modular fashion on a few chips. If such a case is possible, the VLSI algorithm should be modified at the same time to have true floating-point capabilities since there will be no limitation on the hardware size if the intercommunication problem among chips is solved.

Recall that the complete triangulation time of the VLSI structure is limited by the segment time of the DC (See Table 6.2) which is approximately five times greater than that of an MAC. Thus, the triangulation time can be further reduced if a faster DC is designed. However, as indicated in Section 6.1, the I/O bottleneck did not occur because of the domination of the DC segment time. So, it is possible that if the matrix bandwidth is extremely large and DC time is reduced tremendously, the INPORT time will dominate over other segment times. As a result, an I/O bottleneck may be encountered. Therefore, a trade-off between the segment and I/O time may result in limiting the overall triangulation throughput in the future modular designs.

If the VLSI algorithm is realized in a modular fashion and designed with binary floating-point (FLP) arithmetic, all of the processing elements should be reexamined. An FLP PE should possess the

capabilities of alignment or normalization, truncation and rounding of the operands. This, of course, may double or triple the hardware size making a modular layout essential. In addition, the complexity of the cell design depends on other factors, such as degree of precision and error bounds. This may introduce trade-offs between normalized or unnormalized FLP designs. Undoubtedly, the inclusion of FLP arithmetic will involve a huge increase in hardware size. Thus, a near term research requirement is to investigate some area-efficient FLP arithmetic algorithms (multiply-add and division) and then examine the intercommunication problem and potential modularity among these new PE's.

Finally, it must be pointed out that the application of a proposed VLSI systolic array structure to load flow analysis leaves a lot of practical questions unanswered. For instance, in the future when such a VLSI structure is available, will it still be attached peripherally to a host computer of today's serial genre? Also, in order to obtain the fastest possible load flow solution, the structure of a load flow program should be reexamined such that the process of finding a maximum operand and coefficients' normalization can be avoided.

On the whole, as VLSI technology advances, not only should special purpose computer architectures be designed and improved, but also the algorithms, including arithmetic and programming, must be reexamined so that all of the technology, architectures and algorithms can progress harmoniously.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Gross, C. A., Power System Analysis, John Wiley and Sons, New York (1979), pp. 3-9.

2. Kung, H. T., "The Structure of Parallel Algorithms," Advances in Computers, Vol. 19, Academic Press (1980), pp. 65-112.

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

4. Hsu, W. C. and Shanblatt, M. A., Evaluation of a Single VLSI Chip Algorithm for Triangulating Large Band Form Matrices, Tech. Report No. MSU-ENGR 82-015, Michigan State University, East Lansing, Michigan (August 1982).

5. Tinney, W. F. and Hart, C. E., "Power Flow Solution by Newton Method," IEEE Trans. on Power Apparatus and Systems, Vol. PAS-86, No. 11 (November 1967), pp. 1449-1456.

6. Tinney, W. F. and Walker, J. W., "Direct Solutions of Sparse Network Equations by Optimally Ordered Triangular Factorization," Proc. IEEE, Vol. 55, NO. 11 (November 1967), pp. 1801-1809.

7. Shanblatt, M. A., Mickle, M. H. and Vogt, W. G., "Optimal Ordering Strategies for Load Flows on an Array Processor," Proc. 1981 Int'l Conf. on Electrical Energy (April 1981), pp. 78-81.

8. Abulleil, A. M., The Use of an Array Processor in the Solution of Large Scale Load Flow Problems, Ph. D. Thesis, University of Pittsburgh (1981).

9. Philadelphia Electric Company Power Flow Program, POWERFLO, Version 4, Mod Level 30, System Planning Division, Philadelphia Electric Company, Philadelphia, Pa. (8 October 1975).

10. Ralston, A., A First Course in Numerical Analysis, McGraw-Hill, New York (1965), pp. 398-415.

11. Bodewig, E., Matrix Calculus, Interscience, New York (1959), pp. 101-124.

12. Alway, G. G. and Martin, D. W., "An Algorithm for Reducing the Bandwidth of a Matrix of Symmetric Configuration," Computer Journal (August 1965), pp. 264-272.

13. 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.

14. 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.

15. Foster, M. J. and Kung, H. T., "The Design of Special-Purpose VLSI Chips," Computer, Vol. 13 (January 1980), pp. 26-40.

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

17. Hsu, W. C., Leung, Y.-Y. J. and Shanblatt, M. A., "Comparison of Input/Output Structures for Single Chip VLSI Systolic Arrays," Proc. 25th Midwest Symp. on Circ. and Sys. (August 1982).

18. Mead, C. and Conway, L., Introduction to VLSI Systems, Addison-Wesley Pub. Co., Reading, Massachusetts (1980), pp. 1-90.

19. 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.

20. Hwang, K., Computer Arithmetric, John Wiley and Sons, New York (1979), pp. 84-254.

21. Chang, T. L. and Fisher, P. D., "High-Speed Normalization and Rounding Circuits for VLSI Floating-Point Processors," Proc. IEEE Int'l Conf. on Circuits and Computers (1980), pp. 512-516.

22. Taub, H. and Schilling, D., Digital Integrated Electronics, McGraw-Hill Inc. (1977), pp. 35-53.

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

24. Eidson, J. C., "Fast Electron-Beam Lithography," IEEE Spectrum, Vol. 18 (July 1981), pp. 24-28.

25. Bernhard, R., "VLSI Lithography at the Crossroads," IEEE Spectrum, Vol. 18 (July 1981), p. 27.